Персональный компьютер БК-0010, БК-0011м (2'96)	484-54-11 (Саша, Галя)
Последний выпуск журнала	

Международная Академия Информатизации

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР БК-0010, БК-0011м

Компьютика

2'96 (14)

Издается с 1993 г.

В НОМЕРЕ

Издательство Компьютика
Москва 1996

Авторы выпуска

Зыков В. В.
Канивец И. В.
Козлов А. И.
Мальцев С. А.
Неробеев С.М.
Романенко А. Н.
Сорокин А. В.
Суханов А. В.
Усенков Д. Ю.
Храмцов А.В.

РЕДАКТОРЫ: *ВАСИЛЬЕВ Б. М.*
УСЕНКОВ Д. Ю.

Лицензия на издательскую деятельность
ЛР №064197 от 10 августа 1995 г.

Свидетельство о регистрации средства массовой информации
№ 013550 от 26 апреля 1995 г.

Почтовый адрес: 125315, Москва, а/я 17
E-Mail: mail@infoobr.msk.su
Телефон: (095) 292-53-87

Впервые после подключения к БК дисковода для этого компьютера разработана операционная система, почти полностью аналогичная широко используемой на IBM-совместимых компьютерах ОС MS-DOS. В данной статье приводится описание ее возможностей, структуры и команд, по синтаксису практически совпадающих с принятыми на PC, XT, AT и их многочисленной «родне» вплоть до Pentium.

А. В. Храмцов,
г. Армавир

DX-DOS: новая операционная система для БК

Одной из самых удачных попыток разработки операционной системы для ПЭВМ было создание ОС, впоследствии названной MS-DOS. История этой операционной системы, которая сегодня получила распространение во всем мире в количестве (по разным оценкам) от 30 до 50 млн. экземпляров, начинается со скромной ОС 86-DOS, написанной в середине 80-х гг. Т. Петерсоном для компании Seattle Computer Products. Популярность MS-DOS породила желание перенести ее и на другие типы компьютеров, в частности на БК. (Правда, из-за несовместимости системы команд процессоров семейств Intel и DEC, здесь правильное было бы говорить не о «переносе MS-DOS на БК», а о написании новой ОС с использованием идеологии, заложенной в MS-DOS.)

При создании для БК операционной системы DX-DOS была предпринята попытка реализации в ней максимума удобств, имеющихся в MS-DOS. Вместе с тем, в DX-DOS предусмотрена совместимость со всеми ранее разработанными программами, использующими системные вызовы монитора БК.

DX-DOS перехватывает прерывание EMT 36, обслуживающее работу магнитофона, и обрабатывает его своими средствами.

Возможности DX-DOS

- Файловая система, аналогичная MS-DOS, и MSX-DOS.
- Буферизация ввода/вывода при дисковых операциях.
- Прямой и последовательный доступ к файлам.
- Возможность обработки файлов практически любой длины.

- Наличие стандартных внешних устройств (CON, PRN, NUL и др.).
- Обработка пакетных файлов пользователя (.bat-файлов, в том числе autoexec.bat).
- Система функций DOS, доступных программисту.
- Наличие переменных окружения DOS.
- Обработка дискет различных форматов (160 кб, 180 кб, 320 кб, 360 кб, 400 кб, 640 кб, 720 кб, 800 кб).
- Возможность обработки собственного формата диска.
- Возможность работы с различными типами дисководов, кроме НГМД 6022 (количество дорожек определяется при загрузке DOS).
- Возможность подключения новых внешних устройств и новых драйверов.
- Возможность работы с несколькими файлами одновременно.
- Возможность производить одновременно чтение из файла и запись в него.
- Оригинальный алгоритм, позволяющий обрабатывать FAT и каталог любого размера.
- Возможность работы системы с монитором БК-0011(М) и БК-0010(.01), причем на БК-0011(М) программисту доступны все резервы ЭВМ*.
- Все модули системы абсолютно перемещаемы.

Структура операционной системы

DX-DOS состоит из трех частей:

- системный загрузчик;
- ядро системы (DXDOS.SYS);
- командный процессор (COMMAND.COM).

Системный загрузчик располагается на нулевой дорожке дискеты (независимо от того, системная она или нет). Запуск системы осуществляется обычным способом — передачей управления по адресу 160000**. Загрузчик определяет формат дискеты и наличие на ней системного файла DXDOS.SYS (его имя должно стоять в первой строке каталога), а затем загружает его в память. Если файл DXDOS.SYS не обнаружен или имеет место ошибка чтения, выдается сообщение: «**Non system disk or disk error**» («Дискета не системная или ошибка диска»).

После загрузки системного файла DXDOS.SYS ему передается управление. Внутри этого файла находится вторичный загрузчик, алгоритм его работы следующий.

- *Вывод заставки системы.*
- *Проверка типа машины.* Загрузчик распознает два типа машин: «БК-0010» и «БК-0011». Если ПК идентифицирован как «БК-0011», производится переключение страниц памяти: VRAM — с адреса 40000, четвертая страница RAM — с адреса 100000. Если ПК распознан как «БК-0010», проверяется наличие как минимум 8 кб расширенной памяти (диапазон адресов 120000—160000, поиск ведется от верхних адресов к нижним). Если хотя бы 8 кб памяти не обнаружено, то выдается сообщение: «**Fatal error: Cannot allocate memory for DOS**» («Невозможно отвести память под DOS») и загрузка перекращается.
- *Проверка количества дисководов и их параметров* (число дорожек на каждом из дисководов).
- *Пересылка тела DOS в верхние адреса памяти и передача на него управления* (часть вторичного загрузчика также перемещается в памяти).
- *Разворачивание буферов DOS, системных таблиц и рабочей области DOS.* Если обнаружен один дисковод, система генерирует второй псевдодиск, т. е. при обращении к диску B: DOS адресуется к дисководу A: и требует сменить дискету. В случае нехватки памяти выводится сообщение (только на БК-0010): «**Configuration too large for memory**» («Данная конфигурация не помещается в памяти»). В этом случае можно порекомендовать следующие действия:

— отключение одного из дисководов (если их более двух);

— увеличение памяти.

- *Запись в область окружения DOS системной переменной:*
- COMSPEC=<метка диска>:COMMAND.COM
- *Поиск файла AUTOEXEC.BAT.* При обнаружении он открывается, а его FCB заносится в область пакетных файлов DOS.
- *Перехват прерываний EMT 36 и EMT 14.*
- *Установка вектора 20 на диспетчер функций DOS.*
- *Вызов функции 0,* которая загружает значение переменной COMSPEC (как правило, это COMMAND.COM).
- *Передача управления оболочке системы (COMMAND.COM).* Работа этого блока ОС начинается с вывода сообщения о версии командного процессора. Далее осуществляется просмотр области FCB-блоков. Если обнаружен FCB файла AUTOEXEC.BAT, начинается его выполнение. В противном случае выводится приглашение системы и ожидается ввод команд пользователя.

* Напоминаем читателям, что прочие ДОС для БК (ANDOS, MKDOC и т. д. за исключением разве что NORD) фактически превращают БК-0011(М) в БК-0010, а богатые ресурсы ОЗУ одиннадцатой модели используются в лучшем случае под электронный диск. — *Прим. ред.*)

** Система DX-DOS нормально работает (проверено в редакции) на КНГМД 326 (как на БК-0011М, так и на БК-0010(.01) с соответствующим образом доработанным КНГМД 326) и на КНГМД фирмы «АльтПро» для БК-0010(.01) (модели А16 и К16). Работа с контроллерами винчестера (модели «АльтПро» и «Самара») не проверялась. — *Прим. ред.*

Список команд пользователя

DIR <шаблон>

Вывод каталога на стандартное устойство вывода (по умолчанию — на экран). Примеры:

DIR — вывод всего калалога;
DIR /W — вывод каталога в сокращенном («сжатом») виде;
DIR /P — постраничный вывод каталога;
DIR /H — вывод каталога с показом скрытых файлов;
DIR /F — сообщение объема свободного пространства на диске.

Возможна комбинация ключей в любом порядке, кроме ключа F. При задании шаблона допускается использование символов-масок «?» (один произвольный символ или пустое знакоместо) и «*» (любое количество произвольных символов или пустое знакоместо). Примеры:

DIR /P *.TXT — постраничный вывод имен всех файлов типа TXT :

DIR P?AB.TXT — вывод списка TXT-файлов, в имени которых на второй позиции стоит любая буква или цифра, например P1AB.TXT, PRAB.TXT, PAB.TXT (второе знакоместо пустое) и т. д.;

DIR P*.BAT — вывод списка файлов типа BAT, имена которых начинаются с символа «P»;

DIR B:*.TXT — вывод списка файлов типа TXT с диска B: (если метка диска не задана, подразумевается текущий диск).

При вызове команды DIR можно использовать функцию перенаправления ввода-вывода (см. примечание 4):

DIR *.TXT > PRN — печать списка всех файлов типа TXT на принтере;

DIR > DIR.TXT — запись всего каталога текущего диска в файл с именем DIR.TXT.

Возможен вывод по переменной окружения DIRCMD (см. сведения о переменных окружения в примечании редактора в конце статьи).

DEL <шаблон> или ERA <шаблон>

Удаление файлов по шаблону. Примеры:

DEL DIR.TXT — удаление файла DIR.TXT;

DEL *.TXT — удаление всех файлов типа TXT;

DEL *.* — удаление всех файлов (в этом случае DOS выводит перезапрос: «Вы уверены?» и при желании позволяет отказаться от выполнения операции);

DEL *.* < F.TXT — удаление всех файлов без перезапроса (файл F.TXT должен содержать по крайней мере один символ «Y» или «N»).

TYPE <имя файла>

Вывод текстового файла или рисунков на экран (распознаваемые форматы текста: EDASP, MSX, MICRO). Примеры:

TYPE DIR.TXT — вывод файла DIR.TXT;

TYPE DIR.TXT > PRN — вывод файла DIR.TXT на принтер;

TYPE RIS.SCR /S — вывод рисунка на экран.

REN <шаблон1> <шаблон2>

Переименование файлов <шаблон1> по <шаблон2>. Примеры:

REN DIR.TXT DIR.KAT — переименование файла DIR.TXT в DIR.KAT;

REN *.TXT *.DOC — изменение для всех файлов расширений TXT на DOC.

VER

Вывод номера версии DOS. Примеры:

VER — вывод номера версии на экран;

VER > PRN — печать номера версии на принтере;

VER > V.TXT — вывод версии в файл V.TXT.

CLS

Очистка экрана дисплея.

PAUSE [<сообщение>]

Приостанавливает работу DOS*. Примеры:

PAUSE — приостановка работы, сопровождаемая выводом текста: «**Strike any key to continue...**» («Нажмите любую клавишу для продолжения»);

PAUSE НАЖМИТЕ ЛЮБУЮ КЛАВИШУ — вывод текста («НАЖМИТЕ ЛЮБУЮ КЛАВИШУ»), записанного вслед за командой PAUSE с последующей приостановкой работы системы (строка «Strike any key to continue...» в этом случае не выводится).

ECHO <сообщение>

Вывод текстового сообщения на экран. Примеры:

ECHO ---------- — вывод на экран горизонтальной линии;

ECHO OFF — запрещение вывода текста команд на экран при выполнении пакетного файла;

ECHO ON — отмена предыдущей команды ECHO OFF;

ECHO — вывод текущего состояния (ON/OFF), заданного командой ECHO ON или ECHO OFF;

ECHO R > PRN — вывод буквы R на принтер.

REM <коментарии>

Невыполняемая («пустая») команда. Предназначена для записи комментариев в пакетном файле. Пример:

REM производится копирование файлов на диск B:

SET <имя> = <значение>

Установка, просмотр или удаление переменных окружения DOS**. Примеры:

SET — вывод значений переменых на экран;

SET > PRN — то же на принтер;

SET > F.TXT — то же в файл F.TXT;

SET DIRCMD=*.TXT — установка переменной DIRCMD (для команды DIR);

SET DIRCMD= — удаление переменной DIRCMD.

COPY <шаблон1> <шаблон2>

Копирование файлов <шаблон1> в <шаблон2>. Примеры:

A>COPY A.DOC B:F.TXT — копирование файла A.DOC с диска A: на диск B: под именем F.TXT;

---

* Здесь и далее в квадратных скобках в обозначении команды записаны необязательные параметры. — *Прим. ред.*

** Перечень допустимых имен переменных окружения и информация об их назначении автором разработки, к сожалению, не представлены. Некоторые сведения об этих переменных, полученные при испытании DX-DOS в редакции, даны в примечании редактора в конце статьи. — *Прим. ред.*

A>COPY *.TXT B: — копирование всех файлов типа TXT с диска A: на диск B: под теми же именами (если такие файлы на диске B: уже имеются, они будут затерты вновь скопированными);

A>COPY B:*.DOC — копирование всех файлов типа DOC с диска B: на текущий диск A: (под теми же именами, с заменой уже существующих новыми);

COPY CON START.BAT — копирование (ввод текста) с консоли (клавиатуры) в файл START.BAT, окончание ввода — «СУ»+«К» и затем «ВВОД»*;

COPY CON PRN — копирование текста с клавиатуры на принтер (режим «пишущей машинки»);

COPY *.* NUL — копирование всех файлов в «нулевое устройство» (эта команда никаких изменений на диске не производит и служит для проверки читаемости файлов);

COPY A:*.* B:*.* > A.TXT — копирование всего содержимого диска A: на диск B: (причем все сообщения, обычно выдаваемые системой на экран, перенаправляются в файл A.TXT);

COPY A:*.* B:*.* > NUL — то же, но никаких сообщений ни на экран (принтер), ни в файл не выводится (они передаются в устройство NUL).

COPY <шаблон1> + <шаблон2> + ... + <шаблон N> <выходной шаблон>

«Склеивание» нескольких файлов в один (применять символы групповых операций «?» и «*» здесь не рекомендуется, так как это может привести к непредсказуемым результатам операции). Примеры:

COPY D1.TXT + D2.TXT + D3.TXT ALL.TXT — объединение файлов D1.TXT, D2.TXT и D3.TXT в файл ALL.TXT;

COPY *.TXT + *.DOC *.ALL — «склеивание» всех файлов типа TXT и DOC в файлы типа ALL (попарное слияние файлов .TXT и .DOC по порядку их расположения на диске — создается несколько файлов типа ALL, причем имя каждого выходного файла формируется по имени первого файла пары (.TXT), при исчерпании одного из типов файлов (.TXT или .DOC) операция завершается);

COPY *.TXT + *.DOC ALL.TXT — объединение всех файлов .TXT и .DOC в один выходной файл;

COPY *.TXT ALL.TXT — объединение всех файлов типа TXT в один файл ALL.TXT;

COPY CON + D.TXT ALL.TXT — добавление в начало файла D.TXT текста, введенного с клавиатуры (результат — в файле ALL.TXT).

COPY <шаблон1> + <шаблон2> + ... + <шаблон N>

«Приклеивание» к файлу <шаблон1> остальных указанных в команде файлов (отличие от предыдущей команды — отсутствие записанного через пробел имени результирующего файла). Примеры:

COPY D.TXT + D.DOC — к файлу D.TXT добавляется файл D.DOC;

* Команда COPY CON <имя файла> является простым и удобным средством создания нового текстового файла без использования текстового редактора. Однако возможности по редактированию текста ограничены в данном случае использованием клавиши «ЗАБОЙ» (а также клавиш, указанных в разделе «Редактор командной строки DOS») в текущей строке до нажатия на «ВВОД» для перехода к следующей строке текста. — *Прим. ред.*

COPY *.TXT + *.DOC — ко всем файлам типа TXT добавляются файлы типа DOC (попарная «склейка»);

COPY D.DOC + CON — добавление в конце файла D.DOC текста, введенного с клавиатуры.

DATE

Ввод или просмотр текущей даты. Примеры:

DATE — вывод на экран текущей даты и запрос новой (если изменение даты не требуется, следует нажать клавишу «ВВОД»);

DATE > PRN — вывод текущей даты на принтер (текст запроса новой даты печатается, но ввод значения не запрашивается);

DATE < F.RRR — ввод даты из файла F.RRR;

DATE > PRN < F.RRR — две предыдущие операции объединены в одну.

FORMAT <диск:>/<параметры>

Форматирование дисков (внешняя команда*). Параметры:

/1 — форматирование одной стороны дискеты;

/4 — на 80-дорожечном дисководе дискета форматируется на 40 дорожек;

/8 — при форматировании на каждой дорожке размещается 8 секторов;

/9 — при форматировании на каждой дорожке размещается 9 секторов;

/B — после форматирования на диске резервируется место для последующей записи системы;

/S — после форматирования на диск переносится операционная система, диск становится системным;

/? — вывод справочной информации по данной команде;

/D — при форматировании используется нестандартный (встроенный в утилиту FORMAT, а не прошитый в ПЗУ) драйвер.

Возможно совместное использование параметров (кроме /B и /S); параметры /8 и /9 можно использовать только при наличии в контроллере НГМД ПЗУ 326-й прошивки или совместимого с ним. Примеры:

FORMAT A: — форматирование дискеты на максимальный объем (на 80-дорожечном дисководе 800 кб, т. е. 80 дорожек при 10 секторах на дорожке; на 40-дорожечном дисководе 400 кб, т. е. 40 дорожек при 10 секторах на дорожке**);

FORMAT A:/4/9 — форматирование дискеты на 360 кб;

FORMAT A:/4/8/S — форматирование дискеты на 320 кб с переносом на него системы;

FORMAT B:/S/1/4/8 — форматирование диска на 160 кб, дискета становится системной.

Примечание. При задании в команде ключа /S на вновь отформатированную дискету копируются файлы DXDOS.SYS (с атрибутом «скрытый») и COMMAND.COM (при наличии единственного дисковода A: утилита дважды выводит запрос «Insert system disk...» («Вставьте системный диск») и затем — «Insert taget disk...» («Вставьте целевой диск»), после чего ждет нажатия на

* Внешние команды ОС обрабатываются не в COMMAND.COM, а отдельными утилитами из комплекта DOS, имеющимися на дискете. Фактически любая внешняя команда есть запуск отдельной программы. — *Прим. ред.*

** Формат 400 кб (40 дорожек по 10 секторов) для БК является нестандартным, работа с такой дискетой на 80-дорожечном дисководе невозможна (этот недостаток будет устранен в следующей версии DOS).

любую клавишу). Остальные файлы (AUTOEXEC.BAT и утилиты) на новый системный диск копируются командой COPY с шаблоном *.* . Диски емкостью 800 кб можно также копировать с помощью ANDOS XEROX, но использование для этой цели имеющихся на IBM копировщиков дисков DiskCopy и DiskDuplicator не допускается — дискета должна форматироваться на БК.

MODE <имя устройства> <параметры>

Настройка внешнего устройства и запись в файл произвольных кодов (внешняя команда). Возможные параметры: символ «[» означает код ESC, символ «#» указывает, что далее следует код в десятичном формате. Примеры:

MODE PRN [@ — инициализация принтера (ESC @);

MODE CON #155 ПРИВЕТ — передача кода 155 (32/64 символа в строке) и сообщения «ПРИВЕТ» на консоль (т. е. на экран дисплея);

MODE F.TXT [@ [! #128 — передача кодов: ESC @, ESC !, 128 в файл F.TXT.

ADDRESS <имя файла>

Просмотр или изменение адреса загрузки файла в ОЗУ (внешняя команда). Примеры:

ADDRESS prog.com — просмотр адреса загрузки файла prog.com;

ADDRESS prog.com 1000 — установление адреса загрузки файла prog.com равным 1000 (восьмеричное значение).

CBOOT <имя диска> /<параметры>

Коррекция загрузочного сектора дискеты для возможности чтения файлов на IBM-совместимом компьютере (внешняя команда). Возможные параметры:

/IBM — коррекция загрузочного сектора в формат IBM (при этом диск становится несистемным);

/BK — обратное преобразование загрузочного сектора в формате БК;

/? — вывод краткой справочной информации.

Примеры:

CBOOT B: /IBM — коррекция загрузочного сектора диска B: в формат IBM;

CBOOT /BK — коррекция загрузочного сектора текущего диска в формат БК;

CBOOT /? — вывод справочной информации.

Примечание. Преобразование загрузочного сектора в формат IBM необходимо только для дискет емкостью более 360 кб, так как диски 160, 180, 320 и 360 кб читаются на IBM без коррекции загрузчика.

BK_IBM <имя1> <имя2>

Перекодирование текстового файла <имя1> из формата БК (EDASP MICRO) в файл <имя2> в формате IBM (внешняя команда).

IBM_BK <имя1> <имя2>

Перекодирование текстового файла <имя1> из формата IBM в файл <имя2> в формате БК (внешняя команда). Примеры:

BK_IBM DATA.TXT PRN — перекодирование файла data.txt в формат IBM и вывод перекодированного текста на принтер. (Это действие может потребоваться, если к вашему БК подключен принтер с IBM-овской знаковой таблицей «альтернативной» кодировки.)

COPYM

Программа для копирования файлов с диска на магнитную ленту и обратно. Команды этой утилиты можно просмотреть, набрав после ее запуска команду HELP.

Перенаправление ввода-вывода

Перенаправление ввода-вывода осуществляется с помощью знаков:

> — перенаправление вывода (если файл, в который предполагается направлять информацию, уже существовал, его прежнее содержимое уничтожается);

>> — то же, но если файл уже существовал, данные добавляются к его концу; если же такого файла еще не было, создается новый;

< — перенаправление ввода.

Перенаправление ввода-вывода возможно только в программах, которые обращаются к функциям DOS.

Редактор командной строки DOS

Для редактирования командной строки можно использовать следующие клавиши:

«СТРЕЛКА ВНИЗ» — вывод содержимого буфера командной строки на экран (если буфер пуст, команда игнорируется);

«СТРЕЛКА ВВЕРХ» — стирание с экрана раннее введенных символов (информация в буфере сохраняется);

«СТРЕЛКА ВПРАВО» — вывод одного символа из буфера на экран;

«СТРЕЛКА ВЛЕВО» — стирание одного символа (слева от курсора) с экрана;

«ВВОД» — ввод в буфер информации, находящейся на экране (одновременно с запуском команды на исполнение);

«СУ»+«F» (код 6) — включение/выключение одновременного вывода информации на экран и принтер.

При выполнении любой внутренней или внешней команды информация в буфере командной строки сохраняется, строчные латинские символы преобразуются в заглавные.

Примечание. Буфер командной строки можно представлять себе как некую «невидимую» строку, находящуюся под командой. Если нажать клавишу «↓», когда курсор находится в пустой командной строке, в ней появится полный текст сохранившейся в буфере предыдущей команды. Если же в командной строке, например, уже введено два символа, а затем нажимается клавиша «↓», после уже имеющихся символов выводится уже сохраненная в буфере часть прежней команды начиная с третьего символа и до конца (если, конечно, длина прежней команды была больше двух знаков). Аналогично, при нажатии на «→» на месте курсора (он смещается вправо) появится символ из соответствующей знаковой позиции буфера (если он есть). Например, если курсор находился во второй позиции командной строки, после нажатия на «→» на его месте окажется второй символ из буфера.

Пакетные файлы

Как уже говорилось ранее, DX-DOS может обрабатывать пакетные файлы пользователя (.BAT-файлы). Пакетный файл обычно создается с помощью команды **COPY CON <имя файла>** (см. команду COPY) или в текстовом редакторе и должен иметь расширение «BAT».

Вызов пакетного файла на исполнение производится командой:

<ИМЯ ФАЙЛА> <параметр1> <параметр2> . . .

(параметры вводятся, если в пакетном файле предусмотрено использование формальных параметров — см. ниже).

Из командного файла можно вызвать другой с последующим возвратом (максимальное значение вложенности равно 3), а также любую программу пользователя с последующим возвратом в командный файл. Для этого достаточно записать в отдельной строке .BAT-файла имя программы или другого .BAT-файла так, как это делалось бы при их запуске из командной строки.

Следующая дополнительная команда DOS используется только в пакетных файлах:

@ — подавление вывода текста DOS-команды на экран.

Данная команда подавляет вывод на экран той строки, перед которой стоит знак «@». Пример :

@ECHO OFF — отключение вывода текста последующих DOS-команд на экран, причем сама эта команда также не выводится на экран.

Пакетный файл может содержать формальные параметры (%<число>). При его исполнении формальные параметры заменяются на фактические (заданные в командной строке). Максимальное количество параметров равно 10, причем параметру %0 соответствует имя самого пакетного файла. Формальные параметры доступны и в программах пользователя. Примеры (если пакетный файл содержит перечисленные ниже команды и вызван как <имя.BAT> <имя1> <имя2>):

TYPE %0 — просмотр текста самого пакетного файла;

TYPE %1 — просмотр текста первого файла;

COPY %2 %1 — копирование второго файла в первый.

Информация для системных программистов

Файловая система DOS и понятие FCB

Файловая система DOS использует понятие УПРАВЛЯЮЩЕГО БЛОКА ФАЙЛА (FILE CONTROL BLOCK или, сокращенно, FCB). Занося в поля FCB (они перечислены ниже) требуемую информацию, можно считывать или записывать данные произвольной длины в любое место файла (прямой доступ).

Ниже приводится формат полей FCB, программисту достаточно внести в них номер дисковода, имя файла и вызвать функцию создания, открытия или поиска файла (остальные поля заполняет система). Содержимое полей, не описанных в таблице, но используемых DOS, не следует изменять, так как это может привести к повреждению файла и даже всей файловой системы (появление «потерянных кластеров»). Все числа даны в восмеричной системе счисления.

Смещение от начала FCB	Длина в байтах	Назначение
0	1	Номер дисковода: 0 — текущий, 1 — A:, 2 — B: и т. д.
1	13	Имя файла и его расширение
14	2	Номер текущего блока (для функций последовательного доступа), длина блока равна 128 байт
16	2	Длина записи в байтах
20	4	Длина файла в байтах
24	2	Дата создания файла
26	2	Адрес загрузки файла в память (для исполняемых файлов)
30	1	Идентификатор устройства
31	1	Положение имени файла в каталоге
32	2	Начальный кластер
34	4	Используется DOS
40	1	Текущая запись (последовательный доступ к файлам)
42	4	Номер записи (для функций прямого доступа)
46	1	Используется DOS
47	1	Атрибуты файла
50	2	Адрес обмена

В модифицированном FCB (для функции переименования) со смещением 15 (т. е. начиная с 15-го байта от начала таблицы) заносится новое имя.

Поясним назначение отдельных полей более подробно.

Длина записи — количество байтов, которое будет записано на диск (или считано с диска) за одну операцию ввода/вывода.

Идентификатор устройства — имя устройства, которому принадлежит данный FCB:

- дисковод A: — код буквы «A», B: — код «B» и т. д.;
- консоль: 377;
- AUX: 376;
- NUL: 375;
- LST (принтер): 374;
- PRN (принтер): 373.

Положение имени файла в каталоге — порядковый номер имени файла от начала каталога.

Адрес обмена — адрес, по которому будут считываться данные с диска (или записываться на диск); это поле функции DOS не изменяют.

Дисковая область

Для каждого устройства (кроме символьных — AUX, NUL и т. д.) в памяти отводится 1002 байта для размещения FAT и 56 байт для параметров устройства и промежуточных данных (см. таблицу). Все числа даны в восмеричной системе.

Смещение	Назначение
0	Объем сектора в байтах
2	Объем кластера в секторах
4	Количество копий FAT
6	Максимальное количество элементов в каталоге
10	Общее число секторов
12	Идентификатор формата
14	Длина FAT в секторах
16	Число секторов на дорожке
20	Количество сторон диска
22	Указатель типа носителя (0 — носитель съемный, иначе несъемный, зарезервировано для работы с винчестером)
24	Объем кластера в байтах
26	Счетчик количества открытых файлов для данного устройства
30	Последний сектор каталога
32	Начальный сектор каталога
34	Последний свободный кластер на носителе
36	Биты настройки контроллера
40	Параметры дисковода
42	Номер устройства
44	Используется DOS
46	Адрес FAT в памяти
50	Используется DOS
52	Используется DOS
54	Используется DOS

Функции DOS

Доступ к устройствам DOS возможен двумя способами: посредством EMT 36 или через функции DOS по команде IOT (вектор прерывания 20). Параметры EMT 36 не отличаются от передаваемых магнитофону, поэтому все созданные раннее программы будут нормально работать с диском. Но возможности EMT 36 весьма скромны, поэтому желательно, чтобы программы обращались к ресурсам DOS через функции.

Ниже приводится описание всех функций DOS, предоставляемых программисту. Все номера восьмеричные, в скобках указываются их десятичные эквиваленты. (Функции 3, 4, 7, 10 ,12 зарезервированы.)

Функция 0 (0)

Завершение программы пользователя. Управление передается DOS, при этом вызывается оболочка, имя которой записано в переменной COMSPEC.

Функция 1 (1)

Ввод символа со стандартного устройства ввода (по умолчанию консоль). Выходные данные: в R0 — код полученного символа.

Функция 2 (2)

Вывод символа на стандартное устройство вывода (по умолчанию консоль). Входные данные: R0 — код символа.

Функция 5 (5)

Вывод символа на принтер. Входные данные: R0 — код символа.

Функция 6 (6)

Буферизованный ввод с клавиатуры с возможностью редактирования. Входные данные: R1 — адрес буфера, R2 — длина буфера в байтах.

Функция 11 (9)

Вывод строки символов на устройство вывода (по умолчанию консоль). Входные данные: R1 — адрес строки символов (она должна заканчиваться нулевым байтом).

Функция 14 (12)

Очистка буфера командной строки.

Функция 15 (13)

Инициализация драйвера (контроллера) дисковода.

Функция 16 (14)

Задание текущего дисковода. Входные данные: R0 — номер дисковода (0 — дисковод A:, 1 — B: и т. д.).

Функция 17 (15)

Открытие файла методом FCB. Входные данные: R4 — адрес FCB.

Функция 20 (16)

Закрытие файла. Входные данные: R4 — адрес FCB.

Функция 21 (17)

Поиск первого файла по заданному образцу. Входные данные: R4 — адрес FCB

Функция 22 (18)

Продолжение поиска файлов (после вызова функции 21; FCB изменять не следует). Входные данные: R4 — адрес FCB

Функция 23 (19)

Удаление файла. Входные данные: R4 — адрес FCB. (Удаляемый файл должен быть закрытым, при задании имени допускается использовать символ групповой операции «?».)

Функция 24 (20)

Последовательное чтение из файла. Входные данные: R4 — адрес FCB. (После чтения соответствующие поля FCB модифицируются.)

Функция 25 (21)

Последовательная запись в файл. Входные данные: R4 — адрес FCB. (После записи соответствующие поля FCB модифицируются.)

Функция 26 (22)

Создание файла (если такой файл уже существовал, то он «инициализируется» — его размер устанавливается равным 0). Входные данные: R4 — адрес FCB.

Функция 27 (23)

Переименование файла. Входные данные: R4 — адрес модифицированного FCB. (Разрешается использовать символ групповой операции «?».)

Функция 30 (24)

Доступ к буферам DOS. Выходные данные: R0 — адрес области .BAT—файлов, первое слово которой — уровень вложенности (0 — область пуста); R1 — адрес FCB стандартного устройства ввода; R2 — адрес FCB стандартного устройства вывода.

Функция 31 (25)

Получение номера текущего устройства прямого доступа (дисковода). Выходные данные: R0 — номер дисковода.

Функция 32 (26)

Установка адреса области обмена с диском (адреса чтения/записи физического сектора). Входные данные: R0 — адрес области обмена.

Функция 33 (27)

Получение параметров текущего устройства (кроме символьных — AUX, NUL и т. д.). Выходные данные: R2 — адрес списка параметров (изменять эти данные запрещается).

Функция 34 (28)

Аналогична функции 33 (27), но для заданного устройства. Входные данные: R0 — номер устройства (1 — дисковод A: и т. д.). Выходные данные: R2 — адрес списка параметров.

Функция 37 (31)

Перехват EMT. Входные данные: R0 — номер EMT (от 0 до 130); R1 — абсолютный адрес программы обработки данного EMT (если R1=0, то восстанавливается прежний адрес).

Функция 40 (32)

Получение адреса области окружения DOS. Выходные данные: R1 — адрес начала области; R2 — длина области в байтах.

Функция 41 (33)

Прямой доступ, чтение. Входные данные: R4 — адрес FCB. Выходные данные: R0 — количество считанных байтов. Поля FCB не изменяются.

Функция 42 (34)

Прямой доступ, запись. Входные данные: R4 — адрес FCB. Выходные данные: R0 — количество записанных байтов. Поля FCB не изменяются.

Функция 43 (35)

Получение длины файла в байтах. Входные данные: R4 — адрес FCB. Выходные данные: R0 — младшее слово длины, R1 — старшее слово длины.

Функция 44 (36)

Задание позиции прямого доступа (т. е. переход от последовательного доступа к прямому). Номер записи вычисляется по номерам блока и записи, заданных при последовательном доступе. Входные данные: R4 — адрес FCB.

Функция 45 (37)

Задание новой подпрограммы обработки функций DOS. Входные данные: R0 — номер функции; R1 — абсолютный адрес подпрограммы.

Функция 46 (38)

Исполнение программы (автозапуск работает корректно). Входные данные: R1 — адрес строки, содержащей имя файла; R2 — флаг (0 — запуска не происходит, иначе файл запускается на исполнение).

Функция 47 (39)

Считывание нескольких записей. Входные данные: R4 — адрес FCB; R0 — количество записей. Выходные данные: R0 — количество действительно считаных записей. Поле «номер записи» FCB модифицируется.

Функция 50 (40)

Запись нескольких записей. Входные данные: R4 — адрес FCB, R0 — количество записей. Выходные данные: R0 — количество действительно записанных записей. Поле «номер записи» FCB модифицируется.

Функция 51 (41)

Синтаксический разбор строки. Входные данные: R1 — адрес обрабатываемой строки; R4 — адрес формируемого FCB—блока; R2 — тип разбора

(если 0, то обычный разбор, иначе если первым встретится код 0, FCB примет вид: _???????????). Выходные данные: R2 — количество символов «?».

Функция 52 (42)

Получение текущей даты. Входные данные: R0.

Функция 53 (43)

Установка новой даты. Входные данные: R0.

Функция 54 (44)

Абсолютное чтение сектора. Входные данные: R0 — номер дисковода (1 — дисковод A: и т. д.); R1 — номер сектора; R2 — адрес загрузки в ОЗУ. Для нормальной работы область параметров данного устройства должна быть заполненной.

Функция 55 (45)

Абсолютная запись сектора. Параметры аналогичны используемым в функции 54 (44).

Функция 57 (47)

Получение адреса обмена с диском (адреса чтения/записи сектора). Выходные данные: R0 — адрес обмена с диском.

Функция 60 (48)

Получение версии DOS. Выходные данные: R0 — версия, где ст. байт — номер версии, мл. байт — номер подверсии.

Функция 61 (49)

Доступ к внутрисистемной информации. Выходные данные: R1 — адрес области DOS.

Функция 62 (50)

Получение адреса дисковой области. Входные данные: R0 — номер устройства. Выходные данные: R1 — адрес области.

Функция 65 (53)

Получение абсолютного адреса подпрограммы обработки функции DOS. Входные данные: R0 — номер функции. Выходные данные: R0 — абсолютный адрес начала подпрограммы в памяти.

Функция 66 (54)

Получение объема свободного пространства на диске. Входные данные: R0 — номер дисковода (1 — дисковод A: и т. д.). Выходные данные: R1 —

количество секторов в кластере; R2 — общее количество кластеров на диске; R0 — количество свободных кластеров на диске.

Функция 67 (55)

Получение переменной из области окружения DOS. Входные данные: R1 — адрес имени переменной. Выходные данные: R2 — указывает на значение переменной в окружении DOS; R0 — указывает на имя переменной в области окружения DOS. Если такой переменной нет, то C=1 и R2 указывает на свободную строку в области окружения DOS.

Функция 70 (56)

Запись переменной в область окружения DOS. Входные данные: R1 — адрес строки с именем и значением переменной.

Функция 56

Обработка критических ошибок. Выходные данные: R0 — символ «I» — игнорировать ошибку, «R» — повторить, «A» — отменить.

Все функции DOS вызываются с помощью команды IOT, а следующее после нее машинное слово — номер функции. Если при выполнении функции возникла ошибка, то бит C=1, а в ячейке 52 хранится код ошибки.

Пример листинга для организации эхо-печати строки:

```
MOV #1000,R1        ;адрес буфера ввода строки символов
MOV #200,R2         ;длина буфера
IOT                 ;вызов  функции 6 (ввод строки символов)
.#6
IOT                 ;вызов функции 11 (вывод строки
                    ; символов на стандартное
.#11                ; устройство вывода)
IOT                 ;возврат в DOS
.#0
```

Пример листинга для организации прямого доступа:

```
MOV #ADR,R1         ;адрес строки, содержащей имя файла
MOV #FCB,R4         ;адрес формируемого FCB-блока
IOT
.#51                ;синтаксический разбор имени файла,
                    ; заполнение полей FCB
IOT
.#17                ;открытие  файла
MOV #6,16(R4)       ;длина записи = 6 байт
MOV #ADR1,50(R4)    ;задание адреса обмена
MOV #20,42(R4)      ;считывание с 20-й записи
IOT                 ; чтение информации с диска
.#41
IOT                 ;закрытие файла
.#20
```

С помощью функций DOS возможен доступ к внутрисистемной информации (указаны смещения системных таблиц и специальных ячеек):

6 — адрес списка дисковых блоков (адреса абсолютные);
15 — количество устройств прямого доступа в системе (1 байт);
16 — количество дисководов в системе (1 байт);
20 — адрес рабочей области драйвера дисковода;
22 — значение стробирующего бита для принтера (400 — для БК-0010(.01), 40000 — для БК-0011(М), аналогично БЕЙСИКу);
60 — адрес начала DOS;
64 — адрес таблицы подпрограммы чтения сектора с устройства (при старте устанавливается равным 160006 для всех устройств);
66 — адрес таблицы подпрограммы чтения секторов на логическом уровне (на уровне DOS);
104 — тип машины (0 — «БК-0010», иначе — «БК-0011», 1 байт);
126 — адрес списка имен символьных устройств (имена разделяются нулевым байтом);
130 — адрес таблицы переходов для символьных устройств (абсолютные адреса);
164 — верхний адрес пользовательского ОЗУ (40000);
166 — адрес параллельного порта (при старте устанавливается равным 177714).

Остальные ячейки изменять нельзя, так как это может привести к повреждению файловой системы или зависанию компьютера.

Ошибки при работе DOS

Ошибки, выдаваемые при работе DOS, делятся на два уровня: ошибки драйверов (коды 1 — 17, низкий уровень) и ошибки ядра системы (коды больше 20). При возникновении ошибки DOS возвращает код в ячейке 52:

20 — файл не открыт;
21 — неверное имя устройства;
22 — нет свободных кластеров;
23 — неверно сменен диск;
24 — нет места в каталоге;
25 — файл не найден;
26 — выход за пределы файла, данных нет;
27 — то же, но часть данных загружена;
30 — неверное имя файла в поле FCB;
31 — деление на нуль;
32 — файл уже открыт;
33 — ошибка переименования;
34 — ошибка в FAT;
35 — не найден фактический параметр в командной строке;
36 — выход за пределы 64 кб;
37 — файл не закрыт;
40 — нет памяти в области окружения DOS;
41 — неверный синтаксис в командной строке;
42 — переменная не найдена в области окружения DOS.

Заключение

Созданная операционная система, являясь системой типа MS-DOS, допускает последующее наращивание. В версии 1.00 реализована только небольшая часть многочисленных возможностей системы MS-DOS. Основное ограничение DX-DOS — отсутствие поддержки древовидных файловых структур (подкаталогов). Преодоление этого ограничения связано прежде всего с наращиванием аппаратных ресурсов, прежде всего оперативной памяти (речь идет о модели БК-0010(.01)). Для компьютера БК-0010(.01) обычное количество наращиваемой памяти составляет 16 кб. Для БК-0011(М) дополнительное наращивание памяти не требуется.

Следует отметить, что создание операционной системы DX-DOS является, по существу, первым шагом к реализации достаточно масштабной и сложной системы, ресурсы которой включали бы ряд важных функций, таких как прямой доступ к файлам или возможность подключения драйверов. Реализация этих дополнительных возможностей позволит создавать прикладные программные продукты, в которых на полную мощность будет вестись работа с файлами.

Примечание редактора

Описываемая в данной статье операционная система DX-DOS выгодно отличается от практически всех ранее существовавших на БК ОС (ANDOS, MKDOS, NORD и т. д.) своей открытостью для программиста и гибкостью — качествами, присущими ее протипу MS-DOS. Более того, прежние системы правильнее было бы называть «дисковой адаптацией» прошитого в ПЗУ «магнитофонного» редактора, тогда как DX-DOS, содержащая в себе набор доступных для любого программиста системных функций (для вызова которых используется прерывание по IOT, так что эти функции не «перекрывают» мониторные EMT-команды и не препятствуют работе с TRAP) и снабженная COMMAND.COM, является «настоящей» операционной системой, написанной по всем правилам, общепринятым при разработке такого рода программ. Уже сейчас, несмотря на недоработки первой пробной версии, DX-DOS является серьезным конкурентом даже для наиболее популярной ОС ANDOS, не говоря уже о прочих (спасти ANDOS от забвения хотя бы еще на пару лет могло бы, например, появление открытой для любого программиста универсальной библиотеки подпрограммы буферизации файлового доступа, но это отдельная тема для разговора).

Что же касается самих недостатков DX-DOS, с точки зрения пользователя наиболее крупным из них (если не считать отмеченного автором статьи отсутствия подкаталогов) является отсутствие Norton-подобной файловой оболочки, что существенно снижает удобство пользования новой ОС.

Прочие недостатки гораздо менее значительны и даже все вместе взятые не могут испортить общее впечатление от DX-DOS.

- При запуске программы в среде DX-DOS с дискеты формата ANDOS и последующем выходе из этой программы система перезагружается с диска A: (или B:, если она изначально была загружена со второго дисковода), не проверяя, какая система на этом диске записана. В результате, если в указанном дисководе все еще находится системный ANDOS-диск (с которого запускалась прикладная программа), без всякого предупреждения загрузится ANDOS вместо DX-DOSa. *Рекомендация автору ОС:* перед перезагрузкой хотя бы выдавать запрос «Вставьте системный диск» с ожиданием нажатия любой клавиши.

(Разумеется, при работе с винчестером данная проблема практически несущественна.)

- После перезапуска DX-DOS в режиме «БК-0010» (после запуска эмулятора ВК0010) на БК-0011М игнорируется AUTOEXEC. BAT, хотя его отработка может понадобиться и в режиме эмуляции десятой модели. *Рекомендация автору:* можно было бы предусмотреть специальный атрибут файлов «ВК0010 only» и автоматически переводить БК-0011(М) в режим «БК-0010» при запуске программы с таким атрибутом.
- Невозможно прервать работу AUTOEXEC. BAT (в частности, при выводе первоначальной документации пользователя с помощью набора команд ECHO и PAUSE прервать вывод текста можно, лишь нажав на «СТОП» во время подчитывания с диска очередного фрагмента с файла и дав в ответ на запрос системы о дисковой ошибке команду «abort» — клавиша «А»). *Рекомендация автору:* сделать клавишу «СТОП» (или «КТ») универсальным «терминатором» любой операции (аналогично «ESC» и IBM), в том числе при отработке AUTOEXEC-BAT, а также реализовать возможность отказа от отработки AUTOEXEC. BAT, если при загрузке удерживается нажатой определенная клавиша.
- Система DX-DOS не способна правильно работать, если на компьютере установлен единственный дисковод B: (с соответствующим образом установленной перемычкой), так как это «псевдоним» диска A:. *Рекомендация автору:* хотя указанная ситуация (единственный дисковод B:) встречается не слишком часто, предусмотреть в подобном случае эмуляцию «псевдодиска» A:.
- При нажатии на кнопку Reset проходит «вылет» ЭВМ в «магнитофонный» монитор (на БК-0010(.01)). *Рекомендация автору:* предусмтреть перезапуск (инициализацию) ОС при нажатии на Reset, как это сделано, например, в ANDOSe.
- Не работает объявленный в статье вывод рисунков по команде TYPE (независимо от наличия в командной строке ключа /S файл интерпретируется только как текст). *Рекомендация автору:* исправить данную ошибку, а также дополнить команду TYPE ключом /I — просмотр текстов IBM (альтернативная кодировка).
- При вызове FORMAT с ключом /B система почему-то требует после форматирования «предъявить» ей системную дискету-оригинал (как это делается для копирования системы по ключу/S), хотя резервировать место для системы можно и «по умолчанию». Кроме того, встроенный в FORMAT нестандартный драйвер, вызываемый по ключу /D, неработоспособен (по крайней мере, на БК-0010.01 с контроллером A16 «АльтПро»).
- И наконец, весьма значительным недостатком является неполнота поставляемой на дискете DX-DOS документации пользователя (а также наличие в ней большого количества ошибок, в данной статье они исправлены). Так, автор не сообщает никаких сведений об именах переменных окружения, используемых системой, и о их назначении (кроме COMSPEC, содержащей путь к файлу COMMAND.COM и его имя). Экспериментальным путем удалось выяснить еще две переменные окружения (на БК-0010.01) — ECHO (OFF или ON) и DIRCMD, которая содержит шаблоны фильтра «по умолчанию» для команды DIR (например, при **DIRCMD** = *.TXT по команде DIR выводится список только TXT-файлов; **DIRCMD** = *.* эквивалентно отсутствию этой переменной).

Не освещен в авторской документации и такой важный аспект как совместимость DX-DOS с другими распространенными сегодня ОС. Приведенная ниже информация была получена в ходе испытаний новой ОС в редакции.

1. *Совместимость с МК-ДОС, МикроДОС, Нортон, NORD и т. п.* В связи с коренным образом различающимися форматами записи файлов на дискеты («потоковый» в МикроДОС и кластерный — в ANDOS и DX-DOS) файловая совместимость в данном случае практически отсутствует, но в МК-ДОС возможен просмотр дискет DX-DOS 800 кб, чтение и копирование с них файлов при помощи эмулятора ANDOS.

2. *Совместимость с ANDOS.* Несмотря на родственный формат записи файлов, дискеты DX-DOS (по крайней мере проверявшиеся в редакции 180- и 360-килобайтные) читаются в ОС ANDOS только как «IBMовские», поэтому дальнейшая проверка файловой совместимости производилась в среде DX-DOS. Эта ОС, как выяснилось, свободно распознает ANDOS-диск и позволяет без каких-либо проблем копировать любые файлы с дискет ANDOS на диски DX-DOS и обратно. Единственное ограничение накладывается при этом на имена файлов: в ANDOS наличие пробела внутри имени допускается, а в DX-DOS это приводит к сбою уже во время копирования. (Разумеется, каталоги ANDOS 3.1 в DX-DOS не распознается.) Можно также запускать в среде DX-DOS программы непосредственно с ANDOS-диска (на БК-0011(М) предварительно нужно запустить эмулятор ВК0010), но в многофайловых программах подгрузка второго и последующих модулей с ANDOS-диска невозможна даже на БК-0010(.01). И наконец, при выходе из программы перезагрузка системы производится с текущей дискеты (в том числе ANDOS-ной, если она к этому моменту не вынута из дисковода) без проверки, как это уже отмечено в сноске к соответствующему абзацу статьи.

3. *Совместимость с IBM.* Благодаря тому что DX-DOS была разработана как аналог MS-DOS на БК, файловая совместимость между этими системами практически стопроцентная. Так, для работы на IBM с дисками DX-DOS емкостью 360 кб и менее не требуется никакого преобразования загрузчика и запуска на IBM утилит типа 800.COM (хотя для дискет DX-DOS 800 кб использование CBOOT и 800.COM обязательно). Однако «намозолившее глаза» всем пользователям ANDOS требование, чтобы дискета обязательно была отформатирована на БК, остается в силе и здесь: диски, отформатированные на IBM, в системе DX-DOS не читаются. Остается только порекомендовать авторам дополнить эту ОС эмулятором IBM-дисков по типу разработанного для ANDOS эмулятора MSDOS.EMU.

В заключение хотелось бы отметить, что по договоренности с автором DX-DOS единственным распространителем этой ОС в Москве и Подмосковье является издательство «Компьютика» (работоспособность копий, полученных или приобретенных в других местах, не гарантируется: самая первая из появившихся в Москве еще в сентябре 1995 г. «подверсия» DX-DOS оказалась несовместима с большинством типов КНГМД была отправлена редакцией на доработку и заменена описываемой в данной статье).

В настоящее время, согласно полученной по телефону информации, автором разработана и готовится к распространению через издательство «Компьютика» новая, улучшенная версия DX-DOS (возможно, с НОРТОН-подобной оболочкой в комплекте).

За справками можно обращаться по телефону

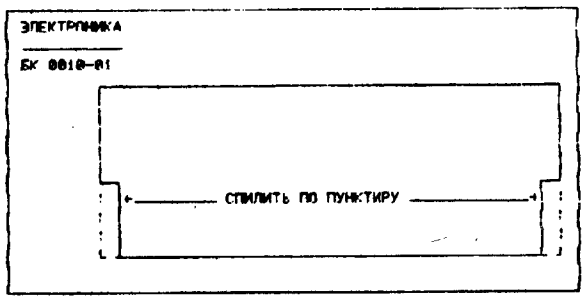
(095) 292-53-87.

СПРАВОЧНЫЙ ЛИСТОК

Подключение к БК-0010.01 новой пленочной клавиатуры МС7008.01

Размеры новой клавиатуры (ее признаком являются «ступеньки» на клавишах «→» и «←») несколько больше, чем у прежней, поэтому для ее установки в БК нужно спилить мешающие края клавиш на верхней крышке корпуса ПЭВМ.

Кроме того, в корпусе имеется прокладка из четырех соединяющихся полосок и в нее упираются пленочные шлейфы клавиатуры. Отметьте ширину мешающего участка на прокладке и спилите его.



Отрежьте шлейфы, идущие от разъемов БК к старой клавиатуре, причем по возможности около самой клавиатуры, чтобы они были как можно длинней, разрежьте их вдоль на 2 — 3 полосы шириной в несколько сантиметров и распаяйте в соответствии с таблицей:

МС7008.01 XT1	БК-0010	МС7008.01 XT2	БК-0010	МС7008.01 XT3	БК-0010
1	XT2.15				
2	XT1.1	2	XT2.9	2	XT1.15
3	XT1.2	3	XT2.10	3	XT2.11
4	XT1.3	4	XT2.14	4	XT2.1
5	XT1.5	5	XT2.8	5	XT1.14
6	XT1.6	6	XT2.4	6	XT1.13
7	XT1.11	7	XT2.5	Звуковой сигнал	
8	XT1.7	8	XT2.2		
9	XT1.9	9	XT2.3		
10	XT1.10	10	XT2.7	XT2.12	
11	XT1.4	11	XT2.6	XT2.8 (общ.)	
12	XT1.8				

Следует учесть, что пленка новой клавиатуры не выдерживает нагрева и резких перегибов.

Звуковой динамик требуется перенести со старой клавиатуры внутрь БК:

- динамик в пластмассовом корпусе можно привинтить к разъему УП внутри БК (ближе к разъему питания), воспользовавшись имеющимся там винтом;
- динамик в металлическом корпусе можно припаять к общей шине на плате БК (с помощью металлических «усиков») в любом месте (например около ПЗУ с БЕЙСИКом).

В. В. Зыков,

г. Елизово, Камчатская обл.

Несколько советов по подключению клавиатуры МС7008.01

С появлением в продаже «мягкой» пленочной клавиатуры МС7008.01 владельцы старых моделей БК-0010.01 наконец-то получили возможность избавиться от надоевшего дребезга контактов.

Проблем с ее установкой, как правило, не возникает, но у общепринятого способа монтажа новой клавиатуры, да и в заводском варианте конструкции БК-0010.01 с МС7008.01 все же есть один недостаток: очень близкое расположение края клавиатуры с выходящими из нее пленочными контактами к краю корпуса БК. Даже подпиливание последнего не дает удовлетворительного результата — все равно пленочные выводные шлейфы клавиатуры изгибаются почти под прямым углом, что приводит к их быстрому «переламыванию», если часто вскрывать корпус.

Для БК-0010.01, особенно ранних модификаций (где нет дополнительной платы в левой части корпуса) можно предложить более «щадящий» способ установки (да и в заводских моделях с мягкой клавиатурой можно провести описанную ниже переделку, что значительно увеличит срок службы пленки).

Идея доработки очень проста. Верхнюю крышку компьютера надо перевернуть на 180 градусов так, что надпись «Электроника БК-0010.01» окажется в правом нижнем углу. С клавиатуры требуется осторожно снять металлическое дно, отвернув шесть шурупов, а затем разметить и просверлить четыре новых отверстия для крепления клавиатуры к верхней крышке (старые не совпадают на каких-то 5—6 мм). Появившиеся справа 5 см пространства позволят свободно, без сильных изгибов расположить пленочные шлейфы внутри корпуса. Надпись же «Электроника БК-0010.01» при желании можно вернуть на место, аккуратно отделив ее и приклеив клеем «Момент».

HARD & SOFT

Полку периферийных устройств для БК в очередной раз прибыло! Теперь к широко используемым в разнообразных играх джойстику и мыши прибавился фотопистолет, аналогичный применяемым в игровых приставках к телевизору.

А. И. Козлов,

г. Северобайкальск

Фотопистолет для БК

Поводом для разработки фотопистолета послужило желание расширить «игровые» возможности компьютера БК. Его универсальность, позволяющая подключать к порту ввода-вывода любые внешние устройства и создавать для них программное обеспечение, позволила сделать это без каких бы то ни было доработок в схеме БК.

Принципы взаимодействия компьютера и фотопистолета не сложны: при точном наведении оптической оси фотодатчика на нарисованную на экране мишень и нажатии на «курок» ЭВМ получает один бит информации и в зависимости от его значения («1» или «0») фиксирует промах или попадание. А затем, по сюжету игры, компьютер меняет размеры или местоположение мишени, выводит на экран соответствующие сообщения, подсчитывает количество призовых очков и т. д. В данной конструкции информация о нажатии на «курок» передается компьютеру через кнопочный выключатель, подсоединенный параллельно клавише «КУРСОР ВЛЕВО», а сигнал «попадание» — через нулевой разряд порта ввода. Точность прицеливания определяет встроенный в пистолет фотоэлемент.

Остается найти подходящую конструкцию фотодатчика с усилителем. В результате проведенных экспериментов выяснилось, что готовые фотопистолеты от игровых приставок типа «Видеоспорт» и «Денди» для БК не подходят. С их помощью удавалось фиксировать наведение на солнце, лампочку, зажженную свечу. Но на экран монитора БК эти фотопистолеты не реагировали*. Но вот мне на глаза попался фотоприемник дистанционного управления ФП-2 для телевизора «Садко 51/61». Попробовал поэкспериментировать с ним — уменьшил сопротивление резистора в коллекторной цепи выходного каскада (рис. 1) и удалил диод в базовой цепи, коллектор транзистора Т5

* Возможной причиной неудачи является то, что «дендиевский» фотопистолет реагирует не на само освещение, а на переход от темноты к свету. Подробнее о работе БК совместно с фотопистолетом «Денди» см. в примечании редакции в конце статьи. — *Прим. ред.*

подключил к контакту В24 разъема «УП» (порта ввода-вывода компьютера), а общую шину ФП-2 — к контакту В19 «УП». В качестве источника питания для фотоприемника использовался выпрямитель со стабилизированным напряжением на 9 В (т. е. в два раза меньше номинального значения напряжения для ФП-2). При поднесении фотоприемника приблизительно на 5 см к изображению курсора на порт ввода-вывода стала подаваться логическая единица — «попадание» зафиксировано!

Для контроля поступающего с фотоприемника сигнала использовалась простенькая программа на ассемблере, заносящая принятые с порта байты в экранное ОЗУ (при постоянном наличии на входе логической «1» на экране должны появляться светлые вертикальные полосы). Количество замеров равно 2000_8, т. е. «контрольная зона» соответствует при нормализованном рулонном смещении экрана области служебной строки.

```
0:  MOV #40000,R1       ; адрес начала экранного ОЗУ
    MOV #2000,R2        ; количество замеров
1:  MOVB @#177714,(R1)+ ; копирование байта с порта на экран
    SOB R2,1            ; цикл по количеству замеров
    BR 0                ; "бесконечный цикл"
```

При поднесении фотоприемника вплотную к изображению курсора служебная строка действительно заполнялась вертикальными полосами, но при отодвигании от экрана количество выводимых точек уменьшалось и на

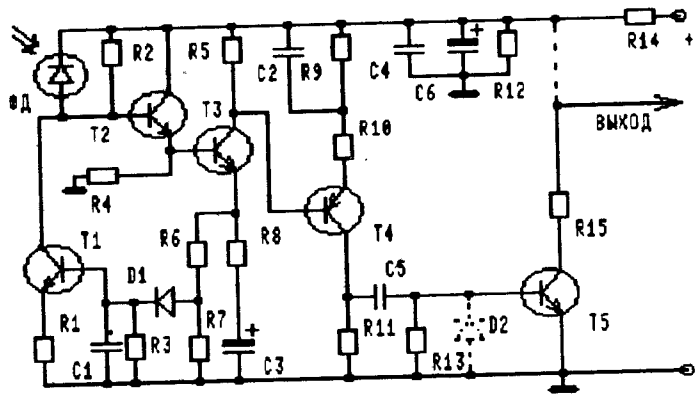


Рис. 1. Принципиальная схема фотоприемника ФП-2.

Используемые радиоэлементы:

- **фотодиод ФД-263-01;**
- **транзисторы Т1, Т2, Т3 — КТ3102БМ, Т4 — КТ361Б, Т5 — КТ315Г;**
- **диоды D1, D2 — КД521В (D2 из схемы при доработке нужно исключить);**
- **конденсаторы С1 — 0.068 мкФ, С2 — 0.47 мкФ, С3 — 47 мкФ, С4, С5 — 0.01 мкФ, С6 — 22 мкФ;**
- **резисторы R1 — 1 кОм, R2 — 300 кОм, R3, R13 — 100 кОм, R4, R5, R6, R7 — 22 кОм, R8 — 470 Ом, R9 — 2.2 кОм, R10 — 220 Ом, R11 — 15 кОм, R12 — 3 кОм, R14 — 1 кОм, R15 — 10 кОм (при доработке схемы заменяется на 100 Ом)**

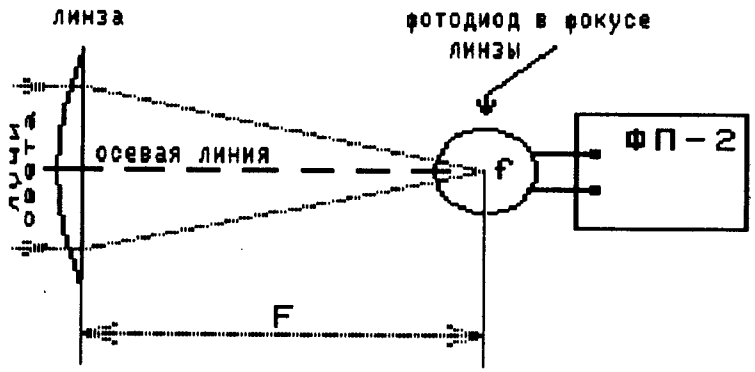


Рис. 2. Использование собирающей линзы. Обозначения: f — фокус линзы (точка, в которой концентрируются световые лучи, падающие на линзу параллельно), F — фокусное расстояние линзы

расстоянии около 10 см они исчезали вовсе. Следовательно, необходимо было увеличить чувствительность фотодатчика, сконцентрировав световые лучи на его поверхности с помощью собирающей линзы (рис. 2). Теперь фотоприемник стал реагировать на изображение курсора на расстоянии до двух метров. Однако выяснилось, что сигнал «попадания» появлялся на входе БК даже при неточном прицеливании (например, когда курсор находился в центре экрана, а оптическая ось линзы и датчика наводилась на его край). Причиной этого являлась слишком большая площадь чувствительной поверхности фотодатчика, из-за чего на его край попадали лучи, проходящие по краям линзы. Пришлось установить перед датчиком непрозрачную диафрагму с узким отверстием в центре (рис. 3; блок-схема фотопистолета в сборе показана на рис. 4). Теперь служебная строка стала заполняться точками (при работе приведенной выше «отладочной» программы) только в случае точного наведения оптической оси на курсор, что нам и требовалось. Можно начинать «охоту за курсором», набрав, оттранслировав и запустив следующую ассемблерную программу (фактически она является примером драйвера фотопистолета и понадобится при создании более сложных игр):

```
;    = ОХОТА - 1 =        (VIS1)
;
; вывод мишени - курсора, перемещаемого
; по экрану от левого края к правому и обратно
;
        MOV #VIS1,@#60   ; адрес программы обработки прерываний
        MOV #200, @#62   ; запрет других прерываний на время
                         ;   работы данной программы
        MOV #14,R0       ; очистка
        EMT 16           ;   экрана
        TST @#40         ; установка
        BNE 0            ;   режима
        MOV #233,R0      ;   32 символа
        EMT 16           ;   в строке
```

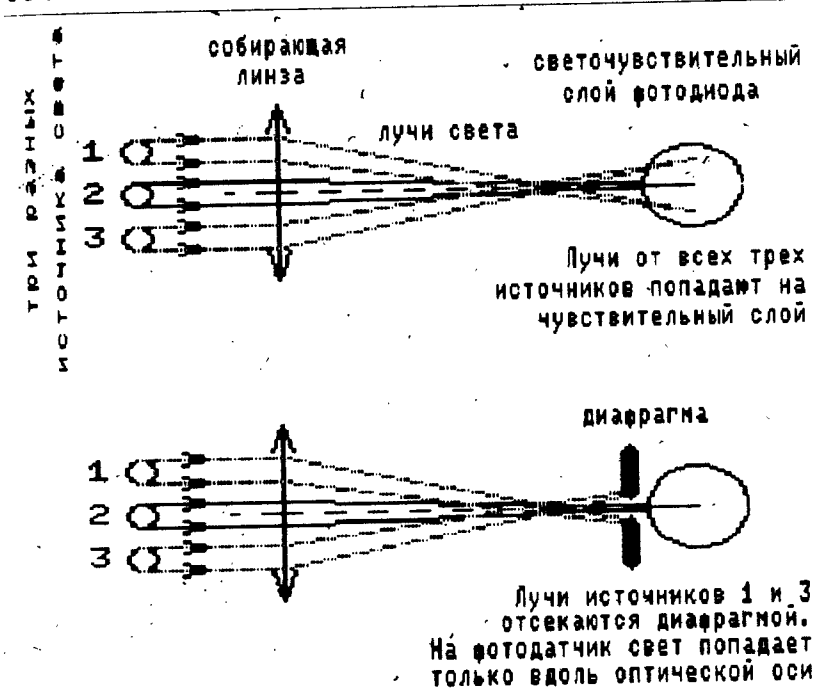
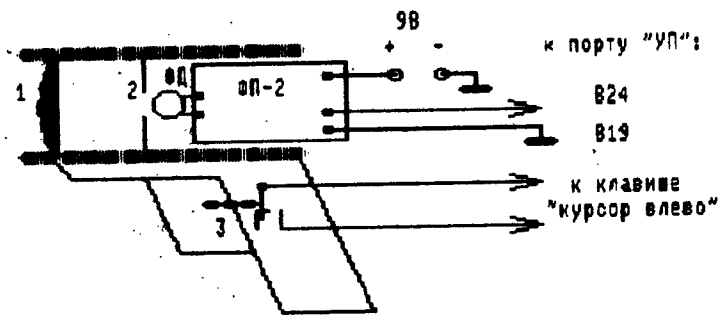



Рис. 3. Установка диафрагмы для повышения разрешающей способности системы (точности наведения)



1 - собирающая линза (F = 10...25 см; чем больше F, тем точнее нужно целиться);
2 - диафрагма (диск из фольги, в центре проколотый иглой);
3 - "курок" (кнопочный выключатель на замыкание).

Рис. 4. Фотопистолет в сборе

```
@:    INC X             ; увеличение координаты X
      JSR PC,CUR        ; вызов подпрограммы отображения
                        ;   курсора по координате X
      CMP #36,X         ; достигнута правая граница экрана?
      BNE @             ; еще нет - повторяем...
1:    JSR PC,CUR        ;
      DEC X             ; иначе начинаем уменьшать X
      BNE 1             ; достигнута левая граница экрана?
                        ; если еще нет, то повторить
      BR @              ;   иначе снова увеличивать координату X
;
CUR:  MOV X,R1          ; п/пр вывода курсора
      MOV #14,R2        ; с задержкой
      EMT 24
      MOV #17777,R4     ; константа задержки #17777
@:    NOP               ;   определяет скорость игры,
      SOB R4,@          ;   ее можно изменять программно
      RTS PC
;
X:    .#@               ; буфер координаты X
;
; собственно драйвер фотопистолета
;
VIS1: MOV #1000,R@      ; п/пр обработки прерывания при
@:    TST @#177714      ;   нажатии на "курок"
      BNE POP
      SOB R@,@
      BR RET
POP:  MOV #10,R1        ; есть попадание!
@:    MOV #7,R@         ; даем звуковой сигнал
      EMT 16
      SOB R1,@
      MOV #20,X         ; после попадания устанавливаем
                        ;   курсор - мишень в центр
                        ;   экрана (можно предусмотреть и
                        ;   случайный выбор координаты X)
RET:  RTI
```

Эта программа дает первоначальное представление о работе фотопистолета, но имеет два существенных недостатка. Во-первых, имитировать попадание (при отладке) можно только нажатием клавиши на клавиатуре и одновременным действием джойстиком (или замыканием подключенной к порту кнопки). А во-вторых, «попадание» будет фиксироваться и при наведении фотопистолета на любой светящийся объект, например, лампочку под потолком. К тому же «охотиться» за курсором на пустом экране неинтересно. В реальной игре можно изобразить «ландшафт» и выводить поверх него движущиеся спрайты с изображениями фигурок животных — вот тогда действительно получится настоящее «сафари». Однако теперь на экране присутствует множество светлых пятен и нам нужно научить программу отличать детали фона от мишеней. Этого можно добиться, применив следующий алгоритм для программы вывода цели.

Предположим, мишень (спрайт с изображением утки) «летит» мимо нарисованного на экране «камыша». После прицеливания играющий нажимает на курок. Узнав об этом, программа на мгновение выводит вместо спрайта темный прямоугольник тех же размеров, что и «утка» (или чуть больший), и проверяет состояние порта ввода-вывода (регистр 177714, естественно, должен содержать нулевой байт). Если же входной бит содержит логическую «1», значит, пистолет направлен на другую (освещенную) часть экрана — играющий промахнулся. Иначе программа переходит ко второму этапу проверки. В центре черного прямоугольника выводится светящийся квадратик размером с курсор и снова делается проверка порта ввода-вывода. Если там имеется сигнал с фотопистолета (логическая «1»), значит, утка подстрелена. Иначе — увы, придется зафиксировать промах...

После внесения соответствующих изменений программа обработки прерываний от фотопистолета (VIS2) и ее игровое «обрамление» стали выглядеть так:

```
;  = OXOTA - 2 =    (VIS2)
;
; вывод мишени - курсора, движущегося по экрану
; от левого края к правому и обратно
;
    MOV #VIS2,@#60  ; адрес подпрограммы обработки прерываний
      MOV #200,@#62 ; запрет других прерываний
      MOV #14,R@    ; очистка
      EMT 16        ;   экрана
      TST @#40      ; установка
      BNE @         ;   режима
      MOV #233,R@   ;   32 символа
      EMT 16        ;   в строке
@:    INC X         ; увеличение координаты X
      JSR PC,CUR    ; вывод курсора
      CMP #36,X     ; достигнута правая граница экрана?
      BNE @         ; еще нет - продолжаем...
1:    JSR PC,CUR    ;
      DEC X         ; иначе начинаем уменьшать X
      BNE 1         ; достигнута левая граница экрана?
                    ; еще нет - продолжаем уменьшать X
      BR @          ; иначе - повтор с начала
;
CUR:  MOV X,R1      ; п/пр вывода курсора
      MOV #14,R2    ;   с задержкой
      EMT 24
      MOV #17777,R4 ; константа задержки #17777
@:    NOP           ;   определяет скорость игры
      SOB R4,@      ;   и может быть изменена программно
      RET
;
X:    .#@           ; буфер координаты X
;
; собственно драйвер фотопистолета
```

```
VIS2: MOV #10,R5        ; выдаем звук
      MOV #7,R0         ; "выстрела"
1:    EMT 16            ;
      SOB R5,1          ;
      CLR R3            ; выводим вместо мишени
      JSR PC,CVA        ;   темный квадрат
      CLR R4            ; обнуляем буфер - сумматор
      MOV #2000,R5      ; 1000 - 3000 раз тестируем порт
0:    ADD @#177714,R4   ; суммируем значения в порту
      SOB R5,0          ;
      TST R4            ; если результат не нулевой,
      BNE RET           ;   то промах
      MOV #177777,R3    ; второй этап проверки -
      JSR PC,CVA        ;   вывод светлого квадрата
      CLR R4            ; обнуляем буфер - сумматор
      MOV #3000,R5      ; 2000 - 5000 раз тестируем порт
2:    ADD @#177714,R4   ; суммируем значения в порту
      SOB R5,2          ;
      TST R4            ; если результат равен нулю,
      BEQ RET           ;   то промах
; здесь можно на всякий случай повторить проверку
;   с выводом пустого квадрата или добавить вывод
;   спрайта, соответствующего попаданию
      MOV #6,R3         ; цикл
3:    MOV #232,R0       ;   из шести
      EMT 16            ;   миганий
      JSR PC,CUR        ;   курсора
      MOV #7,R0         ;   и шести
      EMT 16            ;   звуковых
      SOB R3,3          ;   сигналов
      MOV #20,X         ; установим курсор после попадания
                        ;   в середине строки (можно предусмотреть
                        ;  и случайный выбор координаты)
RET:  RTI

CVA:  MOV @#160,R4      ; определение расположения курсора
      MOV #12,R5        ; цикл из 10 проходов
0:    MOV R3,(R4)       ; заливка прямоугольной области
      ADD #100,R4       ;   размером с курсор цветом из R3
      SOB R5,0          ;
      RTS PC            ;
```

Особое внимание нужно обратить на суммирование значений из входного регистра порта ввода-вывода. Вначале автор пытался использовать простую команду TST @#177714, но при этом срабатывание фотопистолета (появление на входе БК логической «1») происходило не всегда, даже если приложить фотодатчик вплотную к экрану напротив мишени. Причину этого долго не удавалось понять, но наконец, после серии экспериментов с отладочной программой (листинг которой дан в начале статьи) она была найдена. Все дело оказалось в том, что изображение мишени НЕ ВСЕГДА присутствует на экране «физически»: ведь электронный луч телевизионной трубки сканирует по экрану в соответствии с разверткой, в результате чего любое изображение быстро-быстро мигает. Глаз благодаря его «световой инерции» эти

мигания не замечает, а вот фотодатчик «не видит» мишень, если электронный луч находится в другом месте. Поэтому для надежности вместо однократного тестирования порта производится накопление его состояний в буферном регистре — «сумматоре» с анализом полученной суммы: если она равна нулю, значит, фотодатчик действительно не освещен, иначе (если за время тестирования луч хотя бы раз «коснулся» датчика) — попадание будет зафиксировано.

Усвоив общий принцип работы алгоритма, можно написать полноценную игровую программу. Возможно, кому-либо удастся усовершенствовать предложенную идею проверки попадания, а может быть, — и аппаратную конструкцию. Своими находками предлагаю поделиться с другими читателями на страницах журнала.

Примечание редакции

Предложенный автором статьи способ изготовления фотопистолета на основе типового блока дистанционного управления телевизором обладает, по крайней мере, одним бесспорным преимуществом — максимальной доступностью для большинства пользователей БК. Такой блок можно приобрести в магазине или на радиорынке (при отсутствии упомянутого в статье фотоприемника ФП-2 можно попробовать взять аналогичное устройство другого типа) или собрать самостоятельно, благо используемые в его схеме радиодетали не дефицитны. Так что сложности возможны разве только при изготовлении оптической конструкции. Для облегчения работы можно, например, взять готовый пистолет от светотира, установив вместо лампочки и батарейки фотодатчик (в фокусе уже имеющейся там линзы) и блок ФП-2 (или выведя провода от датчика наружу). Кроме того, принципиально возможна адаптация к БК готовых фотопистолетов от приставки «Денди», как это не так давно было сделано в Самаре (к данному варианту мы вернемся чуть ниже).

Что же касается программной поддержки фотопистолета, хотелось бы дополнить статью следующими замечаниями.

Во-первых, следует признать очень удачной выбранную автором концепцию использования отдельного драйвера фотопистолета, «подвешенного» на прерывание и подстыковываемого к любой создаваемой ассемблерной программе. Желательно лишь в большей степени «специализировать» драйвер: он должен взять на себя все функции по определению точности попадания при нажатии на «курок» и возвращать в основную программу признак «попадание»/«промах» (например, значение R0, равное 177777_8 (-1_{10}) или 0, соответственно). Все остальные действия, в том числе по выводу спрайта «сбитой мишени» и звуковому сопровождению «выстрела», промаха и попадания, должны реализовываться исключительно в основной программе — только тогда драйвер фотопистолета будет действительно универсальным. (Размеры спрайта — мишени и сведения о его местоположении можно передавать в драйвер через специально зарезервированные ячейки ОЗУ — «блок параметров».)

Во-вторых, использование для вызова драйвера фотопистолета прерывания от клавиатуры (как и запараллеливание «курка» с какой-либо клавишей) неудобно, поскольку в реальной игре клавиши могут быть задействованы для других целей (скажем, для выбора сложности или «охотничьих угодий»). Поэтому лучше всего применить для этой цели редко используемое (по крайней мере, на БК-0010(.01)) прерывание по вектору 100, подключив «курок» к контакту В1 порта ввода-вывода. Кнопка «курка» при нажатии на нее должна замыкать провода, ведущие к контактам порта В1 и «общий» (А18, А19, В18 или В19). Возможен и другой вариант подключения, когда сигнал нажатия на кнопку «курка» вместе с сигналом фотодатчика подаются на выбранные для этой цели биты порта ввода-вывода, тогда вызов драйвера

фотопистолета необходимо производить в цикле игры с помощью команды JSR (именно так это сделано в самарской разработке).

И наконец отметим, что предложенный способ определения точности попадания по методу «мигающей мишени» благодаря его простоте и надежности пригодится и в других подобных случаях, например при разработке программной поддержки светового пера. Его, кстати, можно изготовить на базе того же блока ФП-2 по той же схеме, только линза в этом случае уже не понадобится — расстояние от светового пера до экрана вряд-ли будет превышать 2—3 см. А вот диафрагму, очевидно, придется оставить, если только само отверстие в корпусе «карандаша» не окажется достаточно узким. В качестве корпуса светового пера можно с успехом использовать пластмассовый «рейсфедер» для жидкого клея, какие несколько лет назад имелись в продаже почти во всех магазинах канцтоваров. Нужно только убрать из него «дозатор» — пластиковый толкатель с пружинкой, выпускающий при нажатии им на бумагу капельку клея. Один из вариантов конструкции светового пера показан на рис. 5. Переключение режимов «активен»/«неактивен» (что аналогично командам «поднять/опустить перо») можно осуществлять с помощью кнопочного выключателя, как показано на рисунке, или перекрывая световой поток дополнительной диафрагмой.

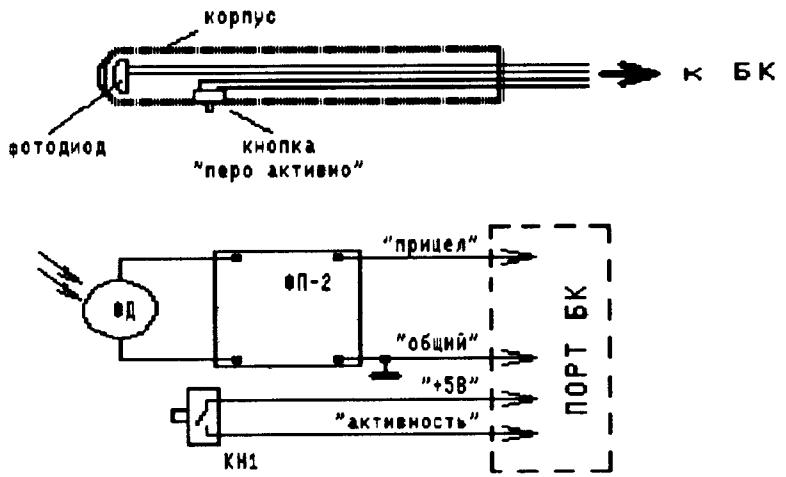


Рис. 5. Возможная конструкция светового пера для БК

При подключении светового пера придется подумать и над такой проблемой, как отслеживание курсором (приемлемого размера, скажем, 2 * 2 пиксела) перемещения светового пера по экрану. Идею «мигающей мишени» здесь можно использовать «в чистом виде», но алгоритм потребуется несколько усложнить. Он может быть, например, таким.

1. Проверка наличия сигнала «активен». Если кнопка отпущена, перо «активно», сигнал +5 В на разряд «активность» не подается и соответствующий бит регистра 177714 содержит «1»; иначе (при нажатой кнопке, что означает «неактивное» перо) — «0».

2. Проверка точности наведения пера на курсор (т. е. имеется ли на входе сигнал «попадания»). При его отсутствии — переход к третьему этапу.

3. Сканирование курсором окружающего пространства, смещая курсор на один шаг (равный ширине или высоте курсора, соответственно) влево, вправо, вверх, вниз и по диагоналям, каждый раз проверяя наведение пера. При неуспехе повторяем ту же операцию, увеличивая шаг, пока курсор не «упрется» в границы экрана или величина шага не превысит заранее оговоренного значения.

Другой возможный вариант определения точки, на которую наведено световое перо, приведен в книге «Электронные устройства для "Sinclair ZX Spectrum"» (М.: фирма «ВА Принт», 1993 г.).

1. Производится очистка экрана.

2. Перемещение светящейся горизонтальной линии (требуемой для надежного определения «попадания» толщины) по экрану сверху вниз, пока она не окажется в поле зрения фотодатчика. Координата линии Y запоминается как вертикальная координата точки наведения пера.

3. Экран очищается и аналогичным образом (путем перемещения вертикальной линии слева направо) определяется координата X.

4. Восстанавливается прежнее изображение на экране. Оно может быть предварительно (перед этапом 1) сохранено в памяти или на диске, либо перерисовано заново. На БК-0011(М) можно также воспользоваться переключением страниц видеоОЗУ: один «экран» рабочий (содержит рисуемую с помощью пера картинку), а второй — вспомогательный (для сканирования текущей точки пера).

Данный метод реализуется несколько проще предыдущего, но работает медленно и при движении линий требует перерисовки в наихудшем случае (когда перо находится в нижнем правом углу) всего экрана, тогда как «отслеживание» курсора при медленном движении светового пера предполагает «сканирующий обзор» лишь небольшой области в радиусе 1—3 «шагов» от прежней точки.

Впрочем, конкретные решения предоставим искать самим читателям, желающим подключить к своей БКшке световое перо. Мы же рассмотрим, как было обещано ранее, основные принципы подключения к БК фотопистолета от «Денди»-подобной игровой приставки «Nintendo Entertainment System» (данная информация приведена в девятом выпуске распространяемой через московский клуб любителей БК файловой «газеты» САПОГ). Схема подключения пистолета от самого «Денди» учитывая сходство этих приставок, очень похожа, если не полностью аналогична. Причем, вопреки распространенному мнению, АБСОЛЮТНО НИКАКОЙ ПЕРЕДЕЛКИ ПИСТОЛЕТА НЕ ТРЕБУЕТСЯ. Кабель пистолета распаивается на разъем порта БК в соответствии с таблицей:

Контакт порта	Разряд регистра @#177714	Маска (восьм.)	Назначение
A31	9	1000	Переключается из «0» в «1» и обратно, когда на фотоэлементе происходит смена темноты на свет
B32	10	2000	Признак нажатия на «курок»: «0» — не нажат, «1» — нажат (в фотопистолете от «Денди», подключенном по самарскому способу, — наоборот)
A8, B8, A9 или B9	—	—	Питание +5 В
A11, B11, A18, B18, A19 или B19	—	-	Общий

Исходная схема распайки фотопистолета для его подключения к «Денди» и контакты порта ввода-вывода БК показаны на рис. 6. (Будьте внимательны: можно «спалить» пистолет, если по ошибке перепутать полярность питания!) Номера используемых разрядов порта, в принципе, можно изменить, но тогда ваша распайка окажется не совместимой с самарскими играми «Утиная охота» и «Стрельба по тарелкам» (других же программ под пистолет пока не существует).

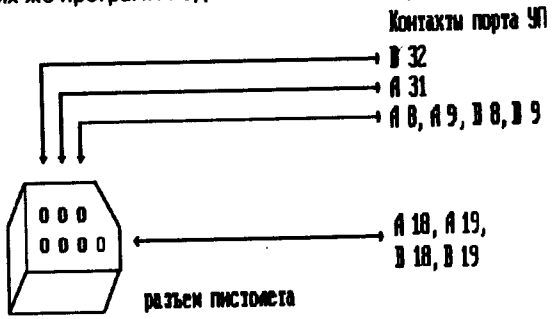


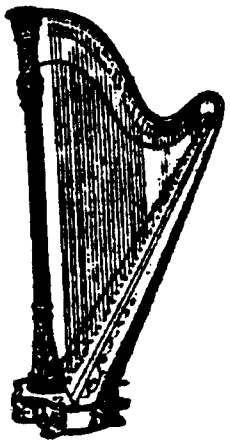
Рис. 6. Распайка фотопистолета от игровой приставки «Nintendo»

Алгоритм работы БК с фотопистолетом рассмотрим на примере программы «Утиная охота». Во время игры на экране изображен болотистый луг с зарослями камыша, откуда в случайный момент времени взлетает утка. Спрайт с ее изображением движется по экрану вправо-влево, постепенно смещаясь вверх и исчезая за верхним краем экрана, если утку не удалось подстрелить. Все это время компьютер циклически опрашивает порт, ожидая сигнала «выстрел».

Когда играющий нажимает на «курок», компьютер, зафиксировав появление логического «0» в бите 10 регистра 177714, начинает проверку точности попадания. Программа «Утиная охота» написана исключительно для БК-0011(М) и единственной причиной ее принципиальной непригодности для десятой модели является именно реализация этой проверки. Программа на мгновение выполняет переключение страниц видеопамяти, в результате чего «красивый фон» вместе со спрайтом утки с экрана исчезает, а на появившемся пустом (черном) поле на месте, где была утка, выводится белый прямоугольник того же размера. Программа фиксирует переключение бита 9 порта из «0» в «1» и обратно (для надежности прямоугольник мигает несколько раз) и вновь переключает страницы видеоОЗУ, возвращая на экран прежнюю картинку. Все это для играющего практически незаметно (если специально не присматриваться к экрану). При отказе же от переключения страниц и определении точности прицеливания по предложенному в статье методу «мигающей мишени» (с выводом черного и белого прямоугольников поверх фона и последующим восстановлением изображения утки и участка фона под ней), процесс несколько замедлится, но зато программу можно сделать универсальной — работоспособной на любой модели БК.

В заключение отметим, что фотопистолет плохо работает при наличии защитного стеклянного фильтра на экране, а также с монохромными мониторами зеленого изображения.

Кстати, почему бы не попытаться подключить к БК недавно разработанный для того же Nintendo шлем системы виртуальной реальности Virtual Boy? Приглашаем тех; кому удастся, сделать это, поделиться своим опытом на страницах нашего журнала.



В появившихся не столь давно суперсовременных электронно-музыкальных инструментах (ЭМИ) имеется, кроме прочего богатства возможностей, и такая: напев в микрофон какой-нибудь мотивчик, вы заносите в память ЭМИ соответствующую этому мотиву мелодию. (Получается даже, что композитору не требуется знать нотную грамоту — просто что хочешь спел и порядок!) Вот бы научить БКшку распознавать ноты с голоса — отличное получилось бы добавление к КЛАВЕСИНу или MAESTRO!

Оказывается, на БК возможно и это. Предлагаемая в данной статье аппаратная разработка представляет собой программно управляемый частотомер, измеряющий тон подаваемого на микрофон звукового сигнала достаточной длительности, будь то голос певца или звучание любого музыкального инструмента. И хотя программа поддержки блока измерения частоты звука позволяет лишь вывести на экран названия и точность прозвучавших нот, любители БК наверняка смогут реализовать на базе предложенного алгоритма множество интересных разработок.

А. Н. Романенко,

г. Лосино-Петровский, Московская обл.

БК-0010(.01): распознавание нот с голоса и музыки

Предлагаемое ниже устройство с подключенным к нему микрофоном позволит вам увидеть на экране дисплея компьютера БК-0010(.01) точность воспроизведения исполнителем или музыкальным инструментом нот в диапазоне от субконтроктавы до шестой октавы. При этом на экране указывается октава, нота и правильность данной ноты (от −50 до +50 центов* с точностью до одного цента). Если несколько раз сыграть или пропеть одну и ту же ноту, ее последнее значение точности запоминается и позже выводится на экран в виде таблицы точности настройки по всем сыгранным нотам. Результаты можно получить и в виде графика для всех нот с субконтроктавы по шестую октаву, что позволяет оценить правильность окончательной настройки музыкального инструмента (в качестве эталона в данной конструкции выступает кварцевый кристалл, с которым редкий камертон сравнится в точности воспроизведения звука).

* Цент — единица частотного интервала, равная 1/1200 октавы (применяется в музыкальной акустике). Так как октава содержит 12 нот (семь «основных» и пять диезов/бемолей), получаем, что один цент равен 1/100 диапазона между соседними нотами. Здесь каждая нота связана с «относительной шкалой», простирающейся на 50 центов в обе стороны (по частоте) от точечного музыкального тона. — *Прим. ред.*

Схема блока определения частоты нот и его программная поддержка

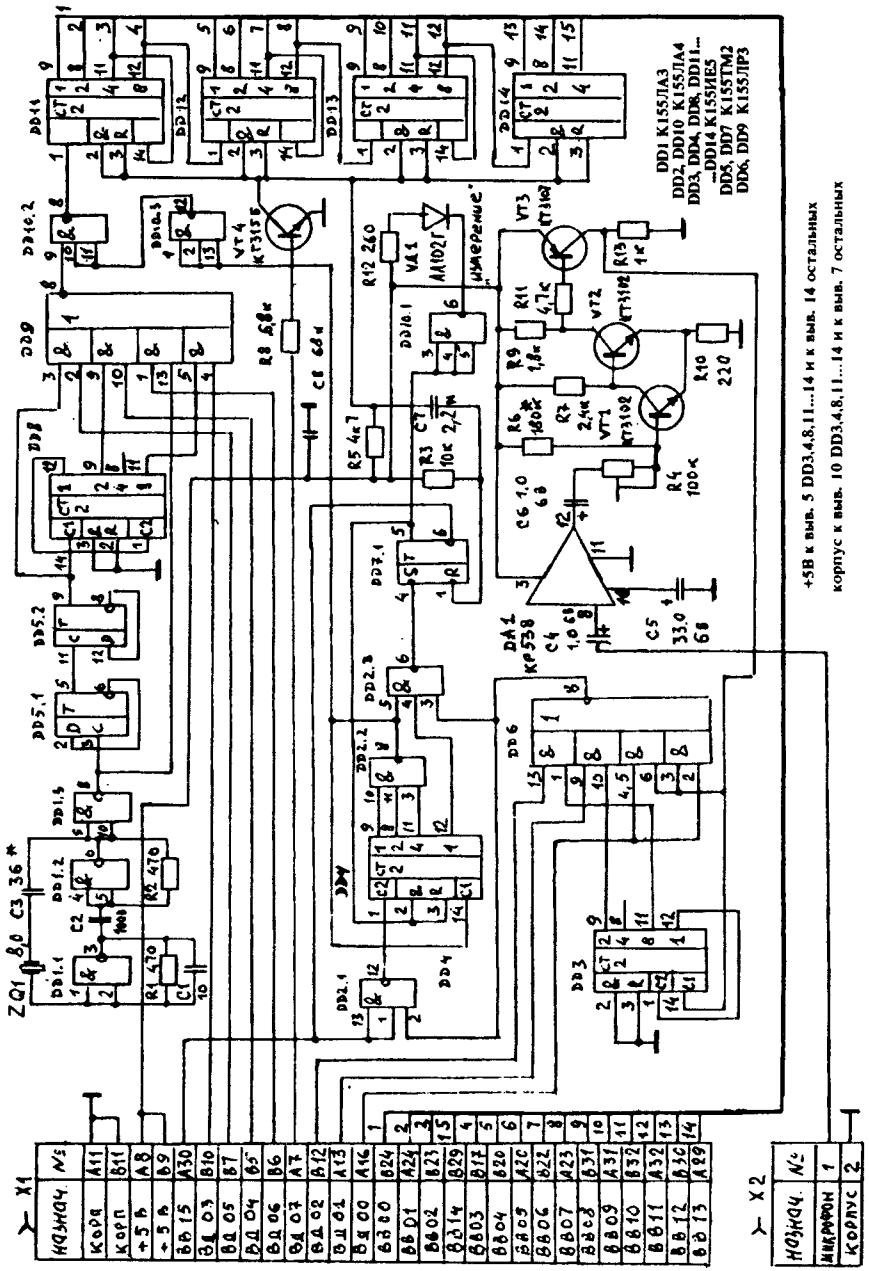
Блок определения частоты нот (будем далее называть его для краткости блоком микрофона, или БМ) преобразует звуковые колебания в двоичный код (сигналы уровней ТТЛ) и передает его на компьютер БК-0010(.01) для обработки программой «Нота». Последняя несложным способом вычисляет октаву, ноту, к которой поданный на микрофон звук ближе всего, и отличие этого звука от истинного музыкального тона для данной ноты в центах.

После запуска программа «Нота» находится в состоянии ожидания. Поданный на микрофон звук анализируется блоком БМ на регулярность воздействия, выполняется предварительное измерение частоты тона, результаты имерения в двоичном коде запоминаются и передаются программе. Она считывает код и, обработав его, устанавливает режим БМ для точного измерения частоты тона. Блок БМ выполняет точное измерение, запоминает результаты и снова передает программе, которая обрабатывает код и выводит на экран название октавы и ноты, а также неточность тона последней по сравнению с истинным для данной ноты в центах). Измерение тона и обновление сообщения на экране произволится в течение всего времени звучания ноты.

Принципиальная схема БМ представлена на рисунке. Генератор импульсов стабильной частоты собран на элементах DD1, R1, R2, C1, C2, C3 и ZQ1 (кварц). Конденсатором C3 на выходе генератора устанавливается частота, равная 8 МГц. Далее импульсный сигнал данной частоты поступает на делители DD5 и DD8. С генератора и с делителей импульсы с частотами 8 МГц, 2 МГц, 500 кГц и 125 кГц подаются на схему выбора частоты (DD9), управляемую программно через порт ввода-вывода БК подачей положительного напряжения на один из разрядов ВД03—ВД06. Очевидно, что на выход DD9 проходят импульсы с частотой, поступающей к тому элементу или, на который с порта ввода-вывода БК подано положительное напряжение. Через элемент DD10.2 они поступают на пятнадцатиразрядный счетчик импульсов (DD11—DD14). Сам же элемент DD10.2 открывается через DD10.3 импульсом с выхода генератора строба (DD2, DD3, DD4) на время, кратное периоду звукового сигнала, поступающего на микрофон. Длительность строба, таким образом, определяется частотой тона голоса или музыкального инструмента и положительным напряжением на одном из разрядов порта ввода-вывода БК ВД00—ВД02.

Сигнал с выхода микрофона преобразуется в импульсы уровней ТТЛ с частотой, равной частоте звукового тона, подается на усилитель DA1, преобразуется триггером Шмитта на VT1, VT2 и далее с усилителя на VT3 импульсы с частотой следования Ft подаются на делитель частоты на DD3. С выходов DD3 и с VT3 импульсы с частотами Ft, 4Ft и 16Ft подаются на схему выбора частоты DD6, управляемую программно через порт ввода-вывода БК подачей положительного напряжения на один из разрядов ВД00-ВД02. На выход DD6 проходят импульсы той частоты, которая подана на вход элемента или, куда с порта вывода подано положительное напряжение.

Готовность к измерению программа инициирует кратковременной подачей высокого уровня на разряд ВД07 порта ввода-вывода. Это положительное напряжение открывает транзистор VT4, низкий уровень на его коллекторе



сбрасывает счетчик DD11—DD14, и через С7 сбрасывает триггер DD7.1. С выхода 6 последнего выский уровень поступает на разряд ВВ15 порта ввода-вывода и запрещает программе считывание показаний счетчика, а через входы 13 и 1 DD2.1 разрешает прохождение импульсов поступающего на микрофон тона на выход делителя DD4. В момент спада на входе 1 DD4 седьмого по счету импульса на выходе 8 DD4 и входе 11 DD2.2 появляется низкий уровень. Он «взводит» отдельный триггер DD4, на выходе 12 которого появляется высокий уровень и подается на вход 4 DD2.3.

Низкий уровень с выхода 8 DD2.2 через DD10.3 открывает DD10.2 для прохождения импульсов стабильной частоты с DD9 на счетчик DD11—DD14. В момент спада DD4 восьмого по счету импульса тона на выходе 8 DD2.2 появляется высокий уровень и закрывает DD10.2. Заполнение счетчика DD11—DD14 прекращается. В момент появления на выходе 8 DD2.2 высокого уровня на выходе 6 DD2.3 появляется низкий уровень и устанавливает триггер DD7.1. Высокий уровень с выхода 5 DD7.1 сбрасывает DD4. Низкий уровень на выходе 6 DD7.1 запрещает по входам 13 и 1 DD2.1 прохождение импульсов тона на вход DD4, а в разряде ВВ15 порта ввода-вывода появляется сигнал, который разрешает программе считать информацию из счетчика DD11—DD14.

После считывания показаний счетчика программа вновь устанавливает высокий уровень в разряде ВД07 порта ввода-вывода. Схема формирования строба переходит в исходное состояние и при подаче на микрофон звукового сигнала весь цикл измерений повторяется. Таким образом, измерение частоты тона производится подсчетом количества импульсов стабильной частоты, генерируемых за время длительности импульса строба.

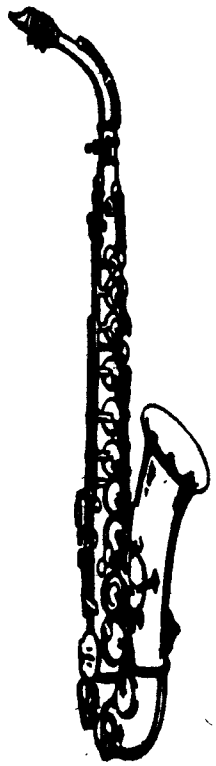
Для первого, оценочного, измерения программа устанавливает длительность строба равной одному периоду колебаний исходного тона, а стабильную частоту импульсов — 125 кГц. При таких условиях в счетчике после первого измерения будет содержаться число от 8 для верхней ноты шестой октавы до 31743 для нижней ноты субконтроктавы. По результатам обработки этого числа для точного измерения программа устанавливает стабильную частоту импульсов и длительность строба такими, чтобы в счетчик поступило не менее 8060 импульсов, что обеспечивает измерение частоты тона с точностью не менее 0,3 цента.

Блок БМ собран автором статьи в корпусе от блока МСТД из недефицитных деталей. При настройке может возникнуть необходимость подбора резисторов R4 и R6 для устойчивой работы триггера Шмитта, а также С3 для установки частоты кварцевого генератора равной 8 МГц.

```
10 CLS
20 ? CHR$(155)
30 DIM M(12,15),L(11),D$(23)
40 ? AT(16,4)"Н А С Т Р О Й К А";
   AT(18,7)"М У З Ы К А Л Ь Н Ы Х";AT(12,10)
   "И Н С Т Р У М Е Н Т О В";AT(18,13)
   "И Г О Л О С А .";AT(1,22)"При выкл.БК подключите
   контроллер 'нота', и к нему - микрофон."
50 CLEAR
60 RESTORE
70 FOR I=1 TO 22
80 READ D$(I)
```



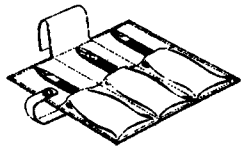
```
90 NEXT I
100 IF INKEY$=""GOTO100
110 CLS
120 L2=LOG(2)
130 G!=EXP(L2/12)
140 G1!=EXP(L2/24)
150 CN!=EXP(L2/1200)
160 D0!=27.5/G!^9/G1!
170 V7=LOG(10)
180 L8=274
190 H=0
200 CLS
210 ?AT(20,9)"Расстройка ноты в центах";AT(34,6)"|";AT(9,7)
    "  -50ц.    -10    -3    -+1   +3     +10      +50ц."
220 Y=0
230 IF H<10 THEN H=H+1 ELSE H=H+10
240 L9%=99*LOG(H)/V7
250 T=0
260 PSET (273-L9%,63+T),2
270 PSET (277+L9%,63+T),2
280 T=T+1
290 IF T<4 THEN 260
300 IF H>49 THEN 320
310 GOTO 230
320 OUT &O177714,&B111111111,1%
330 ?AT(0,10)"Ожидание тона";
340 OUT &O177714,&B1100010001,0%
350 ?AT(2,5)". .";"";". ."
360 ? AT(2,5)"   ";CHR$(162+X);"   "
370 C=INP(&O177714,&B1000000000000000)
380 IF X<30 THEN X=X+1 ELSE X=0
390 IF C><-32768 GOTO 350
400 N=INP(&O177714,&B111111111111111)
410 N1=32767-N
420 IF N1<8 OR N1>31743 THEN 320 ELSE ZZ=0
430 Y%=INT(LOG(N1/31)/LOG(4))
440 OUT &O177714,&B111111111,1%
450 I=1
460 ON 5-Y% GOTO 470,490,510,530,550,570
470 OUT &O177714,&B1100010001,0%
480 GOTO 580
490 OUT &O177714,&B1100010001,0%
500 GOTO 580
510 OUT &O177714,&B1101000001,0%
520 GOTO 580
530 OUT &O177714,&B1110000001,0%
540 GOTO 580
550 OUT &O177714,&B1110000010,0%
560 GOTO 580
570 OUT &O177714,&B1110000100,0%
580 K!=16/4^(Y%+1)
590 S=INP(&O177714,&B1000000000000000)
600 ZZ=ZZ+1
610 IF ZZ>99 GOTO 320
620 IF S><-32768 GOTO 590
630 NN=INP(&O177714,&B1111111111111111)
640 N2=32767-NN
650 IF N2<8060 OR N2>32686 THEN 320
```



```
660 O%=FIX(LOG(8E6*K!/N2/D0!)/L2)+1%
670 IF O%<1 OR O%>10 THEN 320
680 ? AT(1,21)"
690 IF O%=U THEN 730
700 ? AT(18,2)"
710 U=O%
720 D0!=D0!*2^(O%-1)
730 F!=7998785*K!/N2
740 R%=FIX(LOG(F!/D0!)/LOG(G!))+1%
750 IF R%<1 OR R%>12 THEN 320
760 ? AT(0,10)" настройка    ";
770 ?AT(18,2)D$(O%);" Октава"
780 IF Q=R% THEN 820
790 ? AT(39,2)"
800 Q=R%
810 ? AT(39,2)",   ";D$(R%+10)
820 W!=LOG(ABS(F!/(D0!*G!^(R%-1))))/LOG(CN!)
830 N2!=INT(W!+.5-50)
840 M(U,Q)=N2!
850 IF N2!><0 THEN L%=99*LOG(ABS(N2!))/V7 ELSE L%=0
860 IF N2!>=0 THEN L%=ABS(L%) ELSE L%=(-1)*ABS(L%)
870 ?AT((L8)/8,5)" "
880 LINE (275,60)-(L8,60),0
890 L8=275+L%
900 ?AT((L8)/8,5)"*"
910 LINE (275,60)-(L8,60),1
920 IF U=U1 AND Q=Q1 THEN 440
930 U1=U
940 Q1=Q
950 Y=Y+1
960 IF Y<26 GOTO 320
970 K$=INKEY$
980 ? AT(2,20)"Для вывода таблицы и графика неточности выполненной
    настройки нажмите '7', для продолжения настройки - ПРОБЕЛ."
990 IF K$="7" THEN 1020   '1993,03,21, А.Н.Романенко
1000 IF K$=CHR$(32) GOTO 190
1010 IF K$<>"7" OR K$><CHR$(32) GOTO 970
1020 CLS
1030 ? ,,,,,,,,"        Таблица значений неточности настройки
    в центах.",,
1040  ? ""
1050 ?"+-------+-------------------------------------------------+"
1060 ?"| нота  |                  о  к  т  а  в  ы                |"
1070 ?"|       | с-к. конт бол. нал.  1    2    3    4    5    6 |"
1080 ?"+-------+----+----+----+----+----+----+----+----+----+----+"
1090 FOR I=1 TO 12
1100 ?"|";D$(10+I);TAB(10);"|";
1110 FOR Y=1 TO 10
1120 L%=M(Y,I)
1130 IF M(Y,I)<0 AND M(Y,I)>(-1) THEN ?"-";
1140 IF M(Y,I)>0 AND M(Y,I)<1 THEN ?"+";
1150 IF M(Y,I)=0 THEN ?""; ELSE ?L%;
1160 ?TAB(10+Y*5);"|";
1170 NEXT Y
1180 ?""
1190 NEXT I
1200 ?"+-------+----+----+----+----+----+----+----+----+----+----+"
```



```
1210 ? AT(5,22)"Вывод графика-'7'; продолжение настройки- ПРОБЕЛ,
    настройка с обновлением графика и таблицы -'2'.";
1220 K$=INKEY$
1230 IF K$=CHR$(32) THEN 190
1240 IF K$="2" THEN 50
1250 IF K$="7" THEN 1270
1260 IF K$<>"7" OR K$<>CHR$(32) OR K$<>"2" GOTO 1220
1270 CLS
1280 ? " Логарифмический график расстройки нот по октавам:
    (в центах)      субк  контр бол   нал    1-я    2-я    3-я
    4-я   5-я    6-я"
1290 V7=LOG(10)
1300       ? AT(0,2)"40";AT(0,6)"10";AT(0,11)"1 ";AT(0,10)"+2";AT(0,12)
    "центы";AT(0,13)"-2";AT(0,17)"10";AT(0,21)"40";AT(4,22)
    "в окт: до,до-д,ре,ре-д,ми,фа,фа-д,соль,соль-д,ля,ля-д,си"
1310 FOR I=0 TO 123
1320 K=0
1330 PSET (20+4*I,121-K),3
1340 K=K+1
1350 IF K<3 THEN 1330
1360 NEXT I
1370 I=0
1380 H=0
1390 IF H<10 THEN H=H+1 ELSE H=H+10
1400 L9%=59*LOG(H)/V7
1410 PSET (20+4*I,120-L9%),2
1420 PSET (20+4*I,120+L9%),2
1430 IF H>40 THEN 1450
1440 GOTO 1390
1450 I=I+12
1460 IF I>120 THEN 1480
1470 GOTO 1380
1480 FOR I=1 TO 10
1490 FOR J=1 TO 12
1500 IF M(I,J)><0 THEN L%=59*LOG(ABS(M(I,J)))/V7 ELSE L%=0
1510 IF M(I,J)>=0 THEN L%=ABS(L%) ELSE L%=(-1)*ABS(L%)
1520 L7=10+(J+12*(I-1))*4
1530 L8=120-L%
1540 LINE (L7,L8)-(L7,120),1
1550 NEXT J
1560 NEXT I
1570 ? AT(2,23)"Продолж.настр.ПРОБЕЛ,- с обновл.граф.и табл.-'2';
    см.табл.'7'.";
1580 K$=INKEY$
1590 ? AT(3,1)" убк   онтр  ол    ал    -я    -я    -я    -я
    -я    -я"
1600 IF K$="7" THEN 1020
1610 ? AT(3,1)"субк  контр бол   нал   1-я   2-я   3-я   4-я
    5-я   6-я"
1620 ? AT(0,12)"     ";AT(0,22)"          "
1630  IF K$="2" THEN 50
1640  IF K$=CHR$(32) THEN 190
1650 ?AT(0,12)"центы";AT(0,22)"в октавах"
1660 IF K$<>"7" OR K$<>CHR$(32) OR K$<>"2" GOTO 1580
1670 DATA СУБКОНТР,КОНТР,БОЛЬШАЯ,МАЛАЯ,1-я,2-я,3-я,4-я,5-я,6-я,
     ДО,ДО диез,РЕ,РЕ диез,МИ,ФА,ФА диез,СОЛЬ,СОЛЬ диез,
     ЛЯ,ЛЯ диез,СИ
```



Завершаем начатую в предыдущем выпуске публикацию статьи о турбировании БК. Теперь разговор пойдет о модели БК-0011(М).

С. М. Неробеев, А. В. Сорокин,

Москва

Увеличение тактовой частоты БК-0010(.01)/БК-0011(М)

Способы повышения тактовой частоты БК-0011(М)

Прежде чем приступить к описанию конкретных способов проведения доработки, хотелось бы сделать несколько общих замечаний по этому вопросу. Во-первых, повышение тактовой частоты возможно только до 6 МГц, так как реализация доработки, рассчитанной на любую другую частоту в диапазоне от 4 до 6 МГц (это может быть специальный делитель или дополнительный генератор) крайне затруднена и может привести к неоправданному усложнению схемы. Во-вторых, вопрос увеличения быстродействия для пользователей БК-0011(М) не столь актуален, как на БК-0010(.01), поскольку рабочая частота одиннадцатой модели выше (составляет 4 МГц по сравнению с 3 МГц на «десятке»). Таким образом, после удачного проведения доработки тактовая частота оказывается увеличенной не вдвое, как на БК-0010(.01), а всего лишь в 1.5 раза. Тем не менее, такое турбирование все же достаточно заметно, что делает указанную доработку привлекательной для большого числа пользователей БК-0011(М).

Все изложенные в первой части данной статьи замечания, (см. «Персональный компьютер БК-0010 — БК-0011М» № 1 за 1996 г.), о работе БК-0010(.01) с тактовой частотой 6 МГц абсолютно справедливы и для БК-0011(М). Однако здесь распространение получили другие слухи о том, что процент работающих с повышенной тактовой частотой БК-0011(М) гораздо меньше, чем БК-0010(.01). Исходя из собственного богатого опыта, авторы утверждают, что это заблуждение не имеет ничего общего с действительностью: данная доработка успешно проводится на каждой второй-третьей вычислительной машине, т. е. соотношение работающих и не работающих с повышенной тактовой частотой машин такое же, как и для БК-0010(.01).

Идея описываемой доработки для БК-0011(М) та же, что и для БК-0010(.01) (см. № 1 за 1996 г.). Как вы помните, тактовая частота 12 МГц от генератора последовательно делится сначала до 6, а затем до 3 МГц. Таким образом, искомая частота имеется на системной плате БК-0010(.01) и вся доработка сводится к перерезанию существующих печатных дорожек и установке новых связей. Следует заметить, что частота 6 МГц не является только промежуточным результатом деления, она требуется для работы микросхем видеоконтроллера и контроллера клавиатуры, а также для некоторых

других целей. То же справедливо и для БК-0011(М), но так как рабочая частота этой вычислительной машины равна 4 МГц, при делении 12 МГц на 3 частота 6 МГц как промежуточный результат отсутствует. Поэтому, кроме делителя на 3, собранного на микросхеме К155ТМ2 (D39), на системной плате БК-0011(М) присутствует еще и специальный делитель на 2, в качестве которого используется один из элементов микросхемы К531ТВ9 (D8.1). Следовательно, и в этом случае доработка не требует установки дополнительных микросхем и сводится к перерезанию имеющихся печатных дорожек и установке новых связей.

Между тем, многие не знают о существовании на плате делителя на 2 и пытаются установить собственный, благо схема такого делителя достаточно проста (в качестве него может быть использован любой D- или JK-триггер). Типичным примером является доработка, предложенная С. Даниловым из Павловского Посада. Кроме уже указанного недостатка, он допустил еще и ошибку в схеме: для обеспечения работы D-триггера в режиме делителя следует подключать его инверсный выход не к тактовому входу, а к выводу D. К тому же, согласно Павлово-Посадской схеме, повышенная тактовая частота подается только на процессор, для чего приходится отгибать его первый вывод, что само по себе нежелательно. Микросхемы же, обслуживающие процессор, продолжают работать на тактовой частоте 4 МГц, в результате чего возникает значительное рассогласование между некоторыми управляющими сигналами, поступающими от внешних устройств и вырабатываемыми самим процессором. Это, в свою очередь, приводит к зависанию КНГМД. Возможно, данная доработка обеспечивает работу с синхронизированным от вычислительной машины контроллером, так как на разъем МПИ продолжает поступать 4 МГц, однако ни указанный способ синхронизации контроллера, ни сама данная доработка не являются схемотехнически грамотными. Поэтому рекомендовать ее для повторения читателям, по мнению авторов, нельзя.

В одной из предыдущих статей («Доработки контроллера дисковода», № 6 за 1995 г.) мы уже предостерегали читателей, контроллер которых синхронизирован от вычислительной машины, о недостатках такого способа синхронизации. Еще одним аргументом против него является то, что подобный КНГМД не сохраняет работоспособность на тактовой частоте 6 МГц.

Схема доработки, предлагаемая авторами этой статьи, приведена на рис. 1. Доработка проводится следующим образом. На системной плате перерезается дорожка ведущая от вывода 9 ножки микросхемы К155ТМ2 (D39). Данная микросхема находится в левой нижней части платы, а указанная дорожка выходит непосредственно из-под микросхемы D39 в направлении разъемов для подключения клавиатуры. Затем следует подпаять три монтажных провода (к выводу 5 D8, выводу 9 D39 и к месту перехода ранее перерезаной дорожки на нижнюю сторону печатной платы) и подсоединить их к переключателю согласно рис. 2. В отличие от аналогичной доработки для БК-0010(.01), здесь используется однорядный переключатель. Поскольку большинство таких переключателей имеет небольшие габариты, его целесообразно установить на задней панели корпуса вычислительной машины справа от разъема питания (симметрично кнопке перезапуска).

Следует заметить, что при работе с повышенной тактовой частотой перед пользователями БК-0011(М) может возникнуть ряд специфических трудностей. В первую очередь, в отличие от БК-0010(.01), которая в случае удачного

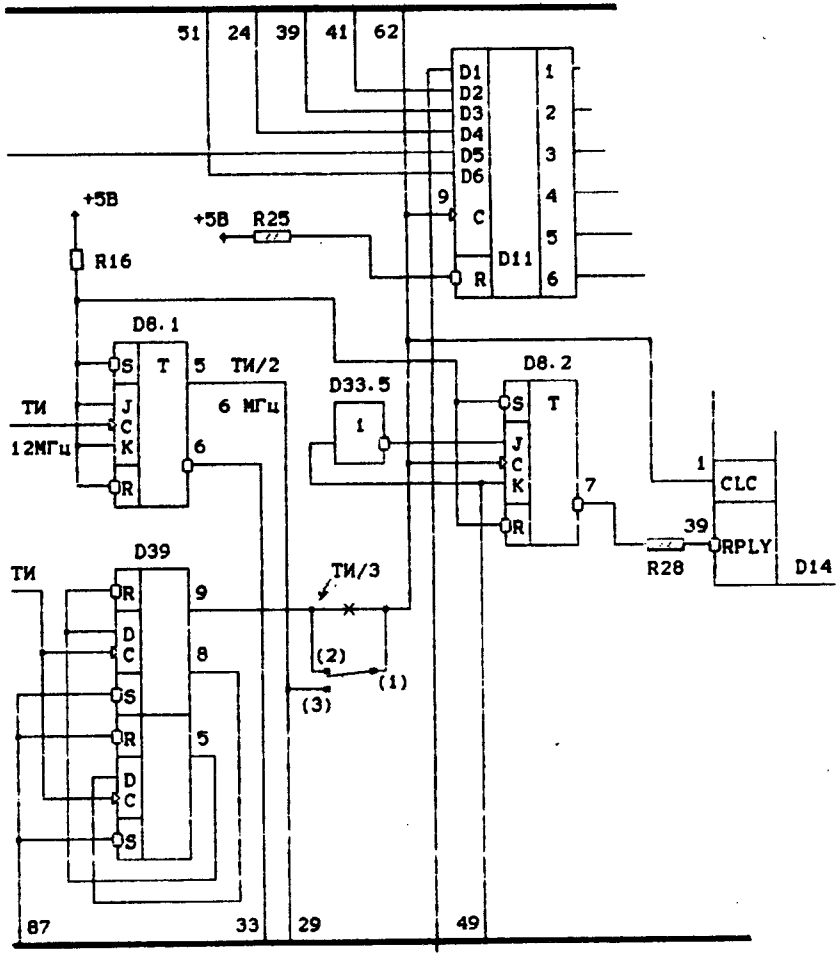


Рис. 1

проведения аналогичной доработки функционирует в турбо-режиме достаточно устойчиво, БК-0011(М) может периодически «подвисать». Помимо уже указанной в № 1 за 1996 г. причины (из-за микросхем видеоконтроллера-контроллера ОЗУ и контроллера клавиатуры), которая для даного случая не столь актуальна, еще одним слабым местом БК-0011(М) являются микросхемы ПЗУ (возникновение

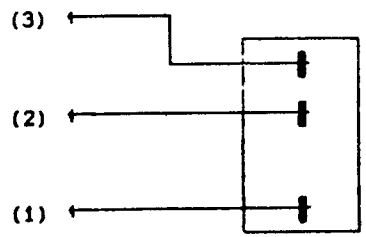


Рис. 2

между управляющими сигналами процессора и сигналом ответа внешнего усройства незначительного рассогласования, величина которого зависит от конкретного образца микросхемы памяти). Поскольку на системной плате присутствуют только две микросхемы ПЗУ с прошивкой базовой операционной системы, наиболее существенное влияние здесь оказывает дополнительная плата ПЗУ с прошивкой языка программирования БЕЙСИК, особенно если в разъемы этой платы (так называемую кассету ПЗУ) установлены дополнительные микросхемы. Именно она, как правило, является причиной нестабильной работы БК-0011(М) с повышенной частотой. Для проверки этого утверждения достаточно отключить плату ПЗУ, отсоединив разъем XT8 от системной платы. Если же БЕЙСИК требуется вам для работы, наиболее простым решением проблемы является подключение между шинами сигнала ответа RPLY и питания +5 В (соответственно, это контакты 02 и 24 разъема XT8) конденсатора с емкостью около 20 пФ, если кассета ПЗУ пустая либо в ней установлены микросхемы КР1801РЕ2, или около 40 пФ для микросхем с электрическим стиранием типа КМ1801РР1 (так как эти ПЗУ обладают большим быстродействием по сравнению с обычными масочными). Тем самым мы добиваемся дополнительной задержки и увеличения длительности сигнала ответа, а несколько необычное включение емкости объясняется тем, что данный сигнал инверсный (активный уровень — низкий). Точные значения емкостей подбираются для каждого конкретного случая индивидуально.

Вообще говоря, используя предложенную выше идею, можно добиться устойчивой работы на частоте 6 МГц для любого конкретного экземпляра как БК-0010(.01), так и БК-0011(М). Однако, как уже говорилось ранее, этот способ достаточно сложен, в частности, введение слишком большой межшинной емкости может вызвать выход из строя микросхем, подключенных к данной шине.

Еще одной характерной проблемой, иногда возникающей при работе с повышенной тактовой частотой как на БК-0010(.01), так и на БК-0011(М), является исчезновение автоповтора, а также появление дребезга клавиатуры (или его значительное усиление на тех вычислительных машинах с полноходовой клавиатурой, где он уже был). Эта проблема, как и предыдущая, решается установкой специальных конденсаторов. На этот раз подбора (в сторону увеличения емкости) требуют конденсаторы С3 и С4, подключенные, соответственно, к выводам 20 и 22 микросхемы контроллера клавиатуры КР1801ВП1-014. (Они имеют одинаковое обозначение и одинаковые номиналы на БК-0010(.01) и на БК-0011(М).) Так же, как и в предыдущем случае, существует предел в увеличении их емкостей — они не должны превышать 330 пФ. Данный способ может быть применен и для решения более общей проблемы — устранения дребезга, в том числе и при работе с обычной тактовой частотой. Единственным недостатком этого способа является некоторое замедление работы с клавиатурой, тем большее, чем больше емкость установленного вами конденсатора.

В заключение хотелось бы сделать общее замечание относительно переключения вычислительной машины в турбо-режим и обратно. Проведение такой операции при включенном питании без дополнительных мер предосторожности может привести к зависанию процессора. Обычной рекомендацией здесь является введение переключателя «Пауза», который временно приостанавливает работу процессора на момент переключения тактовой

частоты, а затем позволяет продолжить работу с той же машинной команды, на которой она была прервана. Аппаратная приостановка работы процессора может быть полезна и во многих других случаях, поэтому мы расскажем о ней в одном из ближайших номеров. Однако существует более простой способ: нажатием на кнопку перезапуска вызывается останов процессора, производится переключение тактовой частоты, а затем кнопка «RESET» отпускается. Недостатком такого способа является повторная инициализация вычислительной машины, однако на БК-0010(.01) и БК-0011(М), оснащенном перезапуском по адресу 100000, в большинстве операционных систем происходит лишь выход в файловую оболочку и повторное чтение каталога диска, при инициализации же монитора БК-0011(М) возврат в оболочку осуществляется достаточно простой последовательностью команд (например **4;1С 100000G** для ANDOS и MKDOS).

ДОПОЛНЕНИЯ К РАНЕЕ ОПУБЛИКОВАННЫМ СТАТЬЯМ

1. Если в Vortex 4.0 при установленном в последней строке курсоре нажать «АР2» + «ЗАБОЙ», эта программа может испортить FAT дискеты. Будьте осторожны!

2. Как определить из М-файла, какая из файловых панелей активна в данный момент: левая или правая? Вам поможет содержимое ячейки XB (из параметров файловой панели, см. «Персональный компьютер БК-0010 — БК-0011М», № 1 за 1996 г., с. 41), которое равно 1 при активной левой панели или 41 — при активной правой.

3. Операционная система MK-DOS некорректно работает с логическими дисками, созданными системой NORD. Запись файла в такой логический диск средствами ОС MK-DOS приводит на винчестере к непредсказуемым последствиям (в чем NORD, вопреки распространенному мнению, вовсе не виноват).

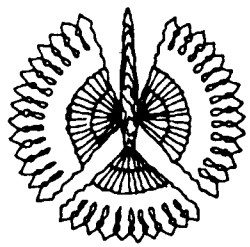
*** Внимание! Опечатка ***

В статье А. Г. Прудковского «Драйвер принтера для БК» (№ 1 за 1996 г.) в листинге на стр. 96 вместо правильного адреса начала подпрограммы сохранения регистров в стеке 110346 ошибочно записано число 100346.

Следует читать:

```
        . . . . . .
AU:  MOV @#4,-(SP)      ; запоминание вектора СТОП
     JSR R4,@#110346    ; запоминание всех регистров
     MOV R5,-(SP)
        . . . . . .
```

ОБМЕН ОПЫТОМ



В данной статье рассказывается о реализации вывода информации о загружаемом с магнитной ленты файле одновременно с процессом самого чтения в режиме псевдопараллельного исполнения двух ассемблерных подпрограмм с использованием прерывания по Т-разряду (вектор 14). Кроме того, в статье подробно рассматривается типовой алгоритм загрузки и роль подпрограмм из ПЗУ монитора, принимающих участие в чтении файлов, что может оказаться полезным при создании читателями собственных программ для работы с магнитофоном.

С. А. Мальцев,

Москва

Использование прерывания по Т-разряду в копировщиках

Об использовании прерывания по вектору 14 (Т-разряд) уже не раз говорилось на страницах журналов (см., например, [1] и [2]). Традиционно оно применяется в отладчиках для реализации пошагового выполнения программ (трассировки). Другой вариант — осуществление режима псевдопараллельного выполнения нескольких программ, который открывает широкие возможности в области сопровождения динамических игр музыкой и т. п. ([2]).

Намного менее известно применение прерывания по вектору 14 в магнитофонных загрузчиках и копировщиках. Один из примеров такого приема программирования в загрузчике — защищенная версия игры «ПЕРЕВАЛ» (BIL corp.). Эта игра состоит из загрузчика и трех подгружаемых в него файлов, записанных на кассету в нестандартном формате. Причем загрузчик обеспечивает псевдопараллельное выполнение сразу трех процессов: загружает заставку игры в память с магнитофона (с диска эта игра не запускается), отображает на дисплее в цифровой форме время, оставшееся до окончания загрузки, и визуализует процесс загрузки путем имитации «синклеровского» мигания бордюра дисплея. Аналогично можно использовать Т-разряд и в копировщиках. В подавляющем большинстве существующих магнитофонных копировщиков для БК-0010(.01) имеется команда чтения первого найденного файла (в стандартном формате БК). Но во время ее выполнения пользователь не знает, какой файл загружается, так как процессор занят чтением программы и не может «отвлечься» на распечатку имени. Псевдопараллельное же исполнение подпрограмм распечатки информации о файле и считывания файла в память позволяет обойти эту трудность и повысить уровень сервиса копировщиков.

Общий алгоритм выполнения команды чтения первого найденного файла при использовании прерывания по Т-разряду выглядит так:

- установка векторов прерывания и подготовка к выполнению «псевдопараллельного» чтения файла, выполняемого в пункте 4;
- настройка драйвера магнитофона (определение константы чтения в ячейке с адресом &O314) и чтение заголовка файла (его имени, адреса загрузки и длины);
- проверка возможности загрузки данного файла;
- загрузка файла с параллельной распечаткой информации о нем;
- чтение контрольной суммы файла и проверка правильности чтения.

Теперь обсудим вопросы практической реализации данного алгоритма. Основное требование, предъявляемое к подпрограмме загрузки — скорость работы. Иначе чтение будет крайне неустойчивым, особенно при большой плотности записи (медленно работающая программа загрузки попросту не успевает отслеживать сигнал с магнитофона). Поэтому устанавливать используемые векторы прерываний и ячейки памяти необходимо перед началом чтения, чтобы не тратить на это время потом. В соответствии со сказанным, начало процедуры чтения будет таким (ассемблер формата MICRO-11M):

```
START: MOV SP,@#310      ;запоминаем указатель стека
       MOV #READ,@#14    ;адрес программы обработки
                         ;прерывания по Т-разряду
       MOV #200,-(SP)    ;ССП с установленным Т-разрядом
       MOV #PRINT,-(SP)  ;адрес "основной программы"
       MOV #214,R0       ;двукратное "АР2"+"СБР" приводит
       EMT 16            ;экран в исходное положение
       EMT 16            ;(верхний левый угол = &O40000)
```

В качестве «основной программы» используется подпрограмма печати информации о файле, а для обработки прерывания по вектору 14 используется подпрограмма чтения очередного бита данных с магнитофона. Если теперь выполнится команда RTT, сработает одна команда из подпрограммы PRINT и произойдет прерывание по Т-разряду. Тогда снова выполнится расположенная по адресу READ подпрограмма чтения одного бита с магнитной ленты, в конце которой должна стоять команда RTT, после чего исполнится еще одна команда подпрограммы PRINT и т. д.

Настройку драйвера и чтение заголовка файла целесообразно осуществить с помощью подпрограммы ПЗУ, расположенной по адресу @#116640:

```
M1:    MOV #177716,R3 ;адрес системного регистра ввода-вывода
       CLR @#302      ;ненулевое содержимое ячейки @#302 -
                      ; признак фиктивного чтения, нулевое -
                      ; обычное чтение
       MOV #320,R1    ;адрес блока параметров
       MOV #5,@#344   ;считываем файл с несуществующим
                      ; именем (последние два символа имени -
                      ; управляющие)
       MOV R1,@#306   ;запоминаем адрес блока параметров
       CALL @#116640  ;вызов подпрограммы ПЗУ монитора
```


Подпрограмма по адресу @#116640, вообще говоря, входит в драйвер магнитофона EMT 36 и осуществляет чтение файла по заданному имени либо фиктивное чтение файла. В нашем примере она используется в режиме чтения файла по имени, но имя задано «несуществующее», с символами, коды которых равны 0 и 5. Таким образом, подпрограмма @#116640 настраивает драйвер магнитофона, считывает заголовок очередного файла в адреса &O346—&O371, обнаруживает несоответствие только что считанного имени с заданным нами «несуществующим» именем и возвращает управление нашей программе. Следует заметить, что скорость выполнения программы, зашитой в ПЗУ БК-0010(.01), несколько больше, чем у записанной в ОЗУ. Из этого следует, что настройку драйвера, считывание заголовка, тела и контрольной суммы файла желательно проводить, используя только ОЗУ или только ПЗУ. Иначе если, например, настройку драйвера (выработку константы чтения) выполнять подпрограммой ПЗУ, а тело файла загружать без использования ПЗУ, то вследствие различия скоростей исполнения команд программ в ОЗУ и ПЗУ надежность чтения файла резко уменьшится. Конечно, можно попробовать ввести коррекцию константы чтения, но это достаточно сложно.

Проверка возможности загрузки включает в себя главным образом контроль длины файла (чтобы не загружать файл, под который заведомо не хватит памяти):

```
MOV @#350,R2  ;длина текущего файла в R2
CMP R2,#40000 ;длина больше &O40000 (для примера) ?
BHI NOMEM     ;если да, то сообщить о нехватке памяти
```

В копировщиках типа BYCOP2, позволяющих загружать несколько файлов подряд без остановки магнитофона, необходимо предусмотреть сравнение имен текущего файла и загруженного перед ним, чтобы не загружать в память два совершенно одинаковых файла (на кассетах файлы часто пишутся по 2—3 копии). Это сравнение лучше всего выполнять с помощью все той же подпрограммы @#116640, указав перед обращением к ней в ячейке @#322 в качестве адреса загрузки файла адрес в ПЗУ (например &O110000), а вместо «несуществующего» имени, поместив в блок параметров имя предыдущего считанного файла. Теперь, если имя текущего файла не совпадает с именем предыдущего, произойдет требуемый нам возврат из подпрограммы @#116640, а при их совпадении будет сделана попытка считать текущий файл в ПЗУ, что вызовет прерывание по вектору 4. В подпрограмме же обработки прерывания по вектору 4 можно еще одним сравнением имен различить, произошло ли прерывание из-за нажатия клавиши «СТОП» на клавиатуре или при попытке записи в ПЗУ*.

* Другой, более простой и быстрый способ различения причины прерывания по вектору 4 показан в «Техническом описании БК-0010» (см. «Персональный компьютер БК-0010 — БК-0011М», № 3 за 1994 г., с. 20), а также в статье «Программное прерывания в БК-0010(.01)» в текущем выпуске журнала, и заключается в анализе бита 2 регистра @#177716. При нажатии клавиши «СТОП» или отработке команды HALT сразу же после прерывания в данном бите содержится «логическая единица», а в случае прерывания по зависанию (обращение к отсутствующему регистру, попытка записи в ПЗУ и т. п.) — «логический нуль». — *Прим. ред.*

Перед считыванием тела файла следует произвести необходимую подготовку:

```
MOV #76577,R1 ;адрес загрузки файла (низ экрана)
NEG @#304     ;читаем "снизу вверх"
```

Для визуализации процесса загрузки современные копировщики осуществляют ее «в экран». Этот способ имеет две разновидности: чтение «сверху вниз» и «снизу вверх».

Такая визуализация загрузки дает пользователю возможность, не дожидаясь окончания чтения файла, прервать его при заведомой ошибке. Ячейка @#304 указывает подпрограмме @#117336, как производить чтение («снизу вверх» — &O177777, «сверху вниз» — 1). Однако способ чтения файла в экран «сверху вниз» ограничивает длину загружаемого файла величиной &O40000 (это объем экранной памяти), поэтому загрузка «снизу вверх» предпочтительнее. Для многофайловых же копировщиков типа BYCOP2 оптимальным является чтение текущего файла «снизу вверх», пересылка его в наиболее младшие свободные адреса памяти, а затем чтение «снизу вверх» следующего файла.

Вернемся к нашей программе и предусмотрим в ней синхронизацию процесса чтения:

```
      MOV @#314,R0
      MOV R0,@R5      ;копия константы - в ячейке &040
M2:   CLR R4          ;обнуляем счетчик
      CALL @#117452   ;читаем импульс, длительность в R4
      CMP R4,R0       ;длительность импульса больше константы ?
      BLO M2          ;если нет, то еще раз
      ASL R0          ; удваиваем константу
      CMP R4,R0       ;длительность импульса больше
                      ; удвоенной константы ?
      BHI M1          ;если да, то ошибка
      CALL @#117376   ;считываем синхроимпульс
```

Для увеличения быстродействия копию константы чтения целесообразно поместить в ячейку &O40, что позволит при чтении тела файла обращаться к ней с помощью косвенной регистровой адресации (@R5), более быстрой, чем абсолютная или относительная. (Значение &O40 в R5 было занесено подпрограммой @#116640 при ее вызове.) Следует помнить, что ячейка &O40 используется драйвером дисплея EMT 16, поэтому после окончания считывания необходимо восстановить ее исходное содержимое (нуль).

Теперь часть программы, отвечающая за чтение тела файла и контрольной суммы, выглядит так:

```
M3:   MOV #10,R0      ;количество битов в байте
M4:   RTT             ;выполнить очередную команду п/пр PRINT
READ: CLR R4          ;обнуляем счетчик длительности
      CALL @#117406   ;длительность очередного импульса
      CMP @R5,R4      ;если длительность больше константы, то C=1
      RORB @R1        ;бит C - в старший бит текущего байта
      SOB R0,M4       ;цикл по текущему байту
```



```
      DEC R1          ;следующий байт
      SOB R2,M3       ;цикл по длине файла
KS:   MOV #20,R0      ;&020 бит в слове контрольной суммы
M5:   CLR R4          ;обнуляем счетчик длительности
      CALL @#117406   ;длительность очередного импульса
      CMP @R5,R4      ;если длительность больше константы, то C=1
      ROR R1          ;бит C в старший бит слова контрольной суммы
      SOB R0,M5       ;следующий бит
      MOV #220,@R3    ;останавливаем двигатель магнитофона
      MOV @#310,SP    ;восстанавливаем стек
      CLR @R5         ;восстанавливаем ячейку @#40
      MOV @#350,R4    ;длина в R4
      MOV #76600,R5   ;фактический адрес загрузки в R5
      CLR R0          ;далее - подсчитываем контрольную сумму файла
M6:   CLR R2
      BISB -(R5),R2
      ADD R2,R0
      ADC R0          ;контрольная сумма накапливается в R0
      SOB R4,M6
      CMP R0,R1       ;сравниваем подсчитанную контрольную сумму
                      ;с принятой
      BEQ OKEY        ;если они равны - ошибок нет
NOMEM: CALL @#100644  ;выдаем сообщение "ОШИБКА"
OKEY: HALT            ;останов
```

В самом начале программы в стек последовательно были записаны слово состояния процессора с установленным Т-разрядом и адрес подпрограммы PRINT. Теперь при выполнении команды RTT управление будет передано на подпрограмму PRINT с установкой Т-разряда. После выполнения одной команды подпрограммы PRINT происходит прерывание по вектору 14 и управление передается на метку READ. После считывания очередного бита данных с магнитофона снова выполняется команда RTT, срабатывает следующая команда подпрограммы PRINT, затем читается следующий бит данных и т. д. Выход из процесса псевдопараллельного выполнения этих программ возможен в двух случаях: раньше завершит свою работу блок чтения тела файла или быстрее будет выполнена подпрограмма PRINT. Рассмотрим первый случай. Когда тело файла прочитано, а подпрограмма PRINT еще не выполнена, управление передается на метку KS (после этого команды подпрограммы PRINT выполняться не будут), осуществляется прием контрольной суммы с магнитной ленты и останов двигателя магнитофона. Указатель стека и ячейка @#40 приводятся в исходное состояние, подсчитывается контрольная сумма загруженного файла и сверяется с принятой. Если обе контрольные суммы совпадают, осуществляется выход из программы в монитор (команда HALT), иначе перед выходом выдается сообщение об ошибке (используется подпрограмма ПЗУ по адресу @#100644).

А вот текст подпрограмы PRINT:

```
PRINT: MOV #76600,-(SP)  ;адрес начала закраски
M7:    MOV @SP,PODST     ;"подставляем" текущий адрес
       MOVB #252,-(SP)   ;код закраски
       .#112637          ;код команды MOVB (SP)+,@#0
PODST: .#0
       INC @SP           ;следующий байт
```

```
BPL M7              ;если адрес меньше 0 (больше &077777),
TST (SP)+           ; то восстанавливаем стек
MOV #117342,-(SP)   ;передача управления на ПЗУ,
                    ; там есть RTS PC
MOV (SP)+,@#14
BR  KS              ;считываем контрольную сумму
```

Ради краткости листинга наша подпрограмма не печатает информацию о файле, а только закрашивает нижнюю часть экрана с адреса &O76600 (читатели же могут сами реализоать по аналогии с приведенным выше листингом любой сервис). На подпрограмму PRINT накладываются довольно жесткие ограничения. Во-первых, в ней нельзя изменять содержимое регистров R0—R5, так как подпрограмма выполняется псевдопараллельно с подпрограммой считывания тела файла, которая использует регистры R0—R5 для своих нужд. Во-вторых, время выполнения любой из команд подпрограммы PRINT должно быть как можно меньше, во всяком случае, не более 18.8 мкс. «Медленные» команды необходимо разбивать на две — три более быстрых, действуя через стек. (Для подсчета длительности выполнения команд БК-0010(.01) можно воспользоваться сведениями из [3].)

Теперь расскажем о назначении последних трех команд подпрограммы PRINT. В большинстве реальных случаев подпрограмма PRINT завершает работу намного быстрее, чем будет считано тело файла (для нас это второй случай), и целесообразно повысить надежность дальнейшего считывания, передав управление на подпрограмму ПЗУ по адресу @#117342 (которая дозагрузит остаток файла). После выполнения предпоследней команды подпрограммы PRINT произойдет прерывание по вектору 14, но управление будет передано уже не на метку READ, а по адресу @#117342. В стек при этом будут помещены слово состояния процессора и адрес последней команды подпрограммы PRINT. При возврате же из подпрограммы @#117342 управление будет передано на последнюю команду подпрограммы PRINT, которая, в свою очередь, передаст управление на блок, отвечающий за прием контрольной суммы.

Кроме процедуры чтения первого найденного файла, с помощью данного приема можно реализовать операцию псевдопараллельной выдачи информации о файле при проверке первого найденного файла без его загрузки в память («Verify»). Таким способом можно проверять файлы любой длины независимо от того, поместятся ли они в памяти БК. Принцип работы процедуры проверки следующий: вместо считывания тела файла осуществляется подсчет его контрольной суммы в ячейке @#0, после чего она сравнивается с принятой с магнитной ленты (в регистре R1). Для реализации такого алгоритма внесем в приведенную выше процедуру чтения первого найденного файла некоторые изменения:

- вместо BHI NOMEM поставим команду NOP (отключение контроля длины);
- вместо команды NEG @#304 запишем последовательно команды CLR R1 и CLR @R1 (очистка буфера контрольной суммы);
- вместо RORB @R1 запишем RORB R1;
- вместо DEC R1 запишем последовательно команды ADD R1,@R0 и ADC @R0 (накопление контрольной суммы);

- вместо CMP R0,R1 запишем CMP @R4,R1;
- в подпрограмме PRINT вместо последних трех команд запишем 1: BR 1 (зацикливание).

В заключение дадим несколько общих замечаний, касающихся и проверки, и чтения первого найденного файла. Если ваш магнитофон инвертирует сигнал или считываемая кассета была записана на «инвертирующем» магнитофоне (что легко определить, считав любую программу в БЕЙСИК или любой отладчик и просмотрев затем содержимое байта &O300: ненулевое значение указывает на инвертирование сигнала), замените в процедурах все адреса 117406 на 117434, а 117452 на 117424. (В реальном копировщике эта замена выполняется автоматически в зависимости от содержимого байта &O300. Часть программы, отвечающая за замену адресов, вставляется в листинг перед синхронизацией процесса чтения.) Недостатком предложенных алгоритмов процедур является некоторое снижение надежности чтения (или проверки) при большой плотности записи на магнитной ленте. Поэтому в реальных копировщиках в блок проверки на возможность загрузки целесообразно включить проверку плотности записи: если константа чтения меньше, чем &O20, то чтение (проверку) следует выполнять стандартно, не используя прерывание по вектору 14. (Этого можно достичь, поместив по адресу команды RTT значение &O240 — код команды NOP.)

Для практической проверки описанных выше приемов был разработан многофайловый магнитофонный копировщик TAPE1, который автор данной статьи рекомендует «вскрыть» в отладчике для более подробного изучения структуры подпрограммы PRINT и других нюансов работы с драйвером магнитофона, не отмеченных выше.

ЛИТЕРАТУРА*

1. *Зальцман Ю.* Архитектура и ассемблер БК // Информатика и образование. 1991. № 5.

2. *Булитко В.* Псевдопараллельное исполнение программ на БК-0010 // Информатика и образование. 1992. № 2.

3. *Зальцман Ю.* Продолжительность исполнения команд на БК-0010 // Информатика и образование. 1991. № 6.

* Статья Ю. А. Зальцмана «МикроЭВМ БК-0010. Архитектура и программирование на языке ассемблера», расширенная и дополненная автором (по сравнению с [1] и [3]), включая сведения о длительностях выполнения команд ассемблера, опубликована в журнале «Персональный компьютер БК-0010 — БК-0011М», № 1 за 1994 г. — № 3 за 1995 г. — *Прим. ред.*

Одной из наиболее интересных и многообещающих алгоритмических возможностей среди предоставляемых компьютером БК программисту на ассемблере является механизм программных прерываний. Но эта возможность часто не используется из-за незнания принципов работы прерываний, боязни «нарушить что-то» в мониторе БК. В данной статье приводится ряд простейших рекомендаций для начинающих программистов.

Д. Ю. Усенков,

Москва

Программные прерывания в БК-0010(.01)

Вначале поговорим о том, что такое прерывание вообще. Прерывание — это ситуация, когда процессор, получив от некоторого устройства (называемого источником прерывания) сигнал (запрос на прерывание), приостанавливает выполнение текущей задачи и переходит к исполнению специального программного модуля, называемого обработчиком или драйвером прерывания. По окончании обработки следует возврат на продолжение выполнения основной программы.

Как видим, прерывание очень похоже на работу с подпрограммой. Но есть и существенное отличие: «настоящее» (аппаратное) прерывание возникает всегда «неожиданно» для процессора («по воле» устройства, пославшего запрос), тогда как переход к подпрограмме совершается при исполнении самим процессором специальной команды в основной программе (т. е. «с точки зрения» процессора это действие полностью «детерминированное», в отличие от «случайного» прерывания).

Однако кроме прерываний от различных устройств (аппаратных) в БК-0010 есть и так называемые программные прерывания, являющиеся своего рода «помесью» — для вызова, как и при работе с подпрограммой, служит специальная команда в основной программе, а механизм вызова и возврата тот же, что и у «настоящих» прерываний. Какие же основные преимущества обеспечивают программные прерывания по сравнению с подпрограммами? Их три:

- **Унификация.** В ПЭВМ для обслуживания внешних устройств используются специальные программы — драйверы, на которые в конечном счете и передается управление при отработке прерываний от данного устройства. Реализацию некоторых «чисто программных» действий (например, чтение символа с клавиатуры или вывод строки на экран) также можно оформить в виде драйверов, тогда последние удобно вызывать по механизму прерывания, как и драйверы устройств.
- **Сохранение состояния.** Вызов драйвера по механизму прерывания предусматривает автоматическое сохранение в стеке и восстановление при возврате значения ССП (слова состояния процессора), тогда как при вызове подпрограмм его сохранение не производится.

- **Экономия памяти.** Команда вызова подпрограммы типа JSR Rn, <адрес> в большинстве случаев (когда адрес вызова задан не через регистр) занимает четыре байта, а для команды прерывания требуется только два. Казалось бы, экономия невелика, но в БК-0010 с его ставшем притчей во языцех малым объемом ОЗУ каждый байт на счету, а при большом количестве вызовов «копеечная» экономия может дать солидный выигрыш в занимаемой памяти.

Учитывая, что пользователям БК чаще всего приходится сталкиваться именно с программными прерываниями (так как драйверы устройств изменяют крайне редко), изучим их более подробно. Это прерывания IOT, EMT, TRAP, а также, с некоторой оговоркой, HALT. К этому же списку можно отнести прерывания по «резервной машинной команде» (т. е. когда код, который пытается выполнить процессор, никакой команде ассемблера не соответствует) и по Т-биту. Последние два случая рассмотрим лишь кратко, так как они используются достаточно редко.

Прерывание по «резервной команде» изначально предназначалось для обнаружения ошибок в программе (скажем, если программиста «черт попутает» передать управление вместо подпрограммы на список данных для нее) и защищает компьютер от зависания. Но этот же механизм можно использовать и для «добавления» в набор ассемблерных операторов своих команд. По такому способу, например, реализованы отсутствующие в стандартном ассемблере БК команды плавающей арифметики в трансляторе языка ФОКАЛ. «Натыкается» процессор на незнакомый код, отрабатывает прерывание, а там его уже «ждет» драйвер, который распознает, на каком коде процессор «споткнулся», является ли это кодом команды плавающей арифметики и каким, и передает управление на реализацию данной операции. (Кстати, коды 10—17 вызывают прерывание не по «резервной команде», а аналогично HALT по вектору 4.)

Прерывание же по Т-биту происходит после исполнения процессором каждой очередной команды программы, если в ССП установлен в единицу так называемый бит Т (Trace). Основное назначение данного прерывания — организация пошаговой отладки, однако умельцы-БКманы нашли способ реализовать с его помощью такую «экзотику», как многопрограммный режим. Подробнее об этом можно прочитать в журнале «Информатика и образование», № 2 за 1992 г., а также в статье С. А. Мальцева об использовании прерывания по Т-разряду в копировщиках.

Однако «вернемся к нашим баранам», то бишь к прерываниям. Начнем с их механизма (он одинаков и в программных, и в аппаратных прерываниях). Встретив команду вызова, например IOT, процессор автоматически заносит в стек значения ССП и счетчика команд (т. е. адрес оператора, следующего после команды вызова прерывания). Затем из специально отведенной для данного прерывания пары машинных слов, называемой вектором прерывания, извлекается и переписывается в счетчик команд значение адреса входа в драйвер обработки прерывания (первое слово вектора) и в ССП — новое значение состояния процессора (второе слово). (Стандартные значения векторов прерывания копируются в ОЗУ монитором БК из прошитой в ПЗУ таблицы сразу же после включения питания; чтобы «переадресовать» то или иное прерывание на другой обработчик, нужно в начале основной программы записать командой MOV адрес входа в него в соответствующий вектор.)

Эти действия можно условно записать в виде ассемблерных команд:

```
MOV PSW, -(SP)          ; PSW - значение ССП
MOV PC, -(SP)
MOV @#<вектор>, PC
MOV @#(<вектор>+2), PSW
```

Заметим, что после входа в обработчик адрес оператора, следующего после команды вызова прерывания (т. е. адрес возврата), хранится в текущей ячейке стека, на которую указывает содержимое SP. Значение же, занесенное в стек до входа в прерывание последним, имеет теперь адрес 4(SP), предпоследнее — 6(SP) и т. д. Это обстоятельство может потребоваться при организации передачи параметров драйверу через стек.

Для возврата из прерывания служит «универсальная» (используемая независимо от причины прерывания) команда RTI. При ее исполнении процессор восстанавливает из стека значения PC и ССП, таким образом возвращаясь к основной программе (если содержимое стека не было изменено):

```
MOV (SP)+, PC
MOV (SP)+, PSW
```

Последнее обстоятельство требует особого внимания при использовании стека в драйвере прерывания. Непосредственно перед отработкой команды RTI значение SP должно быть тем же, что и сразу после входа, в противном случае вместо возврата возможны зависание и прочие неприятности.

Более искушенный пользователь БК может возразить, что для возврата из прерывания существует кроме RTI еще одна команда — RTT. Да, механизм ее действия тот же, но при восстановлении из стека значения ССП она дополнительно устанавливает в единицу бит T. Поэтому команда RTT является специальной, применяется только при работе с отладочным прерыванием (по T-разряду) и для возврата из всех прочих прерываний, как правило, не используется.

Для справки приведем список адресов векторов для всех прерываний:

Адрес вектора	Источник (причина прерывания)
4	Зависание, HALT или клавиша «СТОП»
10	Ошибочный код («резервная команда»)
14	Прерывание по T-разряду
20	IOT
24	Авария сетевого питания
30	EMT
34	TRAP
60	Клавиатура
100	IRQ2 (от устройства, подключенного к порту УВВ)
274	Клавиатура в регистре АР2

Прерывания по EMT и TRAP

Прерывания по командам EMT и TRAP используются в БК чаще всего: EMT — для вызова стандартных драйверов, прошитых в ПЗУ монитора, а TRAP отдана в распоряжение программистам (нужно, однако, учесть, что в ФОКАЛе и БЕЙСИКе TRAP-прерывания уже использованы программистами, писавшими трансляторы с этих языков). Отличительная особенность прерываний по EMT и TRAP — их «многоканальность»: младший байт команды содержит номер от 0 до 377 и может использоваться для выбора конкретной функции из предоставляемого по данному прерыванию набора. Например, EMT 6 — чтение символа с клавиатуры, EMT 20 — вывод строки на экран и т. д. Но независимо от номера вектор прерывания по данной команде один и тот же, следовательно, драйвер прерывания должен вначале извлечь каким-либо способом этот номер, а затем определить по нему и вызвать на выполнение требуемую подпрограмму. Вот как это делает стандартный драйвер EMT, прошитый в ПЗУ монитора БК с адреса 100112 (эта часть драйвера называется диспетчером EMT):

```
MOV R5, -(SP)          ; сохранить регистр в стеке
MOV 2(SP), R5          ; в R5 заносится адрес возврата
MOV -(R5), R5          ; в R5 - код команды, вызвавшей прерывание
BIC #177400, R5        ; выделить младший бит (номер)
MOV 100000(R5), R5     ; из списка извлечь адрес подпрограммы
                       ;  реализации данной функции
JSR PC, (R5)           ; вызов подпрограммы
MOV (SP)+, R5          ; восстановить регистр
RTI                    ; возврат из прерывания
```

Обратим внимание на число 100000. Это константа в так называемой косвенной адресации с индексацией. Предполагается, что с адреса 100000 начинается список адресов подпрограмм в порядке возрастания номеров: для EMT 0 адрес хранится в ячейке 100000, для EMT 2 — в ячейке 100002, EMT 4 — 100004 и т. д. Однако в стандартном EMT-драйвере используются не все номера EMT (только с 4 по 36 включительно, плюс 40—50 для неиспользуемого в индивидуальных БК-0010(.01) последовательного интерфейса ИРПС и 52—100 — резервные), так что оставшаяся область списка (адреса 100000—100002 и свыше 100112) используется в мониторе для других целей (два первых машинных слова — команда JMP на точку входа в монитор при включении питания БК и перезапуске по кнопке RESET или с помощью переключателя СТОП-ПУСК, а адрес 100112, как уже было сказано выше, — начало драйвера EMT). Кроме того, при данном алгоритме диспетчера номера EMT могут быть только четными. В «самодельном» драйвере можно снять это ограничение, добавив после строки BIC #177400, R5 команду ASL R5 (умножение номера на 2).

Тогда список адресов подпрограмм будет выглядеть так:

```
<адрес списка>:    <п/пр0>     ; EMT 0
<адрес списка>+2:  <п/пр1>     ; EMT 1
<адрес списка>+4:  <п/пр2>     ; EMT 2

                     . . .
                   <п/пр 377>  ; EMT 377
```

Учитывая, что EMT служит для вызова большого числа часто используемых функций, как правило, целью написания «самодельного» драйвера является замена не всех EMT-функций, а только одной или двух (например, так реализуется переадресация к диску по EMT 36 в дисковых ОС для БК). В этом случае драйвер перехвата должен содержать команду CMP (начальный адрес драйвера ALTEMT должен быть предварительно, в начале основной программы, занесен в ячейку вектора прерывания командой MOV #ALTEMT, @#30):

```
ALTEMT: MOV R5, -(SP)
        MOV 2(SP), R5
        MOV -(R5), R5
        CMP R5, #104036     ; сравнить код вызвавшей команды
                            ;  EMT с требуемым
        BEQ NEMT36          ; да - на новый драйвер EMT36
        JMP @#100122        ; нет - на продолжение обработки
                            ;  в стандартном драйвере
NEMT36:   . . .             ; новый драйвер EMT36
```

Аналогичным способом можно исключить одну или несколько функций. Вот как, например, делается «отмена» EMT 16 (и работающей «на ее основе» EMT 20) для запрета вывода сообщений об ошибках работы с диском при записи на него копий экранного ОЗУ:

```
ALTEMT: MOV R5, -(SP)
        MOV 2(SP), R5
        MOV -(R5), R5
        CMP R5, #104016
        BEQ NOEMT
        JMP @#100122
NOEMT:  MOV (SP)+, R5       ; сразу выход из EMT
        RTI
```

И наконец, несколько «искусственный», но весьма полезный прием — «временная» подмена какой-либо функции, например для замены пробелами непечатаемых символов:

```
ALTEMT: MOV R5, -(SP)
        MOV 2(SP), R5
        MOV -(R5), R5
        CMP R5, #104016
        BEQ E16
        JMP @#100122
E16:    . . . . . .         ; проверка на "печатаемость"
        BEQ 1               ; переход, если код печатаемый
        MOV #40, R0         ; замена непечатаемого кода пробелом
1:      MOV #100112, @#30   ; временно установить стандартный
                            ;  EMT-драйвер
        EMT 16              ; стандартный вывод
        MOV #ALTEMT, @#30   ; вернуться к "своему" драйверу
        MOV (SP)+, R5
        RTI
```


Таким способом можно осуществлять и переключение между несколькими драйверами, реализуя неограниченное количество различных EMT-функций.

Можно использовать менее «экзотический» вариант, если вспомнить, что адрес подпрограммы, реализующей, например, EMT 16, хранится в ПЗУ монитора в ячейке @#100016 (100000 плюс номер EMT-функции):

```
1:      MOV @#100016, R5
        JSR PC, (R5)
        MOV (SP)+, R5
        RTI
```

Обратим внимание и на такой факт: для использования «самодельного» драйвера EMT с подменой одной из функций необходимо, чтобы адрес входа в драйвер был записан в ячейке @#30. Но, «отпуская» на стандартное выполнение остальные EMT-функции, надо помнить, что EMT 14 и EMT 4 перезаписывают все или некоторые (EMT 4 — только клавиатурные) векторы прерываний стандартными значениями и таким образом «сбивают» наш драйвер. Если речь идет о «самостоятельной» программе, в которой вы хотите использовать свой драйвер, достаточно не забывать сразу же после EMT 14 (или EMT 4) заново переписывать нужные векторы. А что делать, если требуется встроить драйвер в готовую программу или, скажем, написать резидентную утилиту для вывода на принтер картинок с экрана? Как многие уже, наверное, догадались, требуется опять-таки подмена EMT-функций, на этот раз EMT 14 и EMT 4 (это делается аналогично уже рассмотренной замене EMT 16). Еще проще заменить весь драйвер:

```
ALTEMT: MOV R5, -(SP)
        MOV 2(SP), R5
        MOV -(R5), R5
          . . .             ; проверка и подмена каких-либо
          . . .             ;  других EMT-функций
ALLEMT: MOV 2(SP), R5     ;*; для отработки прочих функций
        MOV -(R5), R5     ;*;  заново извлекаем номер
        BIC #177400, R5
        MOV 100000(R5), R5
        JSR PC, (R5)
        MOV #ALTEMT, @#30   ; всегда обновляем вектор EMT
        MOV (SP)+, R5
        RTI
```

К вышеприведенной программе следует сделать два замечания. Во-первых, строки, помеченные звездочками, нужны только в том случае, если при проверке и подмене других EMT-функций (например, тех же EMT 16 или EMT 36) значение R5 было изменено. Если же при входе на метку ALLEMT R5 по-прежнему содержит код вызвавшей прерывание команды EMT <номер>, обе эти строки не нужны и рассматриваемый фрагмент листинга будет таким:

```
          . . .
ALLEMT: BIC #177400, R5
        MOV 100000(R5), R5
          . . .
```

Во-вторых, в данном листинге показано обновление только вектора EMT-прерывания (@#30). Но аналогичным образом можно выполнить перезапись и других векторов или системных ячеек.

Аналогично EMT организуется диспетчер и для TRAP-прерывания (так как TRAP полностью отдано «на откуп» пользователям БК, как правило, нужно писать «полный» драйвер):

```
ALTRAP:             MOV R5, -(SP)
                    MOV 2(SP), R5
                    MOV -(R5), R5
                    BIC #177400, R5
                    MOV <адрес списка>(R5), R5
                    JSR PC, (R5)
                    MOV (SP)+, R5
                    RTI
<адрес списка>:     .#<n/пр0>
                    .#<n/пр2>
                      . . .
                    .#<n/пр376>
```

Но иногда требуется упрощенный вариант, когда с помощью TRAP вызывается только одна функция, а номер используется как ее аргумент. Например, генерация звука, частота которого задается в качестве номера TRAP:

```
ALTRAP: MOV R5, -(SP)
        MOV 2(SP), R5
        MOV -(R5), R5
        BIC #177400, R5       ; R5 - частота звука
        JSR R4, @#110346      ; сохранить R0-R3
        MOV #<длит>, R2       ; R2 - длительность
1:      MOV R5, R0            ; ГЕНЕРАЦИЯ ЗВУКА
        BIS #100, @#177716
2:      SOB R0, 2
        MOV R5, R0
        BIC #100, @#177716
3:      SOB R0, 3
        SOB R2, 1
        JSR R4, @#110362      ; восстановить регистры
        MOV (SP)+, R5
        RTI
```

Заканчивая разговор о EMT и TRAP, отметим следующее. Ранее упоминалось, что не все номера EMT-функций задействованы в стандартном драйвере. Однако некоторые из неиспользуемых EMT производят обращение к тем или иным адресам ОЗУ пользователя, что дает возможность дополнять стандартный набор своими функциями без разработки «самодельного» драйвера. Перечислим некоторые из них (подробности см. в журнале «Вычислительная техника и ее применение», № 11 за 1991 г., с. 43):

№ EMT	Адрес входа	№ EMT	Адрес входа
112	10546	220	5166
114	16604	230	4766
120	14504	232	4466
122	42704	240	5036
126	16504	260	12706
132	4714	262	1000
134	12604	270	4736
140	5000	314	10004
142	12702	326	20426
146	12720	332	1002
154	12736	364	1006
202	5020	366	5704
210	5266	372	16704

Прерывание по IOT

Прерывание по вектору IOT, обычно именуемое ловушкой, используется несколько реже, чем EMT или TRAP. Работать же с IOT-прерыванием проще, так как это не требует организации диспетчера (номеров-то у IOT нет!):

```
        ; адрес ALTIOT предварительно нужно занести в вектор @#20
ALTIOT: JSR R4, @#110346      ; сохранить регистры
          . . .               ; обработка прерывания
        JSR R4, @#110362      ; восстановить регистры
        RTI                   ; выход
```

Но может быть и такой случай: в какой-либо сложной программе вам не хватило количества доступных EMT- и TRAP-функций и нужно создать «на базе» IOT «многоканальное» прерывание, такое же, как в TRAP. Можно сделать и это, но тогда номер функции придется передавать отдельно, например в следующем слове после IOT:

```
        IOT
        .@<номер>
        . . .
```

Диспетчер обработки в этом случае будет таким:

```
ALTIOT: MOV R5, -(SP)          ; пока все
        MOV 2(SP), R5          ;  похоже на EMT...
        MOV (R5), R5           ; а здесь (R5) без минуса! это не
                               ;  код IOT, а следующий за ним номер
        ADD #2, 2(SP)          ; коррекция адреса возврата в стеке
        BIC #177400, R5        ; выделяем номер
        MOV <список>(R5), R5   ; извлекаем адрес подпрограммы
        JSR PC, (R5)           ; дальше -
        MOV (SP)+, R5          ;  опять как в
        RTI                    ;  стандартном EMT-драйвере
```

Смысл коррекции адреса возврата состоит в том, чтобы «сдвинуть» указатель следующей команды, хранящийся в стеке, на код, следующий после номера.

HALT — «почти-прерывание»

Последний еще не рассмотренный нами вариант — прерывание по команде HALT. Хотя во всех справочниках эта команда носит название «останов работы процессора», в БК механизм ее действия другой. При выполнении команды HALT (или при нажатии клавиши «СТОП») процессор волею его разработчиков пытается обратиться к не существующему в БК системному регистру (по адресу @#177676), происходит зависание и, соответственно, прерывание по вектору @#4 (в данном случае значение термина «зависание» несколько иное, нежели обычно имеющееся в виду отсутствие реакции компьютера на внешние воздействия; подробнее об этом будет сказано чуть позже).

В БК прерывание по команде HALT используется обычно как «безвозвратное» (в том смысле, что после него следует не возврат к выполнению прерванной программы, а, например, выход в монитор). Так, в мониторе по HALT или при нажатии клавиши «СТОП» фактически производится его перезапуск (минуя лишь кое-какие начальные установки):

```
100442: MOV #1000, SP       ; восстановить указатель стека
        MOV #220, @#177716  ; регистр - в исходное состояние
        EMT 4               ; обновить векторы клавиатуры
        BR 100300           ; выход в начало монитора
```

Однако, несмотря на подобное «пренебрежение» со стороны разработчиков монитора, HALT является самой «настоящей» командой прерывания! Убедиться в этом очень просто: достаточно завершить обработку прерывания по вектору @#4 командой RTI:

```
START: EMT 14
       MOV #STOP, @#4     ; или NOP
REST:  MOV KOD, R0
       MOV #1000, R1
1:     EMT 16
       SOB R1, 1
       HALT
       INC KOD
       BR REST
KOD:   .#100              ; код символа "@"
STOP:  MOV #10, R1
       MOV #7, R0
2:     EMT 16
       SOB R1, 2
       JMP REST           ; или RTI
```

Эта программа выполняет простейшее действие: выводит на экран восемь строк одинаковых символов, код которых задан в ячейке KOD, после чего следует команда HALT. Если во второй строке стоит команда NOP, после вывода восьми строчек, как обычно, происходит выход в монитор.

Запишем во второй строке оператор MOV, переписывающий вектор @#4 и таким образом заменяющий стандартный обработчик HALT на «самодельный», начальный адрес которого соответствует метке STOP (как это и сделано в листинге). В этом обработчике предусматривается выдача серии из восьми «щелчков» и безусловный переход на начало основной программы — это вариант «обычного», «безвозвратного» прерывания. (Вообще говоря, перед JMP REST следовало бы добавить строку ADD #4,SP, чтобы выполнить

коррекцию указателя стека, иначе после каждой отработки прерывания он сдвигается на 4 байта в сторону уменьшения адресов.) Легко видеть, что в этом случае результат работы программы будет таким: восемь строк из знаков «@», «треск», опять восемь строк из «@», снова «треск» и т. д.

Заменим теперь в последней строке JMP REST на RTI. Как теперь изменится работа программы? Если бы мы имели дело, например, с прерыванием по IOT, можно было бы предположить, что после возврата из прерывания управление на метку REST будет передано только после увеличения кода символа на единицу.

Тогда на экране мы должны увидеть (и услышать) следующее: восемь строк «@», «треск», восемь строк «A» (код уже на единицу больше!), «треск», восемь строк «B», «треск» и т. д. Запустив программу на выполнение, убеждаемся, что результаты ее работы полностью совпадают с «предсказанными», т. е. HALT действительно работает как «классическая» команда программного прерывания.

Здесь следует обратить внимание на одно важное обстоятельство. Указанным выше способом не рекомендуется «укрощать» клавишу «СТОП» — иначе могут возникать различные неприятности, вплоть до зависания компьютера.

Чтобы в своей программе блокировать действие клавиши «СТОП», можно воспользоваться следующим обработчиком прерывания по вектору @#4:

```
STOP:  MFPS @#PSW          ; сохранить старое значение ССП
       MOV (SP)+, @#HLT    ; извлечь адрес следующей команды
       TST (SP)+           ; удалить из стека сохраненное
                           ;  значение ССП
       SUB #2, @#HLT       ; коррекция адреса возврата
       NOP                 ; сюда можно вставить еще что-то
       MTPS @#PSW          ; восстановить ССП
       JMP @HLT            ; на продолжение работы
; внимание! В последней строке относительная косвенная адресация
PSW:   .#0                 ; буферные ячейки используем, чтобы
HLT:   .#0                 ; не занимать (и не сохранять) регистры
```

В данном варианте действие клавиши «СТОП» полностью блокировано (нажатие на нее игнорируется), но при желании можно вставить вместо NOP программный блок (или вызов подпрограммы), обеспечивающий индикацию нажатия на «СТОП», например звуковой сигнал, прежде чем продолжить исполнение основной программы. В этом случае нужно позаботиться о сохранении используемых регистров.

А для чего нужна коррекция адреса возврата? Когда мы нажимаем клавишу «СТОП», процессор «бросает все» (недообработав текущую команду) и «спешит прерваться», а в стек при этом попадает адрес следующего машинного слова. Если же мы хотим продолжить исполнение программы, вернуться надо именно на «недовыполненную» перед этим «текущую» команду, для чего адрес возврата нужно уменьшить на два. Кстати, именно поэтому данный алгоритм нельзя использовать для возврата из прерывания по команде HALT: возврат все время производится на ту же самую строку с HALT, вызывая «прочное» зависание, выйти из которого можно только после выключения питания или по кнопке RESET (если она есть). Поэтому перед

отработкой HALT нужно восстанавливать стандартное значение вектора: MOV #100442, @#4. Нельзя также нажимать клавишу «СТОП» (при обработке прерывания по данному алгоритму) во время ожидания компьютером ввода символа с клавиатуры по EMT 6 или EMT 10.

В заключение разберем такой интересный вопрос. В таблице адресов векторов прерываний мы отметили, что один и тот же вектор @#4 обслуживает помимо «СТОП» и HALT также и прерывания по зависанию. (Для справки: термин «зависание» в данном случае означает, что после обращения процессора к устройству с заданным адресом (в качестве устройств здесь выступают микросхемы ОЗУ и ПЗУ, а также системные регистры) сигнал ответа устройства (RPLY) в процессор не поступает — это признак неисправности устройства или просто его отсутствия (либо означает ошибочную попытку обратиться по несуществующему адресу). Если сигнала RPLY нет, процессор на микропрограммном уровне отсчитывает 64 такта синхрогенератора, после чего отрабатывает прерывание по вектору @#4.) Что же, наша программа не сможет точно распознать причину прерывания: зависание ли это, или HALT (либо «СТОП»)? Тогда как же это делает отладочный монитор ТС из блока МСТД, где сообщение «ЗВ <адрес>» выдается при зависании, но не при нажатии на «СТОП»? Оказывается, разработчики БК «лукаво утаили» от пользователей следующий факт: второй (считая, как это принято, справа налево начиная с нуля) бит регистра @#177716 содержит признак источника прерывания (обо всем этом говорится только во внутризаводской технической документации). Если во втором бите единица, это HALT или «СТОП», если же в нем нуль — зависание. Так что следующая несложная процедура (фрагмент ассемблерного дампа прошивки ПЗУ МСТД в БК-0010.01) отличает зависание от останова достаточно надежно, что будет удобно использовать во многих программах:

```
168722: ......          ; начало обработчика по вектору 4
        BIT #4,@#177716 ; анализ второго бита
        BNE HLT         ; не нуль - HALT или "СТОП"
        ......          ; нуль - зависание
```

Приложение 1. Пример построения EMT-диспетчера

Пусть в некоторой резидентной программе нужно обеспечить следующие характеристики:

- блокирование вывода символов по EMT 16;
- в зависимости от значения в ячейке DEVISE — адресацию по EMT 36: к диску через свой драйвер дисковода, если не нуль, или к магнитофону посредством обращения к стандартному драйверу — при нулевом значении DEVISE;
- в программе предполагается использование «самодельных» драйверов TRAP, «СТОП» и от клавиатуры (т. е. требуется обновление соответствующих векторов).

Для решения поставленной задачи диспетчер EMT-прерываний будет выглядеть так (не забудьте, что адрес входа в него (адрес метки ALTEMT) должен быть предварительно записан в ячейку @#30):

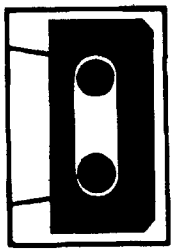

```
ALTEMT: MOV R5, -(SP)          ;
        MOV 2(SP), R5          ;
        MOV -(R5), R5          ;
        CMP R5, #104016        ; если EMT 16,
        BEQ QUIT               ;  то сразу на выход
        CMP R5, #104036        ; если EMT 36
        BNE 1                  ;
        TST DEVISE             ;  и в ячейке DEVISE не нуль,
        BNE NEMT36             ;  то переход к драйверу диска
1:      BIC #177400, R5        ; прочие EMTы и EMT 36 при
        MOV 100000(R5), R5     ;  DEVISE = 0
        JSR PC, (R5)           ;
        MOV #ALTEMT, @#30      ; восстановить векторы
        MOV #ALTRAP, @#34      ;  прерываний по EMT, TRAP,
        MOV #ALTSTP, @#4       ;  "СТОП", а также
        MOV #KLAW, @#60        ;  от клавиатуры - обычный
        MOV #NRKLAW, @#274     ;  и в регистре AP2
QUIT:   MOV (SP)+, R5          ; выход
        RTI                    ;
                               ;
NEMT36: JSR R4, @#110346       ; сохранить регистры R0-R3
        MOV R4, -(SP)          ;  и R4
                . . .          ; операции с диском
        MOV (SP)+, R4          ; восстановить регистры R4
        JSR R4, @#110362       ;  и R0-R3
        JMP QUIT               ; на выход из прерывания
;-------------------------------------------------------
```

Приложение 2. Некоторые особенности использования прерываний в ОС ANDOS v3.1

Для удобства программиста в ОС ANDOS версии 3.1 добавлен ряд возможностей, в том числе касающихся использования прерываний. Все они достаточно подробно изложены в технической документации, входящей в комплект ОС, но заслуживают того, чтобы лишний раз обратить на них внимание читателей.

1. Подпрограмма с адресом @#120002 производит вызов «дискового» эмулятора EMT 36, даже если содержимое вектора @#30 запорчено или соответствует «магнитофонному» варианту конфигурации. Таким образом, при написании программы, работающей одновременно с магнитофоном и дисководом (например, копировщика файлов), можно не заниматься постоянным переписыванием значения вектора @#30, а, занеся в него адрес #100112 (работа с прошитым в ПЗУ стандартным драйвером EMT-прерываний), обращаться к диску при помощи команды JSR PC,@#120002 . Единственный недостаток такого метода — программа будет работоспособна только в ОС ANDOS v3.1.

2. В рассматриваемой третьей версии ANDOS предусмотрена возможность автоматической записи после срабатывания EMT 14 требуемых значений для трех произвольно выбранных пользователем ячеек памяти. Адреса последних нужно занести в двухбайтные ячейки @#120244, @#120250 и @#120254, а переписываемые в них значения — в ячейки @#120246, @#120252 и @#120256 соответственно.



Ставшие «притчей во языцех» проблемы при работе с магнитофоном знакомы каждому пользователю БК, и программирующие на БЕЙСИКе — не исключение. Несмотря на широкое распространение дисководов и появление дисковых версий БЕЙСИКа магнитофон для БК еще долго не станет вчерашним днем (причина тому банальна — дисковод стоит дорого, а магнитофон уже есть почти в каждом доме.
Надеемся, что предлагаемые вниманию читателей статьи помогут разрешить хотя бы некоторые из «магнитофонных» проблем.

И. В. Канивец,

Санкт-Петербург

Работа с файлами на БЕЙСИКе

Для пользователей БЕЙСИКа-БК не секрет, что производить чтение-запись данных из программы на БЕЙСИКе с помощью операторов OPEN, PRINT# и INPUT# очень неудобно. Вот только несколько тому примеров:

- запись данных занимает много времени, так как информация записывается блоками. В результате скорость обмена с магнитофоном уменьшается по меньшей мере в 3—4 раза по сравнению со скоростью, обеспечиваемой драйвером магнитофона (соответственно во столько же раз возрастает требуемая для хранения одного и того же объема информации длина магнитной ленты);
- при чтении информации из файла оператором INPUT# она обрабатывается достаточно медленно (об этом говорилось, например, в журнале «Информатика и образование», №2 за 1990 г., с. 49—50);
- оператор RPINT# часто не позволяет использовать его в цикле. В этом случае, а также когда между двумя операторами PRINT# вклинивается какой-либо другой оператор, считать с магнитной ленты записанные туда данные не удастся (см.: «Наука и жизнь». 1989. № 5. С. 130);
- если при чтении очередного блока возникает ошибка, то чтение прерывается, а программа прекращает работу;
- во время поиска очередного блока на экран не выдается информация о встреченных файлах, а ведь их имена могли бы помочь сориентироваться в том, где находится искомый блок.

Эти причины и побудили меня разработать подпрограмму (листинг 1), которая реализует непосредственное обращение к драйверу магнитофона (по запросу EMT 36). Подпрограмма позволяет записать любой участок памяти на магнитную ленту, считать данные с ленты в любую область памяти, а также дает возможность программно обрабатывать ошибки чтения и осуществлять фиктивное чтение с выдачей имен просмотренных файлов.

Листинг 1

```
10000 RESTORE 10050
10010 FOR I%=&037700 TO &037754 ST 2%
10020 READ J%
10030 POKE I%,J%
10040 NEXT I%
10050 DATA 5568,8,5569,214,5585,8224,32259,5440,4929,3008,772,
    5570,214,-27566,32258,5569,208,-30690,5581,234,5605,16,135
10060 DEF USR0=&037700
10070 RETURN
10100 POKE &0320,KM%
10110 POKE &0322,ADR%
10120 POKE &0324,DL%
10130 NV¤=USR(NAME¤)
10140 OT%=PEEK(&0320)\256%
10150 RETURN
```

Данная подпрограмма присоединяется к программе пользователя, перед запуском которой необходимо в непосредственном режиме выполнить оператор **CLEAR , &O37700**. До первого обращения к строкам, непосредственно осуществляющим обмен с магнитофоном (начало — строка 10100), необходимо инициализировать драйвер, обратившись к подпрограмме в строке 10000: **GOSUB 10000**. После инициализации можно производить обмен с магнитофоном через подпрограмму, начинающуюся со строки 10100. Для этого необходимо:

- занести в переменную **KM%** команду для драйвера в соответствии с табл. 1 и записать требуемую информацию в переменные ADR%, DL% и NAME¤ (не более 16 символов). Назначение каждой из этих переменных указано в табл. 2;

Таблица 1. Содержимое переменной KM%

Значение	Команда драйвера магнитофона
0	Остановка мотора
1	Пуск мотора
2	Запись
3	Чтение
4	Фиктивное чтение

Таблица 2. Назначение переменных

Имя переменной	Назначение
ВХОДНЫЕ ПЕРЕМЕННЫЕ	
KM%	Требуемая операция ввода-вывода
ADR%	Адрес начала записываемого массива байтов или адрес загрузки при чтении
DL%	Длина массива в байтах (при записи)
NAME¤	Имя файла для записи, чтения и фиктивного чтения
ВЫХОДНЫЕ ПЕРЕМЕННЫЕ	
OT%	Ответ драйвера магнитофона
NV¤	Имя встреченного файла

- вызвать подпрограмму: **GOSUB 10100**;
- после выхода из подпрограммы в переменной **OT%** возвращается код результата выполнения операции в соответствии с табл. 3.

Таблица 3. Ответ драйвера

Значение OT%	Интерпретация
0	Операция завершена без ошибок. Если выполненная команда — «чтение», то начальный адрес прочитанного массива содержится в ячейке &O264, а его длина — в ячейке &O266
1	Имя встреченного файла не совпадает с заданным на чтение. Встреченное имя находится в переменной NV¤, начальный адрес файла — в ячейке &O346, длина — в ячейке &O350
2	Ошибка контрольной суммы
4	Останов по клавише «СТОП»

Ниже приводится текст кодового фрагмента подпрограммы (строка 10050) на языке ассемблера:

```
        MOV #10,R0
        MOV #326,R1
M1:     MOV #20040,(R1)+
        SOB R0,M1
        MOV (R5)+,R0
        MOV (R5),R1
        TST R0
        BEQ M3
        MOV #326,R2
M2:     MOVB (R1)+,(R2)+
        SOB R0,M2
M3:     MOV #320,R1
        EMT 36
        MOV #352,(R5)
        MOV #20,-(R5)
        RTS PC
```

Приведем также примеры использования подпрограммы для записи (листинг 2) и чтения (листинг 3) данных из массива A%(2000) под именем файла: «ДАННЫЕ».

Листинг 2

```
10 DIM A%(2000) ' подготовка массива
20 GOSUB 10000
30 FOR I%=0% TO 2000%
40 A%(I%)=I%
50 NEXT I%
100 ?CHR¤(140%) 'режим расширенной памяти
110 DL%=0%
120 FOR I%=0% TO 2000% ' переписываем содержимое массива
    в экранную память
130 POKE &O40000+DL%,A%(I%) ' с адреса &O40000
140 DL%=DL%+2%
160 NEXT I%
165 NAME¤="ДАННЫЕ" ' имя записываемого файла
170 ADR%=&O40000 ' начальный адрес массива
180 KM%=2% ' режим "запись"
182 ? "НАЖМИТЕ ЛЮБУЮ КЛАВИШУ"
185 IF INKEY¤="" GOTO 185 ' пауза, чтобы включить магнитофон на
                             запись
190 GOSUB 10100
200 ?CHR¤(140%) ' выход из режима РП
```

Листинг 3

```
300 ?CHR¤(140%) ' переход в режим расширенной памяти
310 NAME¤="ДАННЫЕ" ' имя читаемого файла
320 ADR%=&O40000 ' начальный адрес загрузки
330 KM%=3%
340 GOSUB 10100
350 ON OT%+1% GOTO 360,460,480,480,500 ' проверка ответа
360 DL%=0% ' чтение прошло без ошибок
370 FOR I%=0% TO 2000% ' заполнение массива
380 A%(I%)=PEEK(&O40000+DL%)
390 DL%=DL%+2%
400 NEXT I%
410 ?CHR¤(140)
420 FOR I%=0% TO 2000% ' вывод массива на экран
430 PRINT A%(I%);
440 NEXT I%
450 END
460 PRINT "НАЙДЕН ФАЙЛ: "NV¤" НАЧ.АДР.="OCT¤(PEEK(&O346))
    "ДЛИНА="OCT¤(PEEK(&O350)) ' встречен другой файл
470 GOTO 340
480 PRINT CHR¤(140%)"ОШИБКА ЧТЕНИЯ"
490 STOP
500 PRINT CHR¤(140%)"ОСТАНОВ ПО КЛАВИШЕ 'СТОП'"
510 STOP
```


При записи значения ячеек массива (строки 30—50) копируются в экранную область памяти в режиме РП (строки 120—160), содержимое которой записывается на ленту.

При чтении данные загружаются в экранную область памяти и затем пересылаются в массив. ОЗУ пользователя при этом не используется. Если встречен другой файл (прочитан его заголовок), то программа выдает информацию о нем (строка 460) и продолжает поиск нужного файла. Предусмотрен также контроль ошибок обмена с магнитофоном (строка 480) и реакция на нажатие клавиши «СТОП».

Можно привести и еще один пример использования предлагаемой подпрограммы (в режиме фиктивного чтения) для распечатки имен файлов,

хранящихся на кассете, и информации о них (начального адреса и длины) на принтере для создания каталога. Кассета вставляется в магнитофон с начала, запускается программа, листинг которой приводится ниже, и через 30 (или 45) минут, когда будет просмотрена вся кассета, на принтере будет напечатана информация о всех встретившихся файлах.

```
10 GOSUB 10000 ' инициализация
20 KM%=4% ' режим - фиктивное чтение
30 GOSUB 10100 ' обращение к драйверу
40 IF OT%=4% THEN STOP ' нажата клавиша "СТОП"
50 LPRINT NV$;" НАЧ.АДРЕС=";OCT$(PEEK(&0346));
   " ДЛИНА=";OCT$(PEEK(&0350))
60 GOTO 30
```

Этот каталогизатор можно существенно доработать, например, считая по прерыванию витки в просматриваемой кассете, печатать также сведения о положении файлов на ленте. Правда, для этого придется написать свои драйвер магнитофона (вместо EMT 36, но на его основе) и сделать согласующее устройство, позволяющее через порт ввода-вывода БК осуществлять управление магнитофоном с электронным управлением: включать перемотку вправо и влево, воспроизведение, запись и останов. С использованием этого устройства автором разработана программа, которая записывается в начале каждой кассеты, содержит информацию о всех имеющихся на ней файлах и позволяет быстро загрузить или запустить любой из них.

Примечание редактора

Предложенная вниманию читателей подпрограмма реализует довольно-таки тривиальную, но весьма плодотворную идею вызова с помощью кодовой USR-подпрограммы стандартного драйвера магнитофона EMT 36. При этом значения переменных KM%, ADR% и DL% предварительно с помощью операторов POKE заносятся в соответствующие ячейки буфера данных EMT 36, расположенного по стандартному адресу &O320. (Имя файла NAME$ передается непосредственно как аргумент USR.) После окончания операции опять же как значение USR возвращается имя встреченного или обработанного файла (в переменной NV$), а в переменную OT% с помощью оператора PEEK извлекается код результата. (Назначение переменных, а также соответствие кодов команд магнитофона и кодов ответа полностью аналогично указанным в руководстве системного программиста БК для EMT 36.)

Предложенный файловый драйвер наверняка найдет самое широкое применение в программах на БЕЙСИКе, особенно в игровых — для подгрузки различных дополнительных блоков (спрайты, кодированные игровые уровни и лабиринты, добавочные USR-функции). Можно также попытаться организовать таким способом оверлейное исполнение БЕЙСИК-программ.

В заключение хотелось бы отметить следующее. Предложенный драйвер является, по сути, аналогом обычных БЕЙСИКовских операторов BLOAD и BSAVE. Но последние — увы! — нельзя вызывать программно, а использовать их в ручном режиме не всегда удобно.

Д. Ю. Усенков,

Москва

Имитация оператора CASE в вильнюсском БЕЙСИКе

Во многих современных языках высокого уровня, например в СИ или ПАСКАЛе, существует весьма удобный для структурного программирования оператор CASE. Выполняемое им алгоритмическое действие — «многоканальное» ветвление (более общепринятое название — выбор альтернативы): в отличие от обычного IF...THEN...ELSE, в котором предполагается два пути разветвления вычислительного процесса, CASE обеспечивает выполнение одной из многих ветвей (причем их количество не ограничивается) в случае равенства некоторого «опорного» параметра указанному для данной ветви конкретному значению. После выполнения предусмотренных алгоритмом для этого случая вычислений с помощью специального оператора производится переход на «узел ветвления» — точку, в которой сходятся воедино все ветви. Для языка СИ эта конструкция выглядит так:

```
. . . .     /* предыдущие операторы */
switch(<параметр>)          /* "опорный" параметр */
  { case <значение1>:       /* начало первой ветви */
      . . . .               /* вычисления */
      break;                /* выход на узел ветвления */
    case <значение2>:       /* начало второй ветви */
      . . . .               /* вычисления */
      break;
. . . . . .
    case <значениеN>:       /* начало последней ветви */
      . . . .               /* вычисления */
      break;
    default:                /* специальная ветвь для случая */
      . . . .               /*  несовпадения параметра ни с */
      break;                /*  одним значением             */
  }
  . . . .   /* следующий оператор, куда передается */
            /*  управление по break                */
```

В вильнюсском БЕЙСИКе БК-0010.01 предусмотрена близкая по смыслу конструкция **ON <индекс> GOTO <список номеров строк>** или **ON <индекс> GOSUB <список номеров строк>**. Ее недостатком по сравнению с CASE является то, что здесь в качестве параметра используется только порядковый номер ветви (т. е. порядковый номер числа, означающего номер строки для передачи управления). Когда в алгоритмической конструкции типа CASE предполагается использование числовых параметров и case-констант, их в отдельных случаях можно пересчитать в индекс для ON...GOTO... , но далеко

не всегда. Если же используются текстовые параметр и значения, приходится заменять конструкцию CASE на менее удобную и довольно громоздкую цепочку операторов IF.

Тем не менее, в БЕЙСИКе БК-0010.01 можно если не реализовать «настоящий» оператор CASE, то по крайней мере не очень сложным способом смоделировать его работу. Идея здесь очень проста: мы можем записать все значения case-констант в операторе DATA (порядок записи должен быть тем же, что и в списке номеров соответствующих строк), сравнивать в цикле параметр с каждым из значений, а при успешном сравнении — использовать переменную цикла в качестве индекса в операторе ON...GOTO... или ON...GOSUB... (при «досрочном» выходе из цикла не через NEXT его переменная будет, очевидно, содержать порядковый номер требуемой case-константы в списке DATA). Если же переданное значение параметра не равно ни одной из заданных констант, выход из цикла происходит по его нормальному окончанию (через NEXT), тогда значение его переменной равно заданному в операторе FOR максимальному плюс 1, а ON...GOTO... передает управление на следующий после него оператор, реализуя таким образом ветвь default. Конечно, такая конструкция не так удобна, как «настоящий» CASE, но все же она во всех отношениях лучше цепочки IF (особенно при большом количестве альтернатив).

Рассмотрим использование описанной конструкции на небольшом примере. Ниже приводится фрагмент листинга некоторой программы, реализующей управление каким-либо объектом (скажем, вездеходом в игре) с помощью словесных команд: «ВПРАВО», «ВЛЕВО», «ВПЕРЕД» и «НАЗАД» (переменная К$ содержит введенное пользователем слово команды):

```
        . . . .                 ' предыдущие операторы
        RESTORE <S>             ' реинициализация списка case-констант
        FOR I%=1 TO 4           ' цикл перебора (всего возможно 4 команды)
        READ K0$                ' чтение константы из списка
        IF K$=K0$ GOTO <N>        ' сравнение с введенной командой
                                  ' и переход на оператор ON... при успехе
        NEXT                    ' продолжение просмотра списка констант
<S>     DATA ВПРАВО, ВЛЕВО, ВПЕРЕД, НАЗАД    ' список констант
<N>     ON I% GOTO <1>,<2>,<3>,<4>  ' выбор ветви реализации команды
        . . . .                 ' ветвь default
        GOTO <END>              ' выход на узел ветвления - аналог break
<1>     . . . .                 ' ветвь команды "ВПРАВО"
        GOTO <END>
<2>     . . . .                 ' ветвь "ВЛЕВО"
        GOTO <END>
<3>     . . . .                 ' ветвь "ВПЕРЕД"
        GOTO <END>
<4>     . . . .                 ' ветвь "НАЗАД"
<END>   . . . .                 ' узел ветвления (следующий оператор)
```

Аналогичным образом можно смоделировать оператор CASE в случае числового параметра и констант — изменится только тип переменных K% и K0% и констант в списке DATA.

КНИЖНАЯ ПОЛКА БКмана

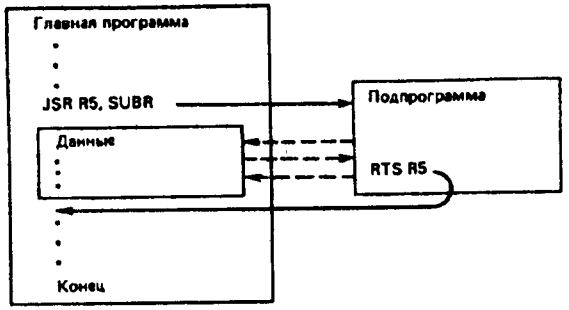
В поисках литературы о программировании на ассемблере БК-0010(.01), БК-0011(М) полезно не забывать о том, что эти компьютеры во многом родственны многочисленным семействам как отечественных ЭВМ («Электроника-60», ДВК, СМ и т. д.), так и американских (PDP-11). Используемые в них процессоры относятся к одному и тому же семейству DEC, а большинство отличий конкретных моделей связано с конструктивными особенностями архитектуры (например, могут различаться адреса системных регистров). Система же команд процессора практически одна и та же на всех моделях, поэтому в книгах по названным выше типам ЭВМ БКманы могут найти немало полезных рекомендаций и приемов программирования на ассемблере.

Предлагаем вниманию читателей две статьи по материалам соответствующих глав из книги американского автора В. Лина «PDP-11 и VAX-11. Архитектура ЭВМ и программирование на языке ассемблера» (М.: Радио и связь, 1989), в которых рассказывается о методах передачи параметров в ассемблерные подпрограммы и об организации кольцевого буфера данных в оперативной памяти.

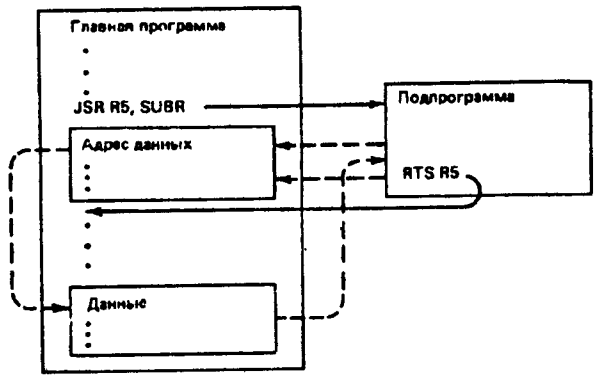
Три основных способа передачи данных в ассемблерные подпрограммы

1. Размещение параметров после команды перехода к подпрограмме

При использовании этого метода мы должны размещать данные или адреса ячеек, в которых они находятся, непосредственно после команды JSR в главной программе. Тогда ответственность за использование регистра связи (указываемого в команде JSR) для получения данных и возврата результатов лежит на подпрограмме. На рис. 1,а показана ситуация, когда после команды вызова подпрограммы размещаются сами данные, а на рис. 1,б — адреса ячеек, содержащих эти данные. Сплошными линиями на рисунках отмечена передача управления из главной программы в подпрограмму и обратно, а штриховыми — пути передачи данных. Следующие примеры иллюстрируют такой способ передачи параметров (листинги адаптированы редакцией к стандарту ассемблер-транслятора М18 для БК).



а) Прямая передача данных



б) Передача адресов в качестве параметров

Рис. 1

Пример 1

```
START: ......
       ......
       JSR R5, SUM
DATA:  .#3.#4
TOTAL: #0
       ......
QUIT:  HALT
SUM:   MOV (R5)+,R0   ; 3 -> R0
       ADD (R5)+,R0   ; (R0)+4 -> R0
       MOV R0,(R5)+   ; РЕЗУЛЬТАТ -> TOTAL
       RTS R5
```

Здесь главная программа вызывает подпрограмму SUM (сложение целых чисел), передает ей данные (3 и 4) и предписывает поместить результат в зарезервированную ячейку памяти с меткой TOTAL.

Давайте сравним алгоритм работы программы с показанным на рис. 1,а (предположим, что вызывающая команда JSR расположена по адресу 2000).

1. Выполнение инструкции JSR R5,SUM:

```
(R5) -> стек
(PC)+4 = #DATA -> R5
#SUM -> PC
```

Сплошная линия на рис. 1,а показывает переход к выполнению первой по счету команды подпрограммы, причем (R5) указывает на первый элемент данных — число 3.

2. Первая команда подпрограммы осуществляет операцию 3 —> R0 (штриховая линия на рис. 1,а), а значение в регистре R5 увеличивается на 2. Теперь он указывает на второй элемент данных — число 4.

3. После выполнения команды ADD имеем:

```
4+(R0) = 4+3 = 7 -> R0
(R5)+2 = #DATA+4 = #TOTAL -> R5
```

Таким образом, после сложения содержимое R5 указывает на ячейку с меткой TOTAL.

4. Очередная команда копирует содержимое R0, равное сумме чисел 3 и 4, в ячейку TOTAL. Затем значение регистра R5 автоматически увеличивается на 2, в результате чего он указывает на ячейку #TOTAL+2.

5. Следующей выполняется команда RTS R5:

```
(R5) = #TOTAL+2 -> PC
верхний элемент стека -> R5
```

Напомним, что старое содержимое регистра R5 было автоматически сохранено в стеке командой JSR. Теперь главная программа продолжает свою работу с команды, расположенной по адресу #TOTAL+2. В отличие от примера 1 здесь используется режим косвенной адресации с автоувеличением, что позволяет с помощью регистра связи R5 получить доступ к данным, находящимся соответственно в ячейках с метками A и B. Предполагается, что результат должен, как и в предыдущем примере, оказаться в ячейке TOTAL.

Пример 2

```
START:  ......
        ......
        JSR R5,SUM
PARAM:  .#A.#B
TOTAL:  .#0
        ......
QUIT:   HALT
A:      .#3
B:      .#4
SUM:    MOV @(R5)+,R0
        ADD @(R5)+,R0
        MOV R0,(R5)+
        RTS R5
```

Обратите внимание на то, что этот способ передачи данных довольно прост: процессор автоматически сохраняет содержимое R5 в стеке, что дает возможность подпрограмме использовать регистр R5 для своих целей. Но имеется и существенный недостаток — смешивание данных с командами (что, например, порождает сложности при дезассемблировании).

Другой немаловажный момент состоит в том, что подпрограмма в обоих примерах задействует для хранения данных также и регистр R0. При этом его старое содержимое (возможно, важное для основной программы) теряется. Для обеспечения универсальности подпрограммы имеет смысл сохранять содержимое R0 в стеке и восстанавливать его перед выходом в главную программу. Подпрограмма из примера 1 может быть модифицирована следующим образом:

```
SUM:    MOV R0,-(SP)   ; ПОМЕСТИТЬ R0 В СТЕК
        MOV (R5)+,R0
        ADD (R5)+,R0
        MOV R0,(R5)+
        MOV (SP)+,R0   ; ВОССТАНОВИТЬ R0 ИЗ СТЕКА
        RTS R5
```

Если предположить, что до выполнения команды JSR значение SP было равно 1072, то карта стековой памяти во время выполнения подпрограммы имеет вид:

До выполнения		Во время выполнения	После выполнения
(R0) = старое (R0)		(R0) — переменное	(R0) = старое (R0)
(R5) = старое (R5)		(R5) — регистр связи	(R5) = старое (R5)
Адрес	Содержимое	Содержимое	Содержимое
1064	******		******
1066	******	старое (R0)	******
1070	******	старое (R5)	******
1072	******	******	******

2. Размещение данных в специально выделенной области

При этом способе передачи данные размещаются в специально выделенной области памяти и с командами не смешиваются. Пару команд, обеспечивающую связывание главной программы и подпрограммы, в данном случае составляют JSR PC,SUBR и RTS PC. Для связи с данными используется какой-либо регистр общего назначения, например R5. Выполняемые действия отражены на рис. 2. Штриховые линии, как и раньше, используются для обозначения путей передачи данных, а сплошные — связи между программой и подпрограммой. Ниже также приводятся два примера, первый из которых иллюстрирует общий алгоритм, а второй — конкретные вычисления.

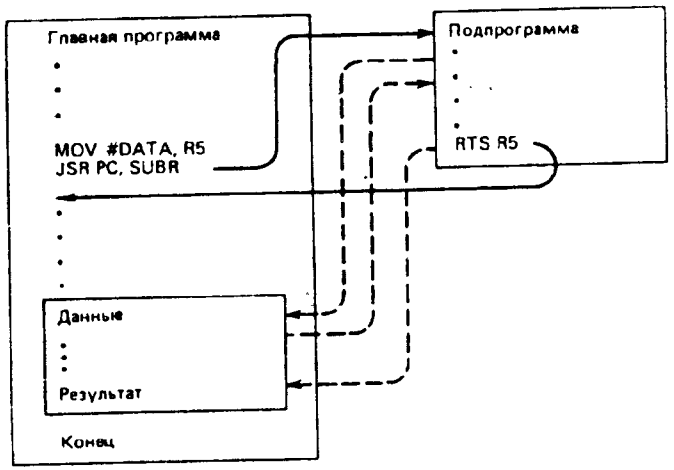


Рис. 2

Пример 1

```
START:  ......
        ......
INIT:   MOV #DATA,R5          ; (R5) <- АДРЕС ДАННЫХ
        JSR PC,SUBR           ; (PC) -> СТЕК, #SUBR -> PC
        ......
QUIT:   HALT
DATA:   .#<D1>                ; <D1>, <D2>,.. -
        .#<D2>                ;   КОНКРЕТНЫЕ ЧИСЛА
        ......
        .#<Dn>
SUBR:   MOV (R5)+,<НАЗН>      ; <НАЗН> - АДРЕС ОБЛАСТИ,
        MOV (R5)+,<НАЗН+2>    ;   КУДА НУЖНО СКОПИРОВАТЬ
        ......                ;   МАССИВ ДАННЫХ
        MOV (R5)+,<НАЗН+2*n>
        ......
        RTS PC
```

Пример 2. Использование подпрограммы SUM для определения суммы всех элементов некоторого массива данных (предполагаем, что сумма не превысит 32000, так что переполнения не произойдет). Пусть первый элемент массива указывает количество суммируемых элементов (размер массива) и равен 10.

```
START:  MOV #DATA,R5      ; (R5) <- АДРЕС МАССИВА
        JSR PC,SUM        ; (PC)+4 -> СТЕК; #SUM -> PC
QUIT:   HALT
DATA:   .#10              ; РАЗМЕР МАССИВА ДАННЫХ
        .#1.#-2.#3.#-4.#5.#-6.#7.#10    ; ДАННЫЕ
TOTAL:  .#0               ; МЕСТО ДЛЯ РЕЗУЛЬТАТА
SUM:    CLR R0            ; R0 - СУММАТОР
        MOV (R5)+,R1      ; R1 - СЧЕТЧИК СЛАГАЕМЫХ
LOOP:   ADD (R5)+,R0      ; ЦИКЛ СЛОЖЕНИЯ
        SOB R1,LOOP       ; (R1) = (R1)-1; ЕСЛИ НЕ 0,
                          ;   ТО ОЧЕРЕДНОЙ ШАГ ЦИКЛА
        MOV R0,TOTAL      ; РЕЗУЛЬТАТ -> TOTAL
        RTS PC            ; СТЕК -> PC
```

В результате, как легко видеть, ячейка TOTAL будет содержать число 14 (восьм.).

3. Пересылка данных через стековую память

Этот способ наиболее удобен, когда нужно реализовать вложенные подпрограммы или при обслуживании прерываний. В основном пересылка параметров через стек осуществляется в несколько этапов:

- главная программа копирует данные в стек;
- подпрограмма извлекает данные из стека, обрабатывает и помещает в стек результат;
- главная программа извлекает результат из стека и использует по назначению.

При таком подходе подпрограмме не надо заботиться о том, где следует искать данные, поскольку они всегда находятся в стеке. Пересылка параметров этим способом показана на рис. 3 (сплошные линии — связь между программами, штриховые — передача данных).

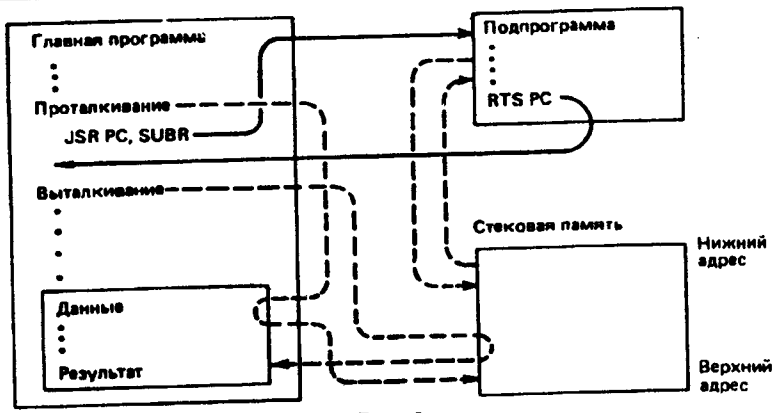


Рис. 3

В этом случае мы должны до выполнения команды JSR протолкнуть все данные в стек, а после возврата по RTS — вытолкнуть из стека результат и поместить его по адресу назначения. Подпрограмма же во время работы извлекает данные из стека и помещает туда результат.

Этот способ кажется довольно простым, однако некоторые детали требуют отдельного рассмотрения. Вспомним, что команда JSR помещает в стек скорректированное содержимое регистра PC для обеспечения возврата из подпрограммы. Но и мы также помещаем в стек данные, а затем извлекаем их оттуда. Очевидно, при недостатке внимательности может возникнуть путаница. Для прояснения ситуации воспользуемся примером, аналогичным примеру 2 из предыдущего раздела.

Пример 1

```
START:  MOV SP,R4          ; СКОПИРОВАТЬ SP В R4
        TST -(R4)          ; УМЕНЬШИТЬ R4 НА 2
        MOV #DATA,R5       ; R5 - УКАЗАТЕЛЬ ДАННЫХ
        MOV (R5)+,R3       ; R3 - СЧЕТЧИК ДАННЫХ В МАССИВЕ
        MOV R3,R2          ; R2 - СЧЕТЧИК ДАННЫХ В СТЕКЕ
PUSH:   MOV (R5)+,-(R4)    ; ПРОТОЛКНУТЬ В СТЕК
        SOB R3,PUSH        ;   ВСЕ ДАННЫЕ
CALLSB: JSR PC,SUM         ; (PC)+4 -> СТЕК; #SUM -> PC
        MOV (R4),TOTAL     ; РЕЗУЛЬТАТ -> TOTAL
QUIT:   HALT
DATA:   .#<n>              ; КОЛИЧЕСТВО ДАННЫХ
        .#<D1>.#<D2>... .#<Dn>     ; ДАННЫЕ
TOTAL:  .#0                ; ЗАРЕЗЕРВИРОВАНО ПОД РЕЗУЛЬТАТ
SUM:    CLR R0             ; R0 - СУММАТОР
LOOP:   ADD (R4)+,R0       ; ИЗВЛЕЧЕНИЕ ДАННЫХ ИЗ СТЕКА И
        SOB R2,LOOP        ;   ИХ СУММИРОВАНИЕ
        MOV R0,-(R4)       ; РЕЗУЛЬТАТ -> СТЕК
        RTS PC             ; ВОЗВРАТ
```

Во избежание путаницы в стеке между данными и сохраненным содержимым регистра PC мы в первую очередь копируем значение указателя стека SP в регистр R4, который затем используется как указатель данных в стеке. Затем уменьшаем R4 на 2, чтобы пропустить одно машинное слово, которое

чуть позже будет использовано для сохранения скорректированного содержимого регистра PC. После этого мы назначаем регистр R5 указателем данных в массиве, а R2 и R3 — счетчиками данных в стеке и в первоначальном массиве, соответственно. После завершения проталкивания данных в стек производится вызов подпрограммы SUM, которая суммирует данные, извлекая их из стека, а взамен помещает туда результат. И наконец, после возврата главная программа извлекает из стека результат и помещает его в ячейку TOTAL.

Ниже представлена карта стека, показывающая его состояние во время работы подпрограммы (предполагаем, что изначально значение SP было равно 1172. В первой колонке таблицы дано содержимое регистров после завершения выполнения команды MOV R3,R2. Вторая колонка показывает состояние стековой памяти после команды SOB R3,PUSH (указатели R5 и R4 использованы для копирования данных в стек из первоначального массива). Третья колонка — помещение скорректированного содержимого регистра PC в стек (по адресу 1170) по команде JSR PC,SUM и переход к отработке подпрограммы. Четвертая колонка показывает перенос результата (суммы данных) из регистра R0 в стек (в ячейку с адресом 1166) командой MOV R0,–(R4) (данные все еще находятся в памяти, но уже не существенны для программы). В пятой колонке видно, что команда RTS PC помещает в регистр PC содержимое ячейки стека с адресом 1170, после чего происходит возврат в основную программу. И наконец, шестая колонка показывает результат выполнения команды MOV (R4),TOTAL. Стек при этом возвращается к первоначальному состоянию.

Состояние регистров после выполнения команды MOV R3,R2	Состояние стековой памяти			
	После выполнения команды SOB R3,PUSH		После выполнения команды JSR PC,SUM	
	Адрес	Содержимое	Адрес	Содержимое
(SP)=1172	....	Dn	....	Dn
(R4)=1170	....	.....	....	.....
(R5)=#DATA	....	.....	....	.....
(R3)=n	1166	D1	1166	D1
(R2)=n	1170	******	1170	#CALLSB+4
	1172	******	1172	******

После выполнения команды MOV R0,–(R4)		После выполнения команды RTS PC		После выполнения команды MOV (R4),TOTAL	
Адрес	Содержимое	Адрес	Содержимое	Адрес	Содержимое
....	******	....	******	....	******
....	.....	....	.....	....	.....
1164	******	....	******	....	.....
1166	Результат	1166	Результат	1166	******
1170	#CALLSB+4	1170	******	1170	******
1172	******	1172	******	1172	******

(Обратите внимание на то, что для размещения данных используются ячейки стековой памяти с адресами, меньшими (SP)–2, т. е. область ЗА ПРЕДЕЛАМИ стека, тогда как обычно данные заносятся в стек командами MOV #<Di>,–(R6) перед вызовом подпрограммы и данные оказываются ВЫШЕ вершины стека. Это позволяет избавиться от необходимости «чистить» стек после использования данных, но требует внимания при сохранении задействованных в подпрограмме регистров (тогда R4 нужно уменьшать уже не на 2, а на величину 2*n+2, где n — количество сохраняемых в стеке регистров. — *Прим. ред.*)

В заключение приведем пример конкретных вычислений с пересылкой параметров через стек. Обратите внимание, что подпрограмма здесь имеет дело только со стековой памятью:

```
START:  MOV SP,R4          ; СКОПИРОВАТЬ SP В R4
        TST -(R4)          ; УМЕНЬШИТЬ (R4) НА 2
        MOV #DATA,R5       ; R5 - УКАЗАТЕЛЬ ДАННЫХ
        MOV (R5)+,R3       ; R3 - СЧЕТЧИК ДАННЫХ В МАССИВЕ
        MOV R3,R2          ; R2 - СЧЕТЧИК ДАННЫХ В СТЕКЕ
PUSH:   MOV (R5)+,-(R4)    ; ПРОТОЛКНУТЬ В СТЕК
        SOB R3,PUSH        ;   ВСЕ ДАННЫЕ
CALLSB: JSR PC,SUM         ; (PC)+4 -> СТЕК; #SUM -> PC
        MOV (R4),TOTAL     ; РЕЗУЛЬТАТ -> TOTAL
QUIT:   HALT
DATA:   .#10               ; КОЛИЧЕСТВО ДАННЫХ
        .#1.#-2.#3.#-4.#5.#-6.#7.#10 ; ДАННЫЕ
TOTAL:  .#0
SUM:    CLR R0             ; R0 - СУММАТОР
LOOP:   ADD (R4)+,R0       ; ИЗВЛЕЧЕНИЕ ДАННЫХ ИЗ СТЕКА
        SOB R2,LOOP        ;   И СУММИРОВАНИЕ
        MOV R0,-(R4)       ; РЕЗУЛЬТАТ -> СТЕК
        RTS PC
```

После выполнения программы ячейка TOTAL содержит число 14 (восьм.).

От редакции

Сравнив между собой описываемые в статье методы, можно сделать следующие выводы. При программировании на БК данные для подпрограмм чаще всего размещаются в регистрах общего назначения (но их количество ограничено) или в отдельной области памяти. Причем, в зависимости от ситуации, либо в подпрограмму передается указатель на массив данных (как это сделано по второму методу из описанных в статье), либо производится обращение из подпрограммы к отдельным данным по адресам (меткам) содержащих их ячеек ОЗУ.

Способ с передачей параметров через стек применяется в вильнюсском БЕЙСИК-трансляторе, причем возможная путаница между данными в стеке и сохраняемым там же значением регистра связи устраняется самим механизмом вызова подпрограмм по их «шитому коду» (вместо команды JSR используется не сохраняющая адрес возврата в стеке JMP @(R4)+ , где R4 — указатель на список «шитых кодов»). И наконец, метод размещения параметров сразу же после команды вызова подпрограммы (первый способ в статье), несмотря на некоторое неудобство при

дезассемблировании, в последнее время завоевывает все большее число сторонников. Однако он требует перезадания для каждой подпрограммы даже одинаковых параметров (например, одного и того же цвета рамки окна или текста в нем) и не позволяет при циклическом вызове динамически менять значения параметров.

Следующий листинг демонстрирует возможные варианты передачи данных по этому методу (ассемблер стандарта М18)

```
        .....          ; предыдущие команды основной программы
    JSR R5,SUBR        ; вызов подпрограммы
; варианты данных:
    .#100              ; константа (1)
    .#METKA1           ; адрес метки (неперемещаемый формат) (2)
    .@METKA2           ; адрес метки (перемещаемый формат) (3)
    .#METKA3           ; содержимое ячейки (неперемещаемый формат) (4)
    .@METKA4           ; содержимое ячейки (перемещаемый формат) (5)
    .A:<текст>
    .E                 ; строка, заканчивающаяся нулевым байтом (6)
    <OPERATOR>         ; однословная команда ассемблера (7)
    .....              ; последующие команды основной программы

SUBR: JSR R4,@#110346 ; сохранение в стеке R0-R4
      MOV (R6),R5     ; R5 - указатель данных
;
      MOV (R5)+,R0    ; извлечение константы (1)
;
      MOV (R5)+,R0    ; извлечение адреса метки (2)
;
      MOV R5,R0       ; извлечение адреса перемещаемой метки (3)
      ADD (R5)+,R0    ;
;
      MOV @(R5)+,R0   ; извлечение содержимого ячейки (4)
;
      MOV R5,R0       ; извлечение содержимого перемещаемой
      ADD (R5)+,R0    ;  ячейки (5)
      MOV @R0,R0      ;
;
1:    MOVB (R5)+,R0   ; извлечение байтов текстовой строки (6)
      BNE 1           ;  до нулевого включительно
      BIT #1,R5       ; если адрес в R5 нечетный,
      BEQ 2           ;  привести его к четному
      INC R5          ;
2:    NOP             ; продолжение работы подпрограммы
;
      MOV (R5)+,ADR   ; помещение переданной в подпрограмму
                      ;  команды внутрь листинга
;
      JSR R4,@#110362 ; восстановление из стека R4 R0
      RTS R5          ; возврат в основную программу
```

Передача в подпрограмму ассемблерных команд позволяет реализовать самомодификацию программы, заменяя некоторый фрагмент одной и той же подпрограммы при различных ее вызовах. Думается, читатели сами догадаются, как осуществить передачу ассемблерных команд, занимающих два или три машинных слова (в подпрограмме потребуется, соответственно, две или три команды: MOV (R5)+,ADR , MOV (R5)+,ADR+2 и MOV (R5)+,ADR+4)

Программа ввода-вывода данных с помощью очереди буферов

Рассмотрим следующую ситуацию. Пусть входные данные могут время от времени поступать в ЭВМ, причем моменты их появления на входе непредсказуемы. Эти данные нужно каким-либо образом обработать, вывести на принтер или записать на дискету (действия, занимающие определенное время), но при этом нужно сохранять постоянную готовность к приему очередного элемента данных.

Решить такую задачу можно следующим образом. Отведем небольшой участок оперативной памяти в качестве временного буфера для сбора данных и уже из этого буфера будем переписывать накопленные порции данных на диск или по другому адресу назначения.

Структура буфера данных

На рис. 1 показана карта буферной памяти на байтовой основе. Начальным адресом буфера является метка QUEBUF, а его емкость равна (MAX+1) байтов или 1/2 (MAX+1) машинных слов. Определим также два указателя: DBNN — вход в очередь буфера данных, DBOUT — выход из очереди буфера данных. Первый из них всегда указывает на ту ячейку, в которую будут записываться следующие данные, а второй — на ячейку, содержимое которой ожидает вывода из буфера. (Конечно, можно использовать и только один указатель, например, вначале «насытив» буфер данными, а затем сбросив указатель на начальный адрес и использовав его для вывода из буфера. Однако конструкция с одним указателем имеет и серьезный недостаток. Ведь

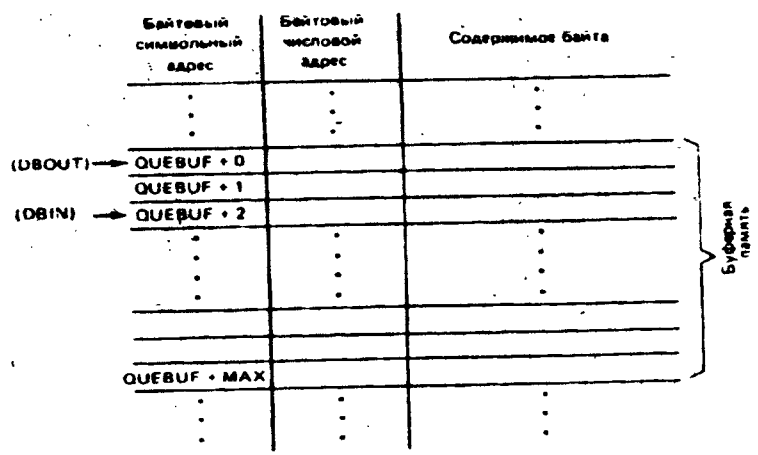


Рис. 1

в процессе вывода из буфера в компьютер могут поступить новые данные, их нужно срочно занести в буфер, причем с гарантией, что они не запортят хранящиеся там уже собранные данные.)

Конфигурацию с двумя указателями лучше всего понять, представив буфер в кольцевой (циклической) форме, как показано на рис. 2. Первоначально оба указателя показывают на максимальный байт MAX. По мере ввода данных в буфер содержимое указателя DBIN увеличивается в направлении по часовой стрелке, в то время как указатель DBOUT все еще остается на месте. Когда данные перестанут поступать, собранные данные, располагающиеся в памяти между указателями, будут выведены из буфера, а указатель DBOUT, соответственно, увеличен. В результате по мере вывода данных «разрыв» между двумя указателями будет уменьшаться. Таким образом, мы получаем буфер бесконечной емкости (при сравнительно небольшом объеме занимаемой памяти!), если только «разрыв» всегда меньше, чем MAX, и больше, чем нуль байтов. Мы можем использовать буферное пространство вновь и вновь без необходимости заботиться о том, чтобы не были испорчены уже собранные или выведены неверные данные. Для поддержания такой идеальной ситуации необходимо иметь в виду следующие условия:

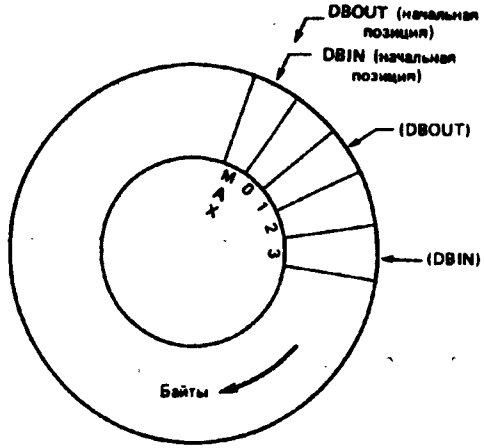


Рис. 2

- совпадение (DBIN) с (DBOUT) означает, что буфер полон;
- совпадение (DBOUT) с (DBIN) подразумевает, что буфер пуст;
- всякий раз, когда любой из указателей равен MAX, его нужно сбросить в начальную позицию.

Разработка алгоритма

Займемся теперь разработкой алгоритма программы сбора кодов нажимаемых клавиш. Для упрощения сымитируем процесс записи собранных данных на диск их занесением в некоторую область памяти. Функциональная блок-схема программы показана на рис. 3. Здесь мы используем прерывания для ввода данных в циклический буфер и подпрограмму для вывода данных из буфера в предназначенную для них область памяти («на устройство назначения»).

Подпрограмма завершает свою работу, если данные из буфера полностью выведены по назначению либо входных данных больше не поступает. В соответствии с этими двумя условиями запрограммируем проверку пустоты буфера (первое условие) и ожидание поступления данных в течение некоторого времени (второе).

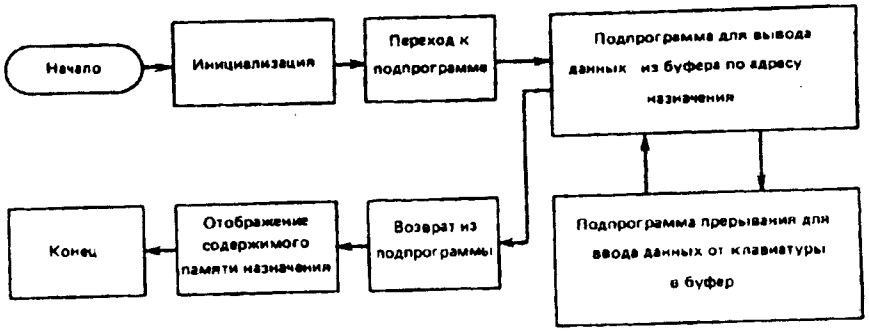


Рис. 3

Прерывание выполняется, когда на клавиатуре ЭВМ пользователь нажимает какую-либо клавишу. Поскольку среди кодов клавиш есть управляющие (не отображаемые), значения которых меньше #40, подпрограмма обслуживания прерываний будет заменять их на два байта: «¬» (код #136) и символа, восьмеричный код которого равен [#100+код управляющей клавиши]. Например, при нажатии клавиши «курсор вверх» будет отображена пара «¬Z». Кроме того, подпрограмма обслуживания прерываний завершает работу при заполнении всего буфера или при нажатии клавиши-«терминатора» «!». Ниже дан ассемблерный листинг (основная программа, модуль QUE1 — подпрограмма вывода данных из буфера, модуль QUE2 — обработчик прерываний от клавиатуры) и блок-схемы модулей (рис. 4—6).

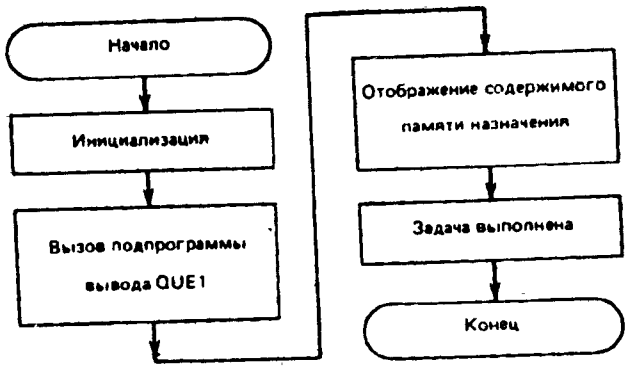


Рис. 4

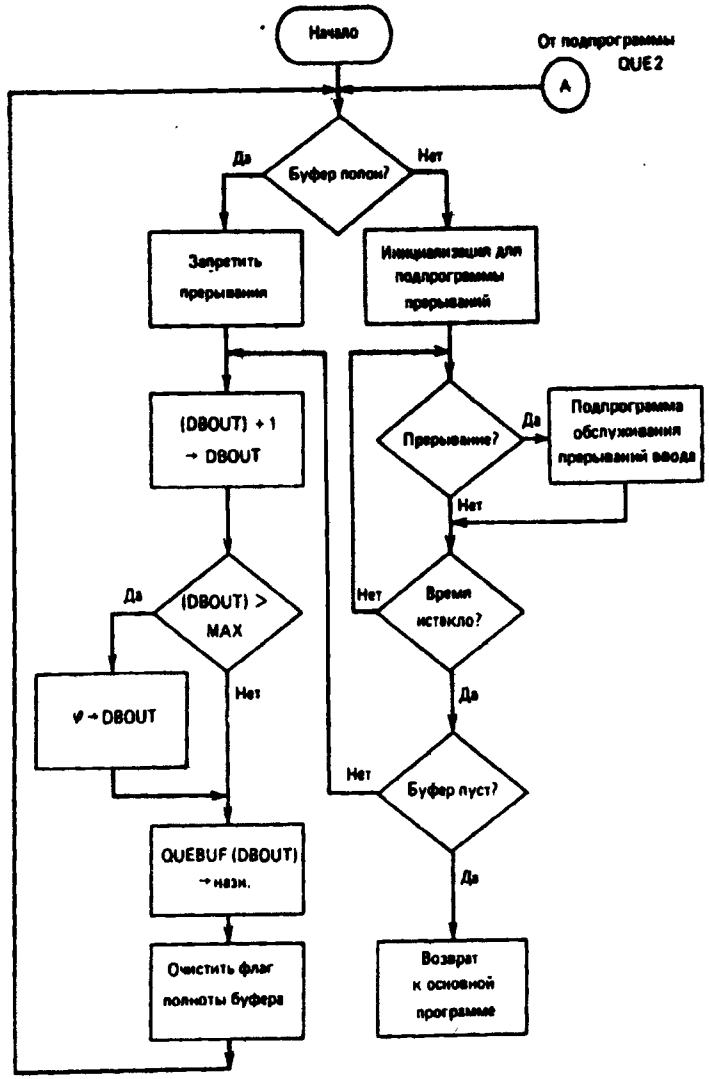


Рис. 5

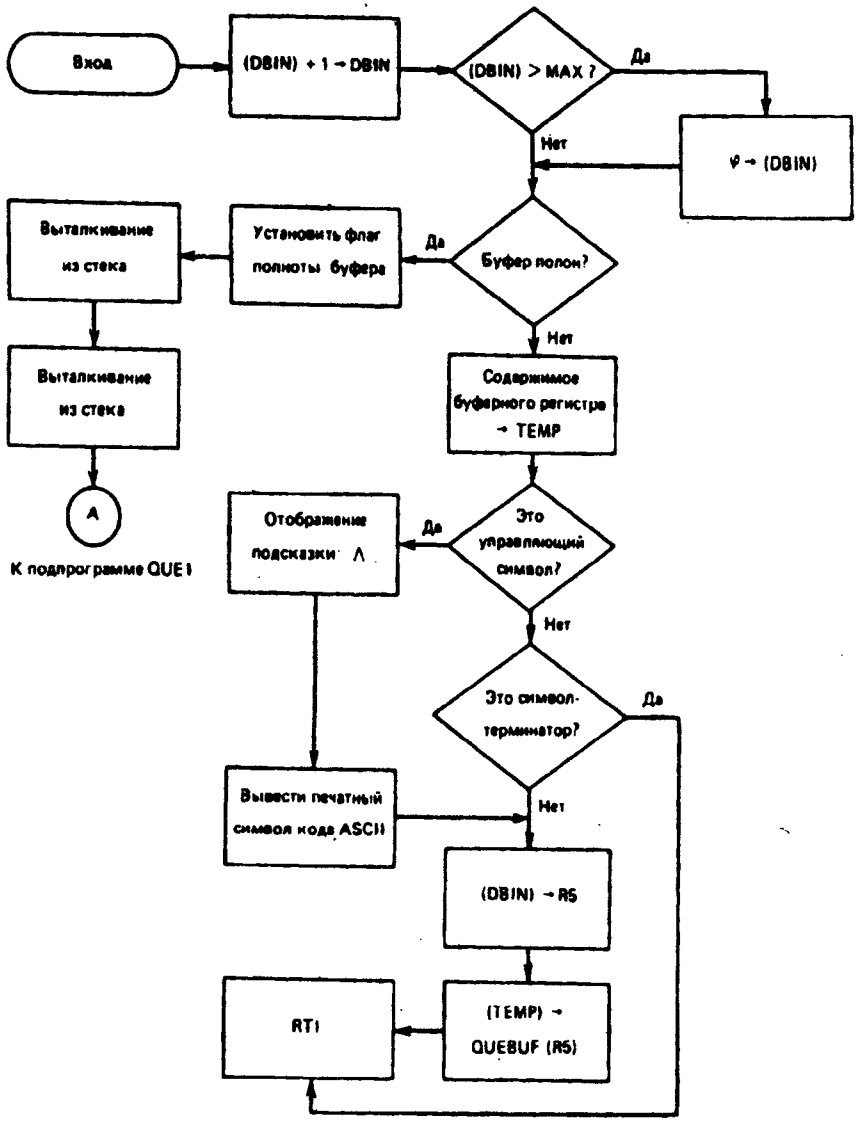


Рис. 6


```
; ОСНОВНАЯ ПРОГРАММА
; Использует модули QUE1 и QUE2
;   QUEBUF - начальный адрес буфера (константа)
;   MAX    - емкость буфера в байтах
;   DBOUT  - указатель выводимых из буфера данных
;   DBIN   - указатель вводимых в буфер данных
;   DST    - начальный адрес области назначения для данных
;-----------------------------------------------------------
START:  CLR R1
        JSR PC,QUE1
        HALT                    ; задача выполнена
;-----------------------------------------------------------
;       Модуль QUE1 - подпрограмма выдачи данных из буфера
;-----------------------------------------------------------
QUE1:   TST BFULL
        BNE FULL
        MTPS #0                 ; инициализация PSW
        BIC #100,@#177660       ; разрешение клавиатурных прерываний
        MOV #QUE2,@#60          ; установка вектора прерывания
        MOV #340,@#62           ; установка PSW прерывания
        CLR R3                  ; инициализация таймера
        CLR R4
WAIT:   INC R3
        BNE WAIT
        INC R4
        CMP TLIMIT,R4           ; время исчерпано?
        BPL WAIT
EMPTY:  CMP DBOUT,DBIN          ; буфер пуст?
        BNE L1                  ; нет
        RTS PC                  ; да - возврат из подпрограммы
FULL:   BIS #100,@#177660       ; запрет прерывания
L1:     INC DBOUT
        CMP DBOUT,MAX           ; DBOUT указывает на MAX?
        BLE XFER
        CLR DBOUT               ; сбросить DBOUT
XFER:   MOV DBOUT,R2
;       MOV QUEBUF(R2),R0       ; вывод из буфера
;       JSR PC,PRINT            ;  на принтер
        MOV QUEBUF(R2),DST(R1) ; данные -> по назначению
;       MOV DST(R1),R0          ; вывод из массива
;       JSR PC,PRINT            ;  на принтер
        INC R1
        CLR BFULL               ; сбросить флаг "буфер полон"
        BR QUE1                 ; следующие данные
DST:    .+20                    ; место для данных
        .E
TLIMIT: .#7
;-----------------------------------------------------------
;       Модуль QUE2 - обработчик прерываний от клавиатуры
;-----------------------------------------------------------
QUE2:   INC DBIN
        CMP DBIN,MAX            ; DBIN равно верхнему пределу?
        BLE FULCHK
        CLR DBIN                ; сбросить DBIN
FULCHK: CMP DBIN,DBOUT          ; буфер полон?
        BNE RD                  ; нет - перейти к чтению кода
FULSET: INC BFULL               ; да - установить флаг "буфер полон"
        TST (SP)+               ; удалить сохраненное (PC)
```

```
        TST (SP)+           ; удалить сохраненное (PSW)
        JMP QUE1            ; буфер полон - выводить коды
RD:     JSR PC,READ         ; чтение кода клавиши в TEMP
        CMPB TEMP,#40       ; введен печатаемый код?
        BGT NORM
CNTRL:; MOV #136,R0         ; вывод символа "¬"
       ; JSR PC,PRINT       ;  на принтер
        BIS #100,TEMP       ; TEMP <- (код упр. клавиши + 100)
NORM:  ; MOV TEMP,R0        ; вывод символа
       ; JSR PC,PRINT       ;  на принтер
        CMPB #41,TEMP       ; символ завершения ("!")?
        BNE SKP1
STOP:   RTI                 ; выход из прерывания
SKP1:   MOV DBIN,R5
        MOVB TEMP,QUEBUF(R5)
        BR STOP
DBIN:   .#17
DBOUT:  .#17
BFULL:  .#0
MAX:    .#17
TEMP:   .#0
QUEBUF: .+20
        .E
;
READ:   BIT #200,@#177660   ; код в регистре данных?
        BEQ READ
        MOVB @#177662,TEMP  ; код символа -> TEMP
        CLR R3
        CLR R4
        RTS PC
```

От редакции

Создание кольцевого буфера на БК представляет не только чисто теоретический, но и практический интерес. Стандартная реализация функции ввода символа (EMT 6) и большинство ее «самодельных» аналогов в различных программах предполагают наличие буфера всего для одного символа, прочие же (если предыдущий код еще не считан) попросту игнорируются. Особенно это заметно в текстовых редакторах: если при наборе текста возникает необходимость выполнить какое-либо действие (например, выход за край экрана с его «перерисовкой» или обращение к диску в VorteX 4.0), программа перестает воспринимать нажатия на клавиши и «съедает» два-три символа. Где уж тут мечтать о динамической обработке текста во время его ввода (скажем, об автоматическом выравнивании строк по правому краю)?

На IBM же даже при очень длительных операциях обработки текста и невысоком быстродействии компьютера все введенные с клавиатуры символы запоминаются и поступают в программу сразу после окончания операции (только иногда, когда пользователь удерживает клавишу нажатой, писк динамика предупреждает о переполнении буфера ввода). Это запоминание нажатых клавиш действует в том числе и во время дисковых операций, например при подчитывании очередного фрагмента текста функцией View (клавиша F3 в Norton Commander). Данное преимущество IBM получает благодаря наличию буфера вводимых символов емкостью в несколько десятков байт.

То же удобство может быть получено и на БК, если реализовать буфер ввода, рассчитанный на несколько символов, воспользовавшись описанным в статье механизмом кольцевого буфера. Надеемся, что разработчики текстовых редакторов и других программ для БК по достоинству оценят приведенные в статье рекомендации.

ГЛЮКАДЕМИЯ

Жаргонный термин «глюк», соответствующий в английском языке не менее жаргонному словечку «bug» («жучок»), известен многим БКманам, которым не раз приходилось сталкиваться с недоработанными и недоотлаженными программами. Для борьбы с этим неприятным явлением и предназначается рубрика «Глюкадемия», открытая в №3 за 1994 г.
Редакция приглашает читателей принять участие в обнаружении «глюков» и поиске способов их устранения, а авторов «отличившихся» программ — к исправлению указанных ошибок в новых версиях.

Доработка подгружаемой функции FILES для программы DESS

Многим пользователям БК, имеющим дисковод, хорошо знакомы программы семейства DESS (автор М. Королев) — весьма неплохая помощь, когда приходится вручную восстанавливать испорченную дискету. Одной из наиболее совершенных является версия 2.7, в которой имеется полезная возможность подгружать в процессе работы дополнительные пользовательские драйверы для реализации функций, отсутствующих в стандартном наборе. В комплект поставки, в частности, входят три таких драйвера: FILES.DSS — запись информации из блоков (секторов) диска в файл, ASSEMB.DSS — просмотр записанных на диске кодов в виде ассемблерных мнемоник и MACRO.DSS — реализация макрокоманд.

Но несмотря на все «плюсы» данной программы, в ней (а точнее — в подгружаемом драйвере FILES) имеется и весьма неприятный «глюк». Какой — станет ясно из следующей короткой истории.

...В один далеко не прекрасный день у автора этих строк «полетела» дискетка. (Причины этого прискорбного события мы здесь обсуждать не будем, но в результате оказались полностью затерты обе копии FAT, что практически исключает возможность полностью восстановить файловую структуру диска.) К счастью, большинство записанных на ней файлов были не единственными копиями, но все же несколько достаточно важных текстов нужно было спасать: требовалось разыскать на диске секторы, содержащие нужный текст, считать их, записать в виде файлов, а затем объединить все фрагменты в требуемом порядке в единое целое. (Подробнее о восстановлении файлов можно прочитать в № 1 за 1993 г., с. 99.)

Найти нужные секторы удалось довольно легко (хотя это и потребовало некоторого времени) с помощью уже упомянутой DESS, а вот записать их содержимое в файл — увы... К сожалению, в подгружаемом драйвере FILES запись такого файла предполагается делать на тот же самый диск, секторы которого считывались. (Можно конечно, переадресоваться на другое устройство, указав его букву с двоеточием в имени файла, но как быть тем пользователям

БК-0010(.01), у кого есть только один дисковод A: ?) Вот это и есть «глюк»: ведь диск-то испорчен, поэтому запись на него невозможна! (А даже если и не испорчен — записываемый файл может затереть содержимое тех самых секторов, которое нужно восстановить.) Словом, хваленый подгружаемый драйвер на деле оказался бесполезным «прибамбасом», а посему пришлось прибегнуть к надежному, но позволяющему считывать только один сектор за каждый прием средству — программе SCREW BlockSaver (см. №1 за 1994 г., с. 109)...

А теперь посмотрим, как исправить вышеназванный «глюк». Очевидно, нужно доработать стандартный драйвер FILES.DSS таким образом, чтобы дать пользователю возможность перед записью файла сменить диск, а после записи — снова установить исходный. Ниже приводится ассемблерный листинг исправленного драйвера (прежний вариант был дезассемблирован с помощью отладчика DEBU10).

```
START:  MOV #214,R0     ; включить режим РП
        EMT 16
        JSR R1,@#1066   ; вывод текста на экран (*)
        .A:               --- DESS BlockSaver 1994 ---
        .B:12.E
        MOV @#1016,R1   ; номер текущего блока (*)
        MOV @#1024,R0   ; номер помеченного блока (*)
        SUB R0,R1       ; вычисление кол-ва сохраняемых блоков
        CMP R1,#27      ; длина фрагмента не более 27 блоков?
        BLOS 1          ; да
        JSR R1,@#1066   ; если больше, выдать сообщение (*)
        .A: Файл слишком велик!
        .E
        JSR PC,KEY      ; ждать нажатия клавиши
        CLRB @#105      ; очистить код нажатой клавиши
        MOV #214,R0     ; выход из режима РП
        EMT 16          ;
        RTS PC          ; возврат
1:      JSR R1,@#1066   ; вывести запрос имени файла (*)
        .A: Имя файла:
        .B:40.E
        MOV #326,R3     ; адрес начала имени в буфере EMT36
        JSR PC,@#1046   ; ввод строки (*)
2:      CMP R1,#346     ; введены все 16 символов?
        BHIS 3          ; да
        MOVB #40,(R1)+  ; иначе дополнить хвостовыми пробелами
        BR 2            ;
3:      MOV @#1024,R0   ; номер помеченного диска (*)
        MOV #40000,R2   ; файл будем формировать в ОЗУ экрана
        MOV @#1016,R1   ; номер текущего блока (*)
        SUB R0,R1       ; количество блоков
        BIC #177400,R1  ; вычисление кол-ва байт
        SWAB R1         ;  в выбранных блоках
        ASL R1          ;  (умножение R1 на 1000)
        ADD 2(SP),R1    ; добавить часть текущего блока (*)
        SUB @#1004,R1   ;  до курсора (*)
        MOV R1,@#324    ; записать в буфер EMT36 длину файла
        ASR R1          ; длина в двухбайтных словах (R1=R1/2)
        JSR PC,@#1056   ; чтение заданного числа блоков (*)
        BCC 4           ; бит C обнулен - ошибок нет
        JSR R1,@#1066   ; иначе - игнорировать ли ошибку? (*)
```



```
        .A: [I] - игнорировать
        .E
        CLRB @#105      ; сбросить предыдущие нажатия клавиш
        JSR PC,@#1076   ; ввод кода клавиши (*)
        CMPB R0,'I'     ; нажато "I"?
        BNE 3           ; нет - попробовать считать заново
4:      CLR @#177130    ; остановить диск
        JSR R1,@#1066   ; запрос на смену диска (*)
        .B:12.B:12.A:  Установите другой диск
        .B:12.A: и нажмите любую клавишу
        .E
5:      JSR PC,KEY      ; ждать нажатия клавиши
        MOV #500,R2     ;  генерация звука
        MOV #1000,R3    ;    заданной частоты
        JSR PC,@#102062 ;    и длительности
        MOV #324,R1     ; записать в буфер EMT36
        MOV #40000,-(R1) ;  начальный адрес файла
        MOV #2,-(R1)    ; команда на запись
        JSR PC,@#100602 ; подпрограмма монитора: запись файла
        TSTB @#321      ; ошибки при записи не было?
        BEQ 6           ; нет
        CMPB @#321,#4   ; или есть, но это нажатие на "СТОП"?
        BNE 5           ; при "настоящей" ошибке все повторить
6:      JSR R1,@#1066   ; запрос на вставку исходного диска
        .B:12.B:12.B:12.A:  Установите исходный диск
        .B:12.A: и нажмите любую клавишу
        .E
        JSR PC,KEY      ; ждать нажатия клавиши
        CLRB @#105      ; очистить код нажатой клавиши
        MOV #214,R0     ; выход из режима РП
        EMT 16
        RTS PC          ; выход
; ожидание нажатия на любую клавишу
KEY:    BIT #100,@#177716
        BEQ KEY
1:      BIT #100,@#177716
        BNE 1
        RTS PC
```

Драйвер представляет собой перемещаемую подпрограмму, особенностью которой являются обращения к заранее оговоренным в документации к DESS ячейкам ОЗУ и готовым интерфейсным подпрограммам (соответствующие строки листинга отмечены знаком (*) в конце комментариев). Чтобы сделать листинг более понятным, кратко обсудим их назначение (полный список подпрограмм и ячеек ОЗУ приводится в документации к DESS).

- Подпрограмма @#1166: осуществляет вывод на экран строки символов, записанной в ассемблерном листинге сразу же за ее вызовом (функциональный аналог команды EMT 20, обеспечивающий перемещаемость программы). Строка должна заканчиваться нулевым байтом. Вызов подпрограммы производится командой JSR R1,@#1066.
- Подпрограмма @#1046: обеспечивает ввод строки символов в ОЗУ. Адрес начала буферной области перед вызовом задается в R3, после выхода содержимое R3 не меняется, а в R1 возвращается адрес завершающего строку кода #12 (восьм.).

О перемещаемости (при задании начального адреса) в данном случае должен заботиться сам пользователь, но в нашем драйвере запись имени файла производится в стандартный буфер EMT 36, поэтому команда MOV #326,R3 перед вызовом JSR PC,@#1046 перемещаемости не нарушает.

- Подпрограмма @#1056: выполняет операцию чтения-записи указанных блоков диска. Перед вызовом в R0 задается номер первого читаемого (или записываемого) блока, в R1 — длина массива информации в двухбайтных машинных словах, а в R2 — начальный адрес в ОЗУ. Вызов производится командой JSR PC,@#1156, после возврата ненулевое значение флага C указывает на ошибку чтения-записи.

Легко заметить, что данная подпрограмма является аналогом стандартной функции чтения-записи блоков, прошитой в ПЗУ контроллера дисководов (@#160004), однако в отличие от последней она использует определяемую программой DESS рабочую область и (под управлением ANDOS или MKDOS) обеспечивает работу с любым «дископодобным» устройством, например с электронным диском через драйвер VDISK.

- Подпрограмма @#1076: опрашивает клавиатуру и возвращает в R0 код нажатой клавиши. Вызов: JSR PC,@#1076.

Данная подпрограмма аналогична стандартной функции EMT 6, но обеспечивает блокировку клавиши «ТАБ» (нажатие которой при использовании обычного EMT 6 приводит к непрерываемому вводу целого ряда пробелов) и преобразует код введенной клавиши в регистр ЛАТ ЗАГЛ (операция BIC #177640,R0).

- Ячейка @#1016: содержит номер текущего блока (того, на котором находился курсор при вызове из DESS подгружаемого драйвера).
- Ячейка @#1024: содержит номер блока, помеченного в DESS с помощью клавиши «СБР».
- Ячейка @#1004: содержит начальный адрес в ОЗУ буфера, выделенного программой DESS для чтения очередного блока.

Кроме перечисленных, в драйвере использована и еще одна, не отмеченная в документации, возможность связи с DESS: в ячейке стека, находящейся над сохраненным при вызове драйвера адресом возврата (в данном случае — в ячейке с адресом SP+2), хранится текущий адрес курсора в буфере чтения блока. Таким образом, пара команд MOV 2(SP),R1 и SUB @#1004,R1 вычисляет длину в байтах фрагмента текущего блока от его начала до курсора DESS.

В драйвере также использованы две подпрограммы, прошитые в ПЗУ монитора БК:

- подпрограмма @#102062: генерация звукового сигнала меняющейся частоты, длительность и исходная высота тона которого заданы в регистрах R2 и R3.;
- подпрограмма @#100602: вызов функции EMT 36 с заданием стандартного адреса буфера (@#320) и последующим контролем на ошибку чтения-записи. Данный адрес представляет собой дополнительную точку входа в подпрограмму монитора, обеспечивающую чтение файла по команде «М».

И наконец, следует отметить нестандартное использование системной ячейки БК @#105: оператор CLRB @#105 служит для сброса кода предыдущей нажатой клавиши — последующая команда EMT 6 всегда будет ждать нажатия новой.

После того как мы разобрались с назначением отдельных подпрограмм и ячеек, алгоритм работы драйвера легко понять из комментариев к листингу:

вначале вычисляется количество блоков, заданных пользователем в DESS для записи, и если оно больше 27 (восьм.), выдается сообщение о невозможности выполнения операции. Данное ограничение вызвано тем, что под информацию, считываемую из блоков и затем записываемую в виде файла, выделяется часть экранного ОЗУ с адресами от 40000 до 67777 (в режиме РП). (Кстати, в исходном варианте драйвера допускалось до 37 (восьм.) блоков, но в данном случае область экрана от 70000 до 77777 используется нами для вывода текстовых сообщений.) Затем, если задано менее 30 блоков, запрашивается имя будущего файла, вычисляется и заносится в буфер EMT 36 его длина (не обязательно целое число секторов !), а содержимое заданных блоков считывается в экранное ОЗУ с адреса 40000. При ошибке чтения выводится запрос: игнорировать ли ошибку. При нажатии клавиши «I» (на любом регистре клавиатуры) все, что удалось считать, будет записано в файл, иначе делается еще одна попытка считать содержимое блоков диска без ошибки. Далее (если ошибки нет или она проигнорирована) на экран выдается приглашение вставить другой диск (двигатель дисковода при этом останавливается) и после нажатия любой клавиши производится запись файла. Затем проверяется байт ответа EMT 36: если при записи была ошибка, после нажатия любой клавиши производится повтор операции, если же ошибки нет (либо операция остановлена самим пользователем путем нажатия клавиши «СТОП»), следует выход в DESS.

Работать с драйвером также весьма просто. Сразу же после запуска DESS нужно нажать клавиши «СУ»+«U», подтвердить команду нажатием на «ВВОД», ввести имя драйвера вместе с расширением .DSS и снова нажать «ВВОД» — драйвер будет подгружен.

Найдя с помощью DESS ряд блоков, содержимое которых нужно записать в файл, установите курсор в начало первого из них и нажмите «СБР», после чего установите курсор на двухбайтное слово, следующее за последним записываемым (либо на начало следующего блока, если предыдущий нужно записать целиком) и нажмите «U». Экран очистится, сверху появится строка-заголовок драйвера и запрос имени файла (либо сообщение, что файл слишком велик, после чего для выхода в DESS нужно нажать любую клавишу).

Введя имя и нажав «ВВОД», дождитесь приглашения сменить диск и вставьте дискету, на которую будут записываться файлы, после чего нажмите любую клавишу. А после выдачи на экран нового приглашения вновь вставьте исходную дискету (содержимое блоков которой нужно спасти) и нажмите любую клавишу для выхода в DESS.

Дополнительно следует отметить, что, поскольку запись файлов производится через EMT 36, эта операция будет успешно производиться в любой дисковой системе с EMT-перехватом, но программа DESS должна быть запущена именно из этой системы. (Например, если нужно восстанавливать дискету формата NORD, а файлы вы хотите записывать на дискету ANDOS, DESS нужно запускать именно из ANDOS, но тогда при просмотре NORDовской дискеты не все функции перемещения по блокам можно использовать, чаще всего допустимы только команда «В» и переход с блока на блок с помощью клавиш управления курсором.) Вообще же желательно целиком работать в какой-то одной системе (лучше всего в формате которой записана восстанавливаемая дискета).

А. В. Суханов,

Москва

Еще немного о VorteX 4.0

В дополнение к статье «VorteX 4.0: похвала и немного критики», опубликованной в журнале «Персональный компьютер БК-0010, БК-0011М», № 6 за 1995 г., хотелось бы сделать ряд дополнений и исправлений на основании личного опыта работы с данной программой.

1. ВОЗМОЖНОСТЬ ЗАПУСКА ИЗ USER-МЕНЮ НЕ ИСЧЕЗЛА! Существует даже возможность автопечати уже подготовленных текстов! Ниже приводится три примера команд USER-меню для ANDOS v3.1. (Напомним, что для редактирования USER-меню надо запустить его с того диска, на котором находится «Vortex», выбрать редактируемый пункт и нажать «БЛОК РЕД». На экране появится редактируемая строка.) Символы в примерах, взятые в круглые скобки, набираются так: нажимаем клавишу «ИНД СУ» — курсор гаснет; нажимаем требуемую управляющую клавишу — появляется ее символ. После окончания редактирования следует нажать «ВВОД», а на запрос «SAVE» еще раз «ВВОД». Если программа «VorteX» записана на диске не под именем «VX», то его после двоеточия в командной строке нужно заменить на требуемое название. Вот примеры команд меню:

1: ЗАПУСК VX:VX(↵).

Происходит простой запуск «VorteX».

2: ЗАГРУЗКА .VXT в VX:VX(↵)(↵)!(↵).

Происходит запуск «VorteX» и открытие файла .VXT, на котором перед вызовом меню находился курсор.

3: ПЕЧАТЬ .VXT из VX:VX(↵)(↵)!(↵)(КТ)(→)(→)(→)(→)(→)(→)(↵).

Происходит запуск «VorteX», открытие файла .VXT, на котором находится курсор, и, если включен принтер, начинается печать текста.

2. Некоторая возможность сохранения всего текста или хотя бы его части при обнаружении поврежденного кластера все же имеется. В то время как программа упорно пытается прочитать или записать сбойный кластер, нужно упорно нажимать «СТОП», пока не появится меню. Теперь выберите пункт «ЗАКРЫТЬ ФАЙЛ» (обычно файл нормально закрывается) и затем откройте файл заново.

3. Управлять принтером, вставляя в текст ESC-коды, конечно, хорошо. Но, поскольку в «Vortex» существует возможность графической печати, лучше было бы предусмотреть в нем «штатную» команду изменения размера букв как для переключения начертания символов (по клавишам «АР2», «НР» и «1»—«6») или подгружаемый шрифт. Вот тогда действительно будет обеспечен режим «WYSIWYG».

4. Замечание к пункту 18 предыдущей статьи о VorteX 4.0. Вообще-то редко кто вставляет таблицы непосредственно внутрь строки, поэтому, чтобы не испортить таблицу, можно использовать форматирование буфера. Выделите буфер, нажмите «ВВОД» и в появившемся меню выберите пункт «ФОРМАТ».

НАЧИНАЮЩИМ ПОЛЬЗОВАТЕЛЯМ

Читатели журнала «Персональный компьютер БК-0010 — БК-0011М» в своих письмах часто задают вопрос: что такое спрайты и как с ними работать? Интерес к этой теме не случаен. Ведь без спрайтов, содержащих изображения персонажей и деталей фона, практически невозможно создать полноценную игру с динамичным сюжетом — их рисование непосредственно во время игры выполнялось бы слишком медленно даже на ассемблере, не говоря уже о БЕЙСИКе и ФОКАЛе. Надеемся, что эта статья поможет начинающим программистам освоить работу со спрайтами.

Д. Ю. Усенков,

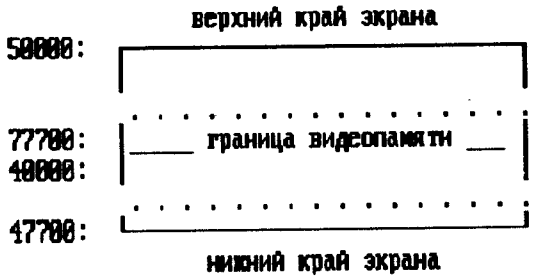
Москва

Что такое спрайт

С английского языка слово «спрайт» переводится как «фея» или «эльф» — должно быть, впервые применивший его в качестве «компьютерного» термина программист обладал довольно-таки романтичным характером. Но в общем-то это изначальное значение слова достаточно верно отражает действительность: компьютерные спрайты представляют собой фрагменты изображения, которые программа может (волею ее создателя) выводить в любое место экрана, так что персонаж игры, подобно сказочной фее, свободно «летает» по экрану. Но вот вопрос: как он это делает?

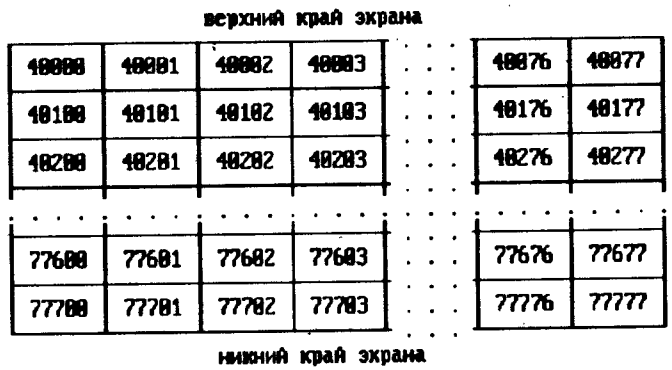
Чтобы ответить на него, нам сначала понадобится вспомнить, как устроена видеопамять компьютера БК (для других типов ЭВМ возможны отличия, но основные механизмы реализации вывода изображения на экран, по сути, остаются теми же).

В БК-0010 под видеопамять выделена область ОЗУ с адресами от 40000_8 до 77777_8 (в режиме РП — от 70000_8 до 77777_8), при этом каждой телевизионной строке соответствует 100_8 байт. При нормализованном рулонном смещении экрана адрес 40000_8 соответствует верхнему левому углу служебной строки, а 77777_8 — нижнему правому углу рабочей области. При наличии же рулонного смещения соответствие видеопамяти полю экрана выглядит несколько сложнее. Прямоугольный массив видеопамяти как бы сдвигается по вертикали относительно экрана вверх, а «вышедшая за его верхний край» часть «переносится» вниз. Наглядно это можно представить в виде следующей диаграммы (адрес 50000_8 взят для примера, конкретное его значение зависит от содержимого ячейки 177664_8):



Массив видеопамяти здесь можно уподобить разрезанному по горизонтали листу бумаги, в котором верхняя часть подклеена вплотную к нижней. (Показанная на рисунке «граница видеопамяти» соответствует переходу от старших адресов видеоОЗУ к младшим при движении вниз по экрану.)

Впрочем, учет рулонного смещения не меняет сути алгоритмов вывода информации на экран, требуя лишь контроля перехода через «границу видеопамяти». (Например при движении вниз, что соответствует увеличению адреса байта экранного ОЗУ на 100_8, нужно сравнить полученную сумму с константой 100000_8 и если она больше, вычесть из нее число 40000_8.) В программах, использующих спрайты, разработчики стараются иметь дело с нормализованным рулонным смещением, поэтому далее мы будем считать, что массив видеопамяти полностью соответствует полю экрана (числа в таблице означают восьмеричные адреса байтов видеопамяти):



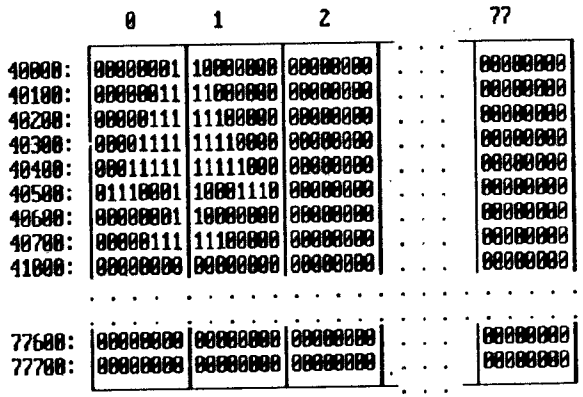
Теперь посмотрим, как кодируется графическое изображение в байтах видеопамяти. В режиме 64 символа в строке каждой графической точке соответствует один бит экранного ОЗУ, причем «1» в этом бите соответствует высвеченной точке (белый цвет), а «0» — погашенной (черный). В режиме 32 символа в строке кодирование выполняется аналогично, но каждой точке на экране соответствует уже пара битов: «11» — красный цвет, «10» — зеленый, «01» — синий, «00» — черный.

красная		черная		синяя		зеленая		← расположение точек (цветной режим)
■	■			■			■	← расположение точек (черно-белый режим)
1	1	0	0	1	0	0	1	← содержимое битов байта
0	1	2	3	4	5	6	7	← номера разрядов байта

На черно-белом дисплее даже в режиме 32 символа в строке изображение фактически строится из «однобитовых» точек, но выводятся они на экран парами. Поэтому красный цвет выглядит как белый, а зеленый и синий — как серый одинаковой плотности, причем на границе между ними появляется белая или черная полоска.

Обратите внимание, что нумерация битов в байте экранного ОЗУ производится от 0 до 7 **слева направо** (в книгах, где описывается структура байта, нумерация обычно показывается справа налево). Поэтому кодирование точек синего и зеленого цветов на данном рисунке получилось «зеркальным» относительно привычного («01» и «10» вместо «10» и «01», соответственно). Тогда нулевой бит байта с адресом 40000_8 соответствует в режиме 64 символа в строке верхней левой точке служебной строки (рулонное смещение мы считаем нормализованным), а седьмой бит байта с адресом 77777_8 — точке в нижнем правом углу рабочей области экрана.

Таким образом, любому изображению на экране соответствует определенное содержимое байтов в соответствующей ему прямоугольной области экранного ОЗУ (суммы чисел слева и сверху дают адреса байтов, причем левые числа указывают адреса начала графических строк экрана):



Здесь показано содержимое экранного ОЗУ (в двоичной форме), когда на пустом экране в верхнем левом углу нарисован «самолетик» (черно-белый режим; аналогичная картина наблюдается и при выводе цветного рисунка).

А что будет, если мы с помощью команд MOVB (или MOV, если рисунок занимает пары байтов, составляющие целые машинные слова, как и в нашем

случае) скопируем содержимое данного фрагмента экранного ОЗУ в другое место экрана, сохранив порядок расположения байтов? Очевидно, в соответствующем второму фрагменту видеопамяти месте появится точно такой же «самолетик»:

```
          0        1        2                    77
40000: 00000001 10000000 00000000  . . .  00000000
40100: 00000011 11000000 00000000  . . .  00000000
40200: 00000111 11100000 00000000  . . .  00000000
40300: 00001111 11110000 00000000  . . .  00000000
40400: 00011111 11111000 00000000  . . .  00000000
40500: 01110001 10001110 00000000  . . .  00000000
40600: 00000001 10000000 00000000  . . .  00000000
40700: 00000111 11100000 00000000  . . .  00000000
41000: 00000000 00000000 00000000  . . .  00000000
       . . . . . . . . . . . . . . . . . . . . . .
       . . . . . . . . . . . . . . . . . . . . . .
76600: 00000000 00000000 00000000  . . .  00000000
76700: 00000000 00000001 10000000  . . .  00000000
77000: 00000000 00000011 11000000  . . .  00000000
77100: 00000000 00000111 11100000  . . .  00000000
77200: 00000000 00001111 11110000  . . .  00000000
77300: 00000000 00011111 11111000  . . .  00000000
77400: 00000000 01110001 10001110  . . .  00000000
77500: 00000000 00000001 10000000  . . .  00000000
77600: 00000000 00000111 11100000  . . .  00000000
77700: 00000000 00000000 00000000  . . .  00000000
```

Выполненное нами действие аналогично цепочке команд:

```
MOVB @#40000,@#76701
MOVB @#40001,@#76702
MOVB @#40100,@#77001
MOVB @#40101,@#77002
 . . . . . .
MOVB @#40700,@#77601
MOVB @#40701,@#77602
```

Таким способом мы можем заполнить хоть весь экран копиями первоначально нарисованного «самолетика». Только где взять этот исходный рисунок?

Точно так же, как мы только что делали в пределах видеоОЗУ, можно переписать заранее нарисованный «самолетик» в ОЗУ пользователя, пристыковав его к кодам игровой программы, а в самой игре — скопировать из пользовательского ОЗУ в нужное место экрана. Единственное, на что при этом нужно обратить внимание, — это размещение кодирующих изображение байтов в пользовательском ОЗУ. Чтобы не расходовать память на хранение пустых участков экрана слева и справа, мы будем переписывать картинку построчно (считаем, что копия картинки размещается в пользовательском ОЗУ начиная с адреса 30000_8):


```
30000: 00000001 10000000
30002: 00000011 11000000
30004: 00000111 11100000
30006: 00001111 11110000
30010: 00011111 11111000
30012: 01110001 10001110
30014: 00000001 10000000
30016: 00000111 11100000
```

Или, если переписать содержимое ОЗУ в привычном виде «потока байтов», получим:

```
00000001 10000000 00000011 11000000 00000111 11100000
00001111 11110000 00011111 11111000 01110001 10001110
00000001 10000000 00000111 11100000
```

Не правда ли, получается очень просто? А ведь все это время мы, сами того не замечая, работали с самым настоящим спрайтом! Да-да, то, что мы называли «прямоугольным фрагментом экранного ОЗУ» (точнее, содержимое байтов этого фрагмента), как раз и есть спрайт. Осталось только обобщить наши рассуждения и перейти от конкретного «самолетика» к спрайтам произвольного размера.

Пусть мы хотим создать спрайт шириной X байтов и высотой Y графических строк (содержимое байтов условно показано как «***»):

```
      1   2   3  ... X-1  X
   1  *** *** *** ... *** ***
   2  *** *** *** ... *** ***
   3  *** *** *** ... *** ***
 ...  ... ... ... ... ... ...
 Y-1  *** *** *** ... *** ***
   Y  *** *** *** ... *** ***
```

Тогда мы должны записать в пользовательское ОЗУ количество байтов, равное произведению X*Y:

```
<a. б.>: (1,1) (1,2) ... (1,X) (2,1) (2,2) ... (2,X) ...
         (Y,1) (Y,2) ... (Y,X)
```

Здесь **<a. б.>** — адрес начала выделенной под хранение спрайта области пользовательского ОЗУ (ее обычно называют буфером), а записи **(i,j)** означают байты спрайта (i=1...Y и j=1...X — номера его строк и столбцов соответственно).

При выводе на экран нужно превратить «потоковую» запись спрайта в нормальную прямоугольную, следовательно, алгоритм программы будет таким.

1. Задание исходных данных. Пусть в R1 записан адрес начала буфера спрайта в пользовательском ОЗУ, в R2 — адрес верхнего левого байта места в экранном ОЗУ (куда предполагается вывести спрайт), R3 — ширина спрайта (X байт), R4 — его высота (Y графических строк).
2. Вывод первой строчки спрайта:

```
      MOV R3,R0          ; скопировать ширину в буферный регистр
      MOVB (R1)+,(R2)+   ; вывести очередной байт
      SOB R0,            ; цикл вывода всех X байтов строчки
```

3. Переход на следующую графическую строку экрана, чтобы вывести вторую строчку спрайта под первой:

```
ADD #100,R2        ; спуститься ниже на 1 граф. строку
SUB R3,R2          ; сместиться левее под начало
                   ;  предыдущей строки
```

4. Повтор пунктов 2 и 3 для вывода второй, третьей и т. д. строчек (всего Y раз), что также можно реализовать в виде цикла SOB с использованием регистра R4.

Теперь несложно написать ассемблерную подпрограмму вывода спрайта на экран:

```
;          Вывод спрайта
; Исходные данные: R1 - адрес начала буфера;
;                  R2 - адрес в видеоОЗУ;
;                  R3 - ширина спрайта (байт);
;                  R4 - высота спрайта (строк).
SPROUT: MOV R3,R0           ; цикл вывода
1:      MOVB (R1)+,(R2)+ ;#;  очередной
        SOB R0,1            ;   строки спрайта
        ADD #100,R2         ; переход на
        SUB R3,R2           ;  следующую строку
        CMP R2,#100000   ;*; проверка перехода
        BLO 2            ;*;  через границу видеоОЗУ
        SUB #40000,R2    ;*;   при ненормализованном рулоне
2:      SOB R4,SPROUT       ; цикл вывода строк спрайта
        RTS PC              ; выход
```

Примечания.

1. В этой программе не предусмотрен контроль выхода спрайта за правую границу экрана, т. е. пользователь сам должен следить, чтобы значения R2 и R3 удовлетворяли условию: (R2 MOD 100_8) + R3 < 100_8 .

2. Нужно также следить, чтобы спрайт не выходил за верхний и нижний края экрана, т. е. должны выполняться условия: R2 ⩾ 40000_8 и R2+R4 < 100000_8 .

3. Если рулонное смещение экрана при выводе спрайта нормализовано, строки, помеченные в комментариях знаком «*», можно удалить.

4. В данном варианте подпрограммы фон спрайта «непрозрачен» и при выводе спрайта закрывает собой фон экрана. Если же заменить строку, отмеченную в комментарии знаком «#», на **BISB (R1)+,(R2)+** , то сквозь фоновые (черные) участки черно-белого спрайта при его выводе на экран будет просвечивать прежний фон.

С помощью этой подпрограммы можно вывести спрайт в любое место экрана и даже получить эффект движения картинки по экрану — достаточно вызывать подпрограмму в цикле с изменением значения R2 с определенным шагом (увеличение/уменьшение R2 с шагом, кратным 1, соответствует движению вправо/влево, а с кратным 100_8 — вниз/вверх). Если же заготовить два и более спрайтов, изображающих последовательные фазы движений персонажа игры, и одновременно с изменением R2 выводить их по порядку (циклически «переключая» значение R1), получится простейший мультфильм.

Вот вроде бы и все. «Но постойте! — воскликнет внимательный читатель. — Если мы будем просто вызывать нашу подпрограмму в цикле, меняя R2, то на экране появится не один движущийся спрайт, а целый ряд неподвижных!»

Действительно, прежде чем выводить спрайт в следующую позицию экрана, надо каким-то образом убрать с него предыдущий. В простейшем случае (когда остальная часть экрана пуста) можно стереть спрайт, «залив» его прямоугольный участок цветом фона. (Подпрограмма стирания будет отличаться от уже рассмотренной только одной деталью: в строке, отмеченной знаком «#», должна стоять команда **MOVB #<цвет>,(R2)+** или, для черного цвета, **CLRB (R2)+** .) Но если сюжет игры предполагает движение персонажа по сложному рисованному фону, то стирающая подпрограмма неминуемо испортит фон. Поэтому реально применяется другой способ: в пользовательской памяти выделяют еще один дополнительный буфер того же объема, что и для спрайта, и ПЕРЕД выводом этого спрайта в дополнительном буфере сохраняют соответствующий прямоугольный участок фона (тоже как спрайт). Тогда для УДАЛЕНИЯ спрайта с экрана достаточно вывести на его место сохраненный ранее участок фона. Подпрограмма же, сохраняющая фон в буфере, будет отличаться от рассмотренной выше только одной строкой, помеченной «#», — в ней должна стоять команда **MOVB (R2)+,(R1)+**

И вот теперь мы можем, наконец, написать программу движения спрайта по экрану со сложным фоном. В ней благодаря эффекту самомодификации используется всего одна спрайтовая подпрограмма (в R5 при вызове передается рабочая команда: MOVB (R1)+,(R2)+ (код 112122_8, вывод спрайта), MOVB (R2)+,(R1)+ (код 112221_8, сохранение спрайта в буфере) или BICB (R1)+,(R2)+ (код 152122_8, вывод спрайта с «прозрачным» фоном; читатели могут поэкспериментировать, подставляя другие однословные команды ассемблера с аналогичным использованием регистров).

Кодирование спрайтов в нашей программе производится непосредственно — расстановкой единичных битов, образующих рисунок, в записях двоичных чисел в комментариях и их ручным переводом в восьмеричную форму (обратите внимание: при перекодировании в восьмеричный формат берется зеркальное отображение двоичной записи!). При написании «настоящих» игр все необходимые спрайты обычно заранее рисуют в специальных программах — спрайтовых редакторах, которые автоматически преобразуют изображения в соответствующие им наборы байтов. Затем последние в виде кодовых модулей добавляются к создаваемому ассемблерному листингу (как ряд команд .#) или пристыковываются к создаваемой программе уже после трансляции в коды с помощью отладчика или «линкера».

После компиляции приведенного ниже листинга в трансляторе типа М18 (или аналогичном) и его запуска на выполнение на экране появляется «кирпичная стенка» и забавный паучок, который «ползает» по стене при нажатии клавиш управления курсором (заметьте: фон при этом не портится!). Выход из программы — по клавише «КТ».

```
;============================================================
;                        Паучок Ананси
;    Клавиши управления:
;    курсорные стрелки  - перемещение паучка;
;    "КТ"               - выход из программы.
;------------------------------------------------------------
START:  EMT 14          ; нормализация режимов вывода на экран
        MOV #40000,R2   ; нач. адрес вывода кирпича
        MOV #40,R1      ; всего сорок рядов
1:      MOV R2,-(SP)    ; адрес начала ряда -> в стек
        MOV #20,R0      ;  из 20 кирпичей в каждом
2:      JSR R4,@#110346 ; регистры в стек
        MOV #STOUN,R1   ; буфер с изображением кирпича
        MOV #4,R3       ; ширина кирпича - 4 байта
        MOV #10,R4      ; высота кирпича - 10 (восьм.) строк
        MOV #112122,R5  ; команда "вывод спрайта"
        JSR PC,SPRITE   ; вывод кирпича
        JSR R4,@#110362 ; регистры <- из стека
        ADD #4,R2       ; следующий кирпич
        SOB R0,2        ; цикл вывода ряда кирпичей
        MOV (SP)+,R2    ; адрес начала ряда <- из стека
        ADD #1000,R2    ; следующий ряд
        SOB R1,1        ; цикл вывода рядов
;
; фон нарисован, можно выпускать паучка
;
        MOV #50040,APAU ; адрес вывода спрайта с паучком
ANANSI: MOV APAU,R2     ; адрес в видеоОЗУ для вывода паучка
        MOV #4,R3       ; ширина спрайта
        JSR R4,@#110346 ; параметры области спрайта -> в стек
        MOV #20,R4      ; высота спрайта
        MOV #FON,R1     ; буфер для сохранения фона
        MOV #112221,R5  ; команда "сохранение фона"
        JSR PC,SPRITE   ;
        JSR R4,@#110362 ; восстановить параметры области
        MOV #20,R4      ; высота спрайта
        MOV #PAU,R1     ; адрес буфера с паучком
        MOV #152122,R5  ; команда "вывод спрайта с прозрачным
        JSR PC,SPRITE   ;  фоном"
;
; паучок выведен - ждем команду пользователя
;
        BIS #100,@#177660 ; запрет прерывания от клавиатуры
INKEY:  MOV #20000,R5   ; константа задержки
11:     SOB R5,11       ; задержка
        BIT #200,@#177660 ; проверка нажатия клавиши
        BNE 1           ;
        BIT #100,@#177716 ; прежняя клавиша еще нажата?
        BEQ 2           ; да - сохранить прежний код
        BR INKEY        ; нет, и не нажималась - ждать нажатия
1:      MOV #7,R0       ; звуковой сигнал
        EMT 16          ;  нажатия на клавишу
        MOV @#177662,R0 ; код клавиши -> в R0
        BIC #177400,R0  ; очистить старший байт
2:      CMPB R0,#3      ; клавиша "КТ"?
        BNE 3
        EMT 14          ; да - реинициализация
        HALT            ;  и выход из программы
```



```
;
; если не выход из программы, то убираем паучка
;
3:      MOV #FON,R1     ; вывод вместо паучка
        MOV APAU,R2     ;  сохраненного ранее
        MOV #4,R3       ;  участка фона
        MOV #20,R4      ;
        MOV #112122,R5  ;
        JSR PC,SPRITE   ;
;
; теперь проверяем, куда сместить паучка
;
        CMPB R0,#10     ; клавиша "ВЛЕВО"?
        BNE 4
        JSR PC,LEFT     ; смещение паучка влево
        BR ANANSI       ; и выводить паучка
4:      CMPB R0,#31     ; клавиша "ВПРАВО"?
        BNE 5
        JSR PC,RIGHT    ; смещение паучка вправо
        BR ANANSI
5:      CMPB R0,#32     ; клавиша "ВВЕРХ"?
        BNE 6
        JSR PC,UP       ; смещение паучка вверх
        BR ANANSI
6:      CMPB R0,#33     ; клавиша "ВНИЗ"?
        BNE ANANSI      ; любая другая - выводить паучка там же
        JSR PC,DOUN     ; смещение паучка вниз
        BR ANANSI
;
XPOZ:   MOV APAU,R1     ; выделение номера позиции
        BIC #177700,R1  ;  паучка в строке
        RTS PC
;
YPOZ:   MOV APAU,R1     ; выделение номера строки
        BIC #77,R1      ;
        RTS PC
;
LEFT:   JSR PC,XPOZ     ;
        TST R1          ; если паучок и так у левого края,
        BEQ RETL        ;  не смещать его
        SUB #2,APAU     ; иначе уменьшить адрес на 2
RETL:   RTS PC
;
RIGHT:  JSR PC,XPOZ     ;
        CMP R1,#74      ; если паучок и так у правого края,
        BEQ RETR        ;  не смещать его
        ADD #2,APAU     ; иначе увеличить адрес на 2
RETR:   RTS PC
;
UP:     JSR PC,YPOZ     ;
        CMP R1,#40000   ; если паучок и так сверху,
        BEQ RETU        ;  не смещать его
        SUB #400,APAU   ; иначе уменьшить адрес на 4*100
RETU:   RTS PC
;
DOUN:   JSR PC,YPOZ     ;
        CMP R1,#76000   ; если паучок и так внизу,
        BEQ RETD        ;  не смещать его
```

```
        ADD #400,APAU      ; иначе увеличить адрес на 4*100
RETD:   RTS PC
;
APAU:   .#0                ; буфер адреса паучка
; ---------------------------------------------------------------
;       Универсальный самомодифицирующийся спрайтовый драйвер
;  Исходные данные: R1 - адрес буфера; R2 - адрес экрана;
;                   R3 - ширина;      R4 - высота;
;                   R5 - код операции:
; =============          112122 - вывод спрайта;
;   SCREW (c)            112221 - сохранение фона;
; =============          152122 - вывод спрайта с наложением.
; ---------------------------------------------------------------
SPRITE: MOV R0,-(SP)       ; сохранить "буферный" регистр
        MOV R5,KOMAND      ; подставить код операции на место
LOOP:   MOV R3,R0          ; копирование ширины в R0
KOMAND: .#0                ; место для кода операции
        SOB R0,KOMAND      ; цикл вывода очередной строчки спрайта
        ADD #100,R2        ; смещение на следующую строчку
        SUB R3,R2          ;  (вниз и влево под начало предыдущей)
        SOB R4,LOOP        ; цикл вывода строчек спрайта
        MOV (SP)+,R0       ; восстановить "буферный" регистр
        RTS PC             ; возврат в основную программу
; ---------------------------------------------------------------
;
; буфер изображения кирпича
STOUN:  .#125252.#125252 ; 10101010 10101010 10101010 10101010
        .#100001.#100001 ; 10000000 00000001 10000000 00000001
        .#100001.#100001 ; 10000000 00000001 10000000 00000001
        .#125252.#125252 ; 10101010 10101010 10101010 10101010
        .#125252.#125252 ; 10101010 10101010 10101010 10101010
        .#000600.#000600 ; 00000001 10000000 00000001 10000000
        .#000600.#000600 ; 00000001 10000000 00000001 10000000
        .#125252.#125252 ; 10101010 10101010 10101010 10101010
        .E
; буфер изображения паучка
PAU:    .#000000.#000000 ; 00000000 00000000 00000000 00000000
        .#000000.#000000 ; 00000000 00000000 00000000 00000000
        .#140000.#000003 ; 00000000 00000011 11000000 00000000
        .#170000.#000017 ; 00000000 00001111 11110000 00000000
        .#170360.#007417 ; 00001111 00001111 11110000 11110000
        .#141400.#000303 ; 00000000 11000011 11000011 00000000
        .#146000.#000063 ; 00000000 00110011 11001100 00000000
        .#170360.#007417 ; 00001111 00001111 11110000 11110000
        .#171414.#030317 ; 00110000 11001111 11110011 00001100
        .#176000.#000077 ; 00000000 00111111 11111100 00000000
        .#176000.#000077 ; 00000000 00111111 11111100 00000000
        .#171400.#000317 ; 00000000 11001111 11110011 00000000
        .#170300.#001417 ; 00000011 00001111 11110000 11000000
        .#140074.#036003 ; 00111100 00000011 11000000 00111100
        .#000000.#000000 ; 00000000 00000000 00000000 00000000
        .#000000.#000000 ; 00000000 00000000 00000000 00000000
        .E
; буфер для сохранения фона
FON:    .+100              ; резервируем 4*20=100 байт
        .E
;
; ========================================= That's end! =====
```


Приложение. Подстыковка спрайтов к создаваемым ассемблерным программам

Как выяснилось из бесед с читателями журнала, многие начинающие программисты испытывают трудности при подстыковке уже созданных (например, в ANIMATICe) спрайтов к собственным программам на ассемблере. Те же сложности возникают и при необходимости подстыковки к программе автономных кодовых фрагментов (например исполняемых подпрограмм воспроизведения мелодий из KLAWESINa или MAESTRO).

Наиболее примитивный способ «сборки» готовой программной разработки состоит в резервировании в создаваемой программе ячеек под начальные адреса подстыковываемых спрайтов, считывании ее (уже после трансляции в коды) в отладчик, подгрузке спрайтов одного за другим с диска в ОЗУ (пользователь должен указывать такие начальные адреса загрузки, чтобы спрайты не затирали друг друга и основную программу) и занесении адресов их начала вручную в зарезервированные ранее ячейки. В некоторых случаях аналогичным способом (резервируя байты в ассемблер-трансляторе и записывая реальные значения в отладчике) приходится сообщать программе значения ширины и высоты каждого спрайта.

Разумеется, такой труд нельзя считать особо интересным, поэтому многие программисты стараются тем или иным способом облегчить этот процесс. При этом возможны два направления поиска: создание программ для автоматической подгрузки автономных модулей к уже оттранслированной программе и преобразование кодовых вставок в текстовые записи, пристыковываемые к ассемблерному листингу создаваемой программы и затем транслируемые вместе с ней.

Разработкой первого типа является программа SCREW LINKER (ее описание предполагается опубликовать в одном из следующих выпусков журнала), которая автоматизирует отслеживание начальных адресов подгружаемых автономных модулей и их подстановку в зарезервированные в основной программе ячейки.

Если же требуется подстыковка всего одного-двух спрайтов, удобнее пользоваться вторым из названных выше методов. Небольшая программа в машинных кодах, листинг которой приводится ниже (она была разработана несколько лет назад В. Кобяковым под ОС NORD и модифицирована автором данной статьи для работы в ANDOS), преобразует содержащийся в отдельном файле спрайт (или любой другой кодовый фрагмент) в последовательность команд **.#<восьм. число>**. Таким образом формируется текстовое представление кодов спрайта в стандарте транслятора серии М18, которое можно подгрузить к создаваемой ассемблерной программе для совместной компиляции или транслировать отдельно и прилинковать к программе в виде .OBJ-модуля.

Работать с перекодировщиком несложно. После запуска выводится текстовая подсказка о его назначении и запрос имени исходного (кодового) файла. После ввода имени и нажатия клавиши «ВВОД» (в ANDOS 3.1 можно воспользоваться встроенным файлером) выполняется перекодирование и запись результирующего файла (имя прежнее, расширение .WDS). По окончании записи нужно нажать «ВВОД» для выхода в файловую оболочку DiskMaster.

Начальный адрес: 001000 Длина: 001300

010046	010146	010246	012702	000320	012722	000003	010712
062722	001252	005022	012722	020040	020227	000400	103773
010701	062701	000302	005002	104020	012701	000326	012702
005020	104010	112741	000040	012701	000320	104036	105737
000321	001341	013700	000350	006200	010027	000000	010700
062700	001152	016701	177764	010002	060102	060102	010267
000176	005067	000174	010705	062705	000564	105715	001404
112504	004767	000764	000772	012767	000007	000752	012003
004767	000756	004767	001002	005301	001406	005367	000730
001366	004767	000752	000760	012737	000002	000320	016737
000076	000322	016737	000072	000324	012701	000326	122711
000056	001404	005201	020127	000336	103771	112721	000056
112721	000127	112721	000104	112721	000123	012701	000320
104036	005003	153703	000321	012602	012601	012600	000207
000000	000000	106214	125012	132665	132665	132665	132665
132665	132665	024265	024503	050126	023522	031471	132665
132665	132665	132665	132665	132665	132665	132665	132665
132665	005243	020267	020040	020040	062760	062560	067727
020304	020327	147315	146545	147157	145711	020171	144724
060720	027040	020043	027040	020043	020040	020040	133440
133412	020040	147040	141145	146157	155730	145157	150040
067560	070307	146541	154715	153440	146440	155541	147311
154716	020170	067713	060704	020170	020040	005267	020267
020040	020040	020040	020040	020040	020040	067524	067527
062162	030563	030056	020040	020040	020040	020040	020040
020040	020040	020040	020040	133440	123012	132665	132665
132665	132665	132665	132665	132665	132665	132665	132665
132665	132665	132665	132665	132665	132665	132665	132665
132665	132665	132665	005271	041040	062727	144704	062724
144440	150715	143040	145141	060714	000072	125073	132665
132665	132665	132665	132665	132665	132665	132665	132665
132665	132665	132665	132665	132665	132665	132665	132665
132665	132665	132665	132665	005243	133473	050040	155145
146171	152330	152141	070040	141141	152157	020331	070320
143557	060560	146715	020331	067524	067527	062162	020163
030566	030056	020060	005267	123073	132665	132665	132665
132665	132665	132665	132665	132665	132665	132665	132665
132665	132665	132665	132665	132665	132665	132665	132665
132665	132665	005271	000000	000000	110422	005267	177162
000207	112704	000056	004767	177760	112704	000043	000402
012704	000012	004767	177742	000207	010046	010146	010346
012700	000006	005001	000405	005001	006103	006101	006103
006101	006103	006101	062701	000060	010104	004767	177672
077015	012603	012601	012600	000207	000000	000000	000000

Контрольная сумма: 104661

Примечание. Данная программа неперемещаема и может обрабатывать кодовые фрагменты ограниченной длины (порядка нескольких килобайт), так как текстовое представление двухбайтного восьмеричного числа вместе с префиксом «.#» занимает в четыре раза больший объем памяти.

СОДЕРЖАНИЕ

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР БК-0010 – БК-0011М

Главный редактор
Васильев Б. М.

Редактор
Усенков Д. Ю.

Корректор
Антонова В. С.

Компьютерная верстка
Иванова Т. А.

Наш адрес:
Москва, м. «Охотный ряд», ул.Тверская, д. 5/6, помещение Международной Академии информатизации, комн. 308

Адрес для переписки:
125315, Москва, а/я 17

Телефон:
(095) 292-53-87

E-Mail (RELCOM):
mail@infoobr.msk.su

Указанный выше адрес для переписки действителен до 1.06.96 г. Новый почтовый адрес будет опубликован в одном из следующих выпусков.

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР БК-0010 — БК-0011М

Подписано в печать с оригинал-макета издательства «Компьютика» 27.04.96.
Формат 60×84 1/16. Бумага офсетная.
Усл.печ.л. 6,5. Заказ № 47
Цена 6500 руб. (по подписке).
В розничной продаже цена договорная.

Отпечатано в типографии «Принт-Сервис»

Планируемое содержание выпуска 3'96:

- AUGUSTA — новый язык программирования для БК
- Микропроцессор К1801ВМ1: устройство и техническое описание
- Описание драйверно-мониторной системы БК-0011
- Переделка БК-0011 в БК-0011М
- Блок питания принтера МС6313
- Многофункциональный АЦП для БК
- Принтер D100: руководство пользователя

Редакция журнала «Персональный компьютер БК-0010 — БК-0011М» с целью изучения покупательского спроса начинает прием от читателей предварительных заказов на брошюру «Ассемблерный листинг и описание работы драйверно-мониторной системы компьютера БК-0010(.01)». Ориентировочная стоимость издания — 10000 руб. (без учета почтовой пересылки), приблизительный срок выхода из печати (при наличии достаточного количества заказов) — декабрь 1996 г.

Заказы на брошюру
можно присылать по адресу:
125315, Москва, а/я 17
или оформить лично в помещении редакции.

Вниманию всех пользователей издательской системы БК-Page Maker

В редакции имеется исправленная версия программы BPM, подготовленная по заказу издательства «Компьютика» ее автором М. И. Кондратовичем. Владельцы прежней версии могут получить исправленный файл в редакции бесплатно или заказать по почте, уплатив только 12000руб. (чистая дискета, пересылка бандероли и орграсходы).