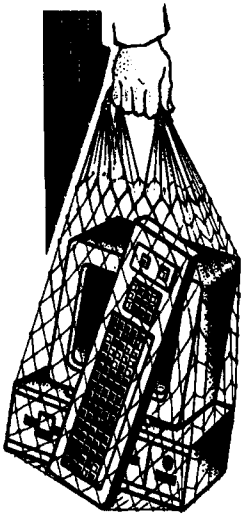
УВАЖАЕМЫЕ ЧИТАТЕЛИ!

У нас в редакции вы можете приобрести отдельные выпуски журналов «Персональный компьютер БК-0010 — БК-0011М», «Персональный компьютер УКНЦ» и «Информатика и образование». Здесь же можно оформить полугодовую подписку на все перечисленные издания с получением экземпляров лично в редакции или по почте. У нас вы также можете приобрести различное программное и аппаратное обеспечение для IBM, БК и УКНЦ.



ПРОГРАММНОЕ И АППАРАТНОЕ
ОБЕСПЕЧЕНИЕ ДЛЯ БК-0010(.01)
И БК-0011(М)

ИНФО

...ммматор ППЗУ.

Мультирежимный контроллер НГМД фирмы АЛЬТЕК для БК-0010(.01).

Усовершенствованный монитор среды БК-0010 для БК-0011(М) с возмож-
ностью реализации графики четкостью не хуже SVGA на IBM-совместимых.

Дисковая операционная сис... ...OS v3.1

База данных, работающа... ...DOS - BASIS 2.10

Мощный графический р... ...1(М) Techno Art

Дисковая операционная... ...БК-0010(.01) и БК-0011(М).

Профессиональный про... ...копий экрана БК-0010 дл
переноса графики на IBM.

Конвертор листингов н... ...ЙСИКе в текстовые файл
формата EDASP и обратно.

Конвертор программ из вну... ...формата ФОКАЛа в листинг
стандарта EDASP.

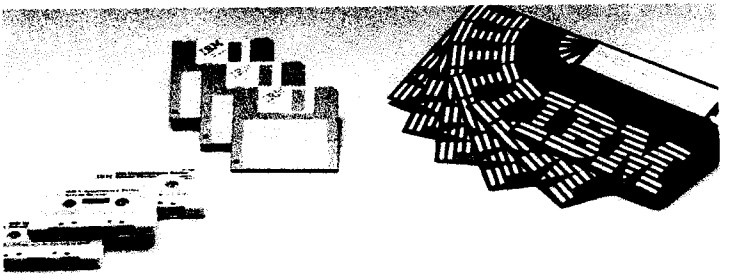
Резидентный драйвер ANIRA... ...RIADA, дополняющий ОС ANDO
следующими возможностями:

— копирование экрана в файл...
— сопровождение диалога с Б... ...ринтере;
— поддержка дополнительног... ...ойства «P:» — «принтер».

Новые игры для БК-0011М: MA... ...AN, TOWER, DIZZY.

Широкий набор драйверов и ути... ...(в том числе для принтера D100
и многое другое.

Заключаем с авторами договоры на рекламу и коммерческое распространение программных и аппаратных разработок. Приглашаем к сотрудничеству книготорговые организации, фирмы и заинтересованных лиц для реализации нашей печатной продукции.



Телефон: 151-19-40
Факс: 208-67-37
E-Mail: mail@infoobr.msk.su

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР БК-0010 БК-0011м

Приложение
к журналу
«ИНФОРМАТИКА
И ОБРАЗОВАНИЕ»

Выпуск 5'94 (6)

Издается с 1993 г.

В НОМЕРЕ

Москва «Информатика и образование» 1994

Авторы выпуска

Гречихин А.
Денсель В. В.
Зальцман Ю. А.
Калмыков Ю. П.
Кузьмич А. В.
Лавриков Д. Е.
Лобанов А. Ю.
Надежин А. М.
Румянцев С. К.
Сандлер М.
Усенков Д. Ю.
Щекунов В. Л.

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

«Библиотека журнала «Информатика и образование»
Свидетельство о регистрации средства массовой информации № 0110336
от 26 февраля 1993 г.

Телефон: (095) 151-19-40, 208-30-78
E-Mail: mail@infoobr.msk.su
Факс: (095) 208-67-37

Продолжаем публикацию неформального учебника по программированию на ассемблере для начинающих пользователей БК-0010(.01). Первые уроки этого компьютерного языка см. в № 1 — № 4 за 1994 г.

Ю. А. Зальцман,
г. Алма-Ата

МикроЭВМ БК-0010. Архитектура и программирование на языке ассемблера

В этой части мы закончим рассмотрение ЕМТ-функций БК-0010 и их роли в программировании на ассемблере.

- ЕМТ 10 — ВВОД СТРОКИ СИМВОЛОВ С КЛАВИАТУРЫ. Данная команда позволяет вводить с клавиатуры в заданную область ОЗУ последовательность кодов символов (а также командные коды). Адрес начала области ОЗУ, в которую осуществляется ввод, задается в R1, ограничители строки — в R2: младший байт R2 — длина строки в байтах (нуль соответствует 200000 байт), старший — код символа-ограничителя. Работа ЕМТ 10 заканчивается после удовлетворения одного из двух условий: строка достигает заданной длины или вводится символ-ограничитель, причем его код заносится в строку последним. По окончании ввода в R1 возвращается адрес байта, следующего за концом строки, в R2 — разность заданной длины и реально введенного количества символов. Ошибочно введенные знаки можно удалять клавишей «ЗАБОЙ». ЕМТ 10 широко применяется для ввода текстов, чисел в символьном виде для последующей их обработки, имен файлов и т.п.
- ЕМТ 20 — ВЫВОД СТРОКИ СИМВОЛОВ НА ЭКРАН. Эта команда выполняет действие, противоположное ЕМТ 10. Адрес начала строки символов задается в R1, ограничители строки — в R2 (так же, как для ЕМТ 10, младший байт — длина строки, 0 соответствует 200000, старший — код символа-ограничителя, причем он выводится на экран последним). С помощью ЕМТ 20 можно не только выводить символы, но и изменять режимы вывода (очистка экрана, переключение формата 32/64 символа в строке и т. п.), если коды соответствующих команд присутствуют в строке. Применяется очень широко для вывода текстовых сообщений. Примеры:

1)

```
0:    EMT 6              Пауза до нажатия клавиши
      MOV #TEX,R1        Адрес начала текста
      MOVB #14,(R1)+     Команда очистки экрана
      MOVB 'П',(R1)+     Запись в ОЗУ
      MOV 'ри',(R1)+     строки символов -
      MOV 'ве',(R1)+     слова "Привет"
      MOVB 'т',(R1)+
      CLRB (R1)+         Нулевой байт - конец текста
      MOV #TEX,R1        Вывод на экран строки текста,
      CLR R2             с метки TEX
```

```
        EMT 20          до нулевого байта
        BR 0            Программа зациклена
TEX:    END
```

2)
```
0:      MOV #TEX,R1     Ввод строки текста (максимальная
        MOV #5377,R2    длина 256д байт, символ-
        EMT 10          ограничитель - код 12 "ВВОД")
        MOV #TEX,R1     и ёе последующая выдача на
        MOV #5377,R2    экран. Буфер строки начинается
        EMT 20          с метки TEX
        BR 0            Программа зациклена
TEX:    END
```

• EMT 22 — ЗАПИСЬ СИМВОЛА В СЛУЖЕБНУЮ СТРОКУ. Позволяет вывести в служебную строку необходимые сообщения, например фамилию автора, название программы и т.п. Информация выводится по одному символу (соответственно, для вывода строки нужно использовать цикл), код этого символа записывается в R0, а номер позиции в служебной строке — в R1. Символы выводятся как в режиме БЛРИСУ, т.е. коды управления (10, 22, 30 и т.п.) отображаются в виде стрелок. Нулевое содержимое R0 означает очистку служебной строки. Позиция отсчитывается от левого края, первому символу соответствует 0. При использовании команды необходимо учитывать, что в правой части служебной строки ЭВМ сама выводит сообщения при переключении режимов, при этом текст, выведенный туда пользователем, может быть затерт. Пример:

```
0:      MOV #2,R1       Вывод начиная с 3-й позиции
        EMT 6           Пауза до нажатия клавиши
        MOV #TEX,R2     Адрес начала текста
        MOV #233,R0     Команда переключения
        EMT 16           формата
        CLR R0          Очистка
        EMT 22           служебной строки
        MOV 'Пр',(R2)+  Запись в ОЗУ
        MOV 'ив',(R2)+   строки символов -
        MOV 'ет',(R2)+   слова "Привет!"
        MOV '! ',(R2)+
        CLRB (R2)+      Нулевой байт - конец текста
        MOV #TEX,R2     Адрес начала текста
1:      MOVB (R2)+,R0   Вывод текста
        BEQ 0            в служебную строку
        EMT 22
        INC R1          Программа зациклена
        BR 1
TEX:    END
```

• EMT 24 — УСТАНОВКА КУРСОРА ПО КООРДИНАТАМ. Координата X задается в регистре R1, Y — в R2. В графическом режиме по EMT 24 аналогичным образом устанавливается графический курсор. Если значение координат превышает число знакомест (или точек) экрана, то курсор устанавливается по модулю имеющегося на экране числа знакомест. (В данном случае слова

«по модулю» обозначают использование функции MOD, т. е. вычисление остатка от деления. Так, если значение X задано равным 79, а ширина экрана в данном режиме вывода соответствует 64 символам, то курсор устанавливается в позиции, равной 79 MOD 64 = 15. — *Прим. ред.*) Началом координат, как это принято для БК-0010, считается левый верхний угол экрана.

- EMT 26 — ЧТЕНИЕ КООРДИНАТ КУРСОРА. Как и предыдущая команда, EMT 26 работает и в символьном, и в графическом режимах. Значение координат читается: X — в R1, Y — в R2.
- EMT 30 — ФОРМИРОВАНИЕ ТОЧКИ ПО КООРДИНАТАМ. Координаты точки задаются: X — в R1, Y — в R2. В R0 указывается код операции: 1 (или любое ненулевое значение) — запись точки, 0 — стирание. Предварительная установка графического режима не требуется. Цвет (в цветном режиме) может быть задан соответствующей командой заранее. Если координаты точки выходят за пределы экрана, то команда игнорируется, но заданные координаты запоминаются как последние отработанные.
- EMT 32 — ФОРМИРОВАНИЕ ВЕКТОРА. Задаются координаты конца вектора: X — в R1, Y — в R2. В R0 указывается код операции, как и для EMT 30: 1 (или любое ненулевое значение)— запись вектора, 0 — стирание. Может быть предварительно задан цвет. Началом вектора считается конец последнего начерченного или стертого вектора (по EMT 32) либо последняя обработанная точка (по EMT 30). Если координаты начала или конца вектора оказываются за пределами экрана, вектор в соответствии с ними все равно строится, хотя на экране отображается только его видимая часть. Приведем пример.

```
;"муаровый" узор
   MOV #1,R0        Зачерчивание экрана
   CLR R3             из левого верхнего
0: CLR R1             угла фигурой типа
   CLR R2             "радиальный луч"
   EMT 30           Регистр R3 используется
   MOV R3,R1          как буфер координаты
   MOV #357,R2        X конца луча
   EMT 32
   CMP (R3)+,(R3)+
   BR 0             Программа зациклена
```

Отметим, что предпоследняя команда ничего, в общем-то, не сравнивает, а использована только для увеличения содержимого R3 сразу на 4. При этом она занимает всего одно слово, в отличие, например, от команды ADD #4,R3, занимающей два слова. С таким использованием команды CMP мы уже знакомы.

- EMT 12 — УСТАНОВКА КЛЮЧЕЙ КЛАВИАТУРЫ. Как известно, БК-0010 имеет механизм КЛЮЧЕЙ (или функциональных клавиш) — клавиши 0...9 могут быть запрограммированы на выдачу определенных текстовых строк при нажатии их в регистре «НР» («АР2»). Для программирования ключа необходимо задать его номер (от 1 для клавиши «1» до 12 для «0») в регистре R0, а адрес текста ключа в памяти — в R1 и выполнить EMT 12. Первый байт текста должен содержать длину строки в байтах и как символ не выводится. При содержимом R1, равном 0, происходит сброс ключа. Если программирование ключей производится для языка высокого уровня, например ФОКАЛа, то тексты ключей необходимо размещать в специально отведенной для этого зоне (для ФОКАЛа — по адресам 1000...1377). R0 не сохраняется. Пример (программирование ключа 5):

```
        MOV #TEX,R1          Адрес начала текста
        MOVB #7,(R1)+        Число символов текста
        MOVB 'K',(R1)+       Текст
        MOV 'EY',(R1)+       "KEY N 5"
        MOV ' N',(R1)+
        MOV ' 5',(R1)+
        MOV #5,R0            Номер ключа
        MOV #TEX,R1          Адрес текста ключа
        EMT 12               Запрограммировать ключ
0:      BR 0                 Ожидание (вечный пустой цикл)
TEX:    END
```

После запуска данной программы ключ будет запрограммирован. Если нажать клавишу «СТОП», можно выйти в ПМ или МСД и проверить результат, нажимая клавишу «5» в регистре «АР2». (Отметим, что ожидание с помощью BR 0 в конце программы нам понадобилось только потому, что мы еще не успели познакомиться с командой останова процессора.)

- EMT 34 — ЧТЕНИЕ СЛОВА СОСТОЯНИЯ ДИСПЛЕЯ (ССД, не путать с ССП!). ССД — это двухбайтное машинное слово, дающее исчерпывающее представление об установленных режимах дисплея БК-0010. По EMT 34 ССД заносится в R0, причем каждому режиму дисплея соответствует один из битов: если он равен 1, то режим включен, если 0 — выключен (или включен альтернативный режим). В системной области ОЗУ начиная с адреса 40 размещен массив служебных переменных — признаков режимов, с которых и считывается при отработке EMT 34 слово состояния дисплея. При этом каждому биту ССД соответствует байт в памяти, его содержимое равно 0 при бите ССД, равном 0, и 377 — при 1 (кроме бита 03, сигнализирующего о включении русского регистра, которому соответствует содержимое ячейки 43, равное 200), как это указано в таблице ССД.

Таблица ССД

Разряд ССД	Адрес байта	Включенный режим дисплея
00	40	Цвет (режим 32 символа в строке)
01	41	Инверсия экрана
02	42	Режим «РП» (расширенной памяти)
03	43	Регистр «РУС»
04	44	Подчеркивание
05	45	Инверсия символа
06	46	Индикация символов управления
07	47	Блокировка редактирования
08	50	Режим «ГРАФ»
09	51	Режим «ЗАП»
10	52	Режим «СТИР»
11	53	Режим 32 символа в служебной строке
12	54	Подчеркивание в служебной строке
13	55	Инверсия символа в служебной строке
14	56	Гашение курсора
15	57	Не используется

Очевидно, что, пользуясь ССД (или прямо проверяя значения служебных байтов — переменных), можно установить, какие режимы

дисплея включены, и программно переключать нужные режимы как в кодах (выдавая необходимые команды с помощью EMT 16), так и на языках высокого уровня.

- EMT 36 — РАБОТА С МАГНИТОФОНОМ. Это самое сложное из всех командных прерываний БК-0010. Перед обращением к нему в R1 заносится адрес БЛОКА ПАРАМЕТРОВ. Блок параметров — это специально выделенная зона ОЗУ, куда предварительно записывается информация, обеспечивающая работу EMT 36. Блок параметров может быть размещен начиная с любого четного адреса ОЗУ, но в системной области под него специально зарезервирована зона адресов 320...371. Приведем распределение ячеек этой зоны (при расположении блока параметров в любом другом месте ОЗУ последовательность ячеек сохраняется):

Адреса байтов	Содержимое
320	Команда
321	Ответ
322,323	Адрес массива на запись или чтение
324,325	Длина массива на запись (при чтении нужно обнулять)
326...345	Имя массива на запись или чтение
346,347	Адрес текущего массива
350,351	Длина текущего массива
352...371	Имя текущего массива

(ТЕКУЩИМ называется массив (файл), имя которого прочитано последним, независимо от того, загружался ли он в ОЗУ.) Вот по этим-то адресам и заносятся все необходимые данные при чтении или записи файлов в ФОКАЛе, БЕЙСИКе, ПМ или МСД.

Байт «КОМАНДА» в блоке параметров задает необходимый режим работы ЭВМ с магнитофоном и может принимать значения:

0 — останов двигателя магнитофона,
1 — пуск двигателя,
2 — запись файла на МЛ,
3 — чтение с МЛ,
4 — фиктивное чтение.

Отметим, что при исполнении команд 2, 3 и 4 двигатель магнитофона включается и отключается автоматически, отдельная подача команд 0 и 1 не требуется. (Вообще говоря, команды 0 и 1 нужны только при оборудовании магнитофона дистанционным управлением. В этом случае, когда после завершения операций 2, 3 или 4 БК отключает двигатель, невозможно даже просто перемотать ленту — дистанционное управление блокирует запуск двигателя с клавиш магнитофона. Для ручной перемотки нужно либо отключить ДУ, отстыковав соответствующий штекер магнитофонного кабеля БК, либо, что проще, заранее предусмотреть в программе команды включения-выключения двигателя аналогично реализации МП и МС в МСТД. — *Прим. ред.*)

Байт ОТВЕТА сообщает о результатах исполнения команды и может принимать следующие значения:

0 — операция завершена без ошибок,
1 — имя файла на МЛ не совпадает с заданным,
2 — ошибка контрольной суммы,
4 — останов по клавише «СТОП».

Теперь, наверное, вам понятно, как записать или прочитать заданный файл. Рассмотрим пример (выбран вариант записи файла на МЛ, как более сложный):

```
; Запись файла на МЛ с помощью EMT 36
BEG:  MOV #346,R3          Конец имени файла
0:    MOVB #40,-(R3)       Очистка имени файла
```

```
        CMP #320,R3          (запись пробелов во все байты,
        BNE 0                до R3 = 320)
        MOV #2,(R3)+         Команда "ЗАПИСЬ"; R3+2 = 322
        MOV 'А=',R0          Запрос "АДРЕС="
        CALL INP             Ввод адреса; R3+2 = 324
        MOV 'Д=',R0          Запрос "ДЛИНА="
        CALL INP             Ввод длины; R3+2 = 326
        MOV 'И=',R0          Запрос "ИМЯ="
        EMT 16               Выдача запроса
        SWAB R0              на экран
        EMT 16
        MOV #5020,R2         Ограничители: "ВВОД" или длина 16д
        MOV R3,R1            Адрес имени - 326
        EMT 10               Ввод имени
        MOVB #40,-(R1)       Заменить последний байт имени
                             ; (код 12) на пробел
        MOV #320,R1          Адрес блока параметров
        EMT 36               Выполнить запись файла
        EMT 6                Ожидание нажатия клавиши
        BR BEG               Программа зациклена

; Выдача запросов, ввод числовых данных
INP:    EMT 16               Первый символ запроса
        SWAB R0
        EMT 16               Второй символ
        CALL @#100472        Ввод восьмеричного числа в R5
        MOV R5,(R3)+         Число - в блок параметров; R3+2
        MOV #12,R0           Перевод
        EMT 16               строки
        RET                  Выход из подпрограммы

        END                  Конец программы
```

Построчный комментарий, видимо, делает ненужными дальнейшие пояснения. Отметим только, что для ввода восьмеричного числа, задающего начальный адрес и длину, мы воспользовались подпрограммой ПЗУ, расположенной по адресу 100472, — это избавило от необходимости отводить под аналогичную, довольно длинную подпрограмму место в ОЗУ. Часть команд программы, повторяющихся дважды, вынесены в подпрограмму INP. Данный алгоритм не является единственно возможным, но автор стремился сделать программу предельно короткой. Удалось ли это — решать читателям, когда они начнут писать свои аналогичные программы. Тем и интересно программирование, что вариантов решения одной задачи, даже не слишком сложной, может быть множество. Нередко при работе с файлами требуется не ввод адреса и длины с клавиатуры, а задание этих параметров программно (например, начало и длина файла, набираемого в редакторе текстов), тогда алгоритм будет иным. Если же требуется не запись файла на МЛ, а его чтение, фиктивное чтение или управление магнитофоном, характер программы также изменится. Мы надеемся, что после приведенного примера читателю не составит труда написать эти программы самостоятельно.

Пользуясь случаем, рассмотрим кратко формат записи файла БК-0010 на МЛ. Файл начинается с УСТАНОВОЧНОЙ ПО-

СЛЕДОВАТЕЛЬНОСТИ (УсП), которая представляет собой запись 10000 нулевых битов. Эта последовательность нулей служит, во-первых, для поиска начала файла, а во-вторых, для автоматической подстройки скорости чтения (для разных БК-0010, а особенно для разных магнитофонов она может существенно различаться; это же свойство обеспечивает возможность записи файлов с повышенной скоростью). Но ведь последовательность из 10000 нулей может присутствовать и в самом файле! Чтобы не спутать такой фрагмент с началом файла, на МЛ вслед за УсП записывается МАРКЕР — специальная кодовая последовательность импульсов. Помимо опознавания начала файла маркер служит также для определения ФАЗЫ записи — ведь полярность выхода различных магнитофонов может быть разной. Если маркер опознан, начинается чтение следующего за ним ОГЛАВЛЕНИЯ ФАЙЛА, представляющего собой точную копию блока информации, находящейся при записи файла по адресам 322...345 (т. е. адреса, длины и имени файла), но при чтении эти параметры заносятся по адресам 346...371. После чтения оглавления производится сверка имени файла, и при его несовпадении с заданным в ячейку 301 записывается байт ответа и производится выход в основную программу. Затем, если имя совпало с заданным на чтение, производится, в зависимости от команды, истинное либо фиктивное чтение. После загрузки в ОЗУ БК считывает последнее слово файла — КОНТРОЛЬНУЮ СУММУ (КС). Затем вычисляется КС загруженного в ОЗУ массива и сравнивается с прочитанной. В зависимости от результата этого сравнения формируется байт ответа в блоке параметров и работа команды EMT 36 завершается. Запись производится аналогично. КС, сопровождающая файл на МЛ, вычисляется как сумма всех его байтов с учетом битов переноса. При вычислении КС загруженного файла его УсП, маркер, оглавление и сама КС не учитываются.

(Завершая обсуждение функции EMT 36, следует обратить внимание на ее использование в БК, оснащенных дисководом. При работе с большинством имеющихся сегодня дисковых систем (кроме разве лишь давно устаревших версий НОРТОНа для БК без расширенного ОЗУ) реализуется автоматическая «переадресация» чтения-записи файлов по EMT 36 на диск. Делается это путем перехвата EMT-прерывания и подмены функции EMT 36 на ее «дисковую» версию, что обеспечивает нормальную работу с диском большинства старых программ, рассчитанных на магнитофон.

Из сказанного следует два достаточно важных вывода. Во-первых, при написании прикладных программ с целью обеспечения их универсальности ввод-вывод файлов нужно производить стандартным способом через EMT 36, а не путем прямого обращения к «магнитофонным» подпрограммам монитора (кроме особых случаев, когда это действительно необходимо, а программа рассчитана только на работу с магнитной лентой). Второй вывод касается адаптации уже существующих программ к диску. Эта адаптация в большинстве случаев состоит в обеспечении загрузки с диска отдельных частей многофайловых программ. Так вот, делать это следует тоже только с использованием стандартной функции EMT 36. Попытки же писать собственные загрузчики, непосредственно работающие с каталогом имен и секторами данных на диске, приводят к тому, что подобная адаптация работает лишь в одной ОС. А в результате кому-то приходится в конце концов «взламывать» такую программу и делать адаптацию заново, дабы получить возможность работы с ней в других операционных системах. — *Прим. ред.*)

Теперь кратко упомянем несколько оставшихся EMT. Все они относятся к работе с ТЛГ-каналом. Поскольку эти команды применяются, когда необходимо обеспечить связь между несколькими ЭВМ, между ЭВМ и принтером с последовательным каналом и т.п., то ясно, что они — «не для простых смертных», а для довольно квалифицированных пользователей (программистов). Как уже говорилось, для использования ТЛГ-канала в некоторых моделях БК необходимо установить на плате компьютера перемычки, чтобы соединить вход

и выход ТЛГ с контактами разъема порта ввода-вывода, что также требует некоторой квалификации. Перемычки показаны на принципиальной схеме ЭВМ (см. № 2 за 1994 г., с.35, справа внизу, перемычки 52 и 53. — *Прим. ред.*), а мы на этом останавливаться не будем, чтобы не «провоцировать» начинающих на рискованные эксперименты, грамотный же специалист легко найдет их сам. Как говорится, «Богу — Богово...».)

Несколько более подробные сведения о данных командах содержатся в «Руководстве системного программиста», прилагаемом к БК, а мы их просто перечислим и назовем.

- EMT 40 — ИНИЦИАЛИЗАЦИЯ ДРАЙВЕРА ТЛГ-КАНАЛА.
- EMT 42 — ПЕРЕДАЧА БАЙТА ПО ТЛГ-ЛИНИИ.
- EMT 44 — ПРИЕМ БАЙТА С ТЛГ-ЛИНИИ.
- EMT 46 — ПЕРЕДАЧА МАССИВА ПО ТЛГ-ЛИНИИ.
- EMT 50 — ПРИЕМ МАССИВА С ТЛГ-ЛИНИИ.

В EMT-диспетчере БК-0010 предусмотрены также еще 16 так называемых РЕЗЕРВНЫХ ВХОДОВ EMT с номерами 52, 54, ... 110. При обращении к ним происходит передача управления по адресам, равным соответственно 160000, 160004, ... 160074. Эти входы могут быть использованы, если вместо тест-ПЗУ разместить, начиная с адреса 160000, ПЗУ пользователя. (При подключенном контроллере дисковода его ПЗУ попадает как раз в область адресов начиная с 160000, и тогда некоторые из названных EMT-функций можно использовать для вызова подпрограмм доступа к диску. Подробнее об этом см. в № 1 за 1993 г., с. 102. — *Прим. ред.*)

На этом рассмотрение командных прерываний EMT можно считать законченным. К вопросу о том, как использовать полученные знания, мы вернемся позже, а пока продолжим изучение операторов ассемблера и первым разберем уже упомянутый ранее оператор TRAP.

- TRAP X — КОМАНДНОЕ ПРЕРЫВАНИЕ ПО ВЕКТОРУ 34, где X — номер TRAP, восьмеричное число от 0 до 377. Предназначен этот оператор в основном для программ пользователя, а в остальном он очень похож на EMT. Пользователь может написать свою программу TRAP-диспетчера и создать свой пакет программ обработки прерываний TRAP, аналогичный EMT-пакету, но решающий другие задачи. Воспользуемся случаем и приведем пример простейшего TRAP-диспетчера. Предположим, что мы хотим использовать прерывание TRAP для такой банальной цели, как вывод на экран символов, коды которых соответствуют номерам TRAP. Вспомним, как выполняется командное прерывание. Обнаружив в программе команду прерывания TRAP, процессор выполняет действия, которые можно условно изобразить следующей программой:

```
MFPS -(SP)      Сохранить текущее ССП
MOV PC,-(SP)    Сохранить адрес следующей
                ; команды
MTPS @#36       Записать новое ССП
MOV @#34,PC     Записать адрес программы
                ; обработки прерывания
```

Конечно, такую программу нам писать не нужно, процессор делает это сам: программа обработки прерываний заложена в него на аппаратном уровне (если точнее — на так называемом МИКРОПРОГРАММНОМ). Но вот процессор перешел к программе обработки прерывания. Что будет собой представлять эта программа для нашего случая? Раз мы хотим получить вывод символов, код которых соответствует номеру TRAP, нам, видимо, следует извлечь этот номер. Вспомним, что коды TRAP лежат в диапазоне 104400...104777, т. е. искомый номер — это младший байт команды. А как «выудить» командный код? В стеке у нас сохранен адрес следующей команды. Уменьшив его на 2, получим адрес TRAP. А дальше уже просто: косвенное обращение по этому адресу даст код TRAP, используем его младший байт как код символа, вот и все! Но прежде чем все это делать, мы должны записать в вектор 34

значение, соответствующее адресу нашей программы обработки прерывания. Пусть программа обработки TRAP помечена меткой TRP. Запишем:

```
MOV #TRP,@#34     Записать новый
                  ;вектор 34
```

Если бы требовалось на время исполнения TRAP запретить все прерывания от внешних устройств, мы могли бы задать также нужный приоритет процессора во втором слове вектора, например:

```
MOV #340,@#36     Задать максимальный
                  ;приоритет
```

Однако в данном случае это не принципиально. А теперь напишем саму программу обработки прерывания.

```
; Программа обработки прерывания TRAP
TRP: MOV R0,-(SP)          Сохранить R0
     MOV 2(SP),R0          Извлечь адрес следующей команды
     MOV -(R0),R0          Извлечь текущую команду (TRAP)
     EMT 16                Вывести на экран символ, код
                           ; которого в младшем байте R0
     MOV (SP)+,R0          Восстановить R0
     RTI                   Выход из прерывания
```

Как видим, программа получилась очень простой, это диктуется целью, которую мы себе поставили. Если теперь написать, например, последовательность команд: **TRAP 101, TRAP 102, TRAP 103**, то на экран будут выведены три символа: **«ABC»**. Команды их выдачи на экран занимают всего три слова, тогда как, например, при «классическом» способе вывода (занести код символа в регистр R0 и выполнить EMT 16) потребовалось бы девять слов! Между тем, сама программа обработки TRAP занимает у нас семь слов. При достаточно частом обращении к ней мы получим существенную экономию памяти. Если же задача сохранения и последующего восстановления R0 не ставится, то программа обработки TRAP будет еще короче.

(Может показаться, что в данном примере функция TRAP используется слишком уж «примитивно». Однако на практике необходимость в реализации подобного механизма передачи аргумента функции через номер команды TRAP встречается довольно часто. Так, в трансляторе вильнюсского БЕЙСИКа именно таким способом осуществляется обращение к подпрограмме выдачи на экран сообщений об ошибках: номер ошибки содержится в младшем байте команды TRAP. — *Прим. ред.*)

- IOT — КОМАНДНОЕ ПРЕРЫВАНИЕ ПО ВЕКТОРУ 20. Предназначено для обслуживания устройств ввода-вывода в операционных системах, в БК-0010 практически не используется. Выполняется точно так же, как EMT или TRAP, с той разницей, что младший байт команды не содержит дополнительной информации (номера). Может быть использовано для обращения к какой-либо одной подпрограмме.
- BPT — КОМАНДНОЕ ПРЕРЫВАНИЕ ПО ВЕКТОРУ 14. Используется в отладчиках, подробно будет рассмотрено позже.
- RTI — ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ. Используется для выхода из подпрограммы обработки прерываний (кроме отладочных). Восстанавливает стек, ССП и адрес очередной команды. При отработке оператора RTI процессор автоматически выполняет (на микропрограммном уровне) действия, аналогичные последовательности команд:

```
MOV (SP)+,PC      Восстановить адрес
                  ; команды прерванной
                  ; программы
MTPS (SP)+        Восстановить ССП
```

- RTT — ВОЗВРАТ ИЗ ОТЛАДОЧНОГО ПРЕРЫВАНИЯ. Используется для выхода из отладочного прерывания, когда

установлен Т-разряд ССП, но может применяться и для выхода из других прерываний. Отличие RTT от RTI состоит лишь в том, что прерывание по Т-разряду при отработке RTT не наступает, а выполняется следующая команда и лишь после нее происходит прерывание по Т-разряду. Можно сказать, что команда RTT «непрозрачна при трассировке». (Подробно о ее использовании мы поговорим позднее.)

- HALT — ОСТАНОВ. В сущности, эта команда есть не что иное, как командное прерывание по вектору 4, и дальнейшие действия ЭВМ всецело зависят от того, как построена программа обработки прерывания по этому вектору. Команда (обычно неправильно относимая к «командам управления машиной») может быть использована не только для останова, но и для любых других целей, аналогично прочим командам прерываний, но нужно помнить, что, изменяя вектор 4 и программу обработки прерывания по этому вектору, мы меняем и порядок работы клавиши «СТОП».

Операторы управления машиной

- WAIT — ОЖИДАНИЕ. Приостанавливает выполнение текущей программы до прерывания от внешних устройств (клавиатуры, таймера, клавиши «СТОП»). Может быть использован для организации паузы и ожидания реакции пользователя (или прерывания от внешних устройств), но нужно помнить, что прерывания возникают при этом в соответствии с приоритетом внешних устройств и процессора, и для прерывания, например, от клавиатуры приоритет процессора должен быть менее 4. (Применяется редко.)

- RESET — СБРОС ВНЕШНИХ УСТРОЙСТВ. Оператор вызывает выдачу сигнала INIT по командной магистрали общей шины, который сбрасывает все устройства ЭВМ (порт, системные регистры) в исходное состояние. Особенность выполнения данного оператора на БК-0010 такова, что после него обычно необходимо инициализировать драйвер клавиатуры командой EMT 4. Применяется крайне редко. (Оператор RESET может быть использован и нестандартно, обеспечивая при этом возможность распознавания одновременного нажатия нескольких клавиш на клавиатуре БК — см. № 1 за 1993 г., с. 129. — *Прим. ред.*)

Прочие операторы

- NOP — «ПУСТАЯ» ОПЕРАЦИЯ. Используется при отладке, если нужно временно исключить какую-то команду, или для создания небольшой задержки (около 4 мкс). Не делает ничего. Код этой команды — 240.

К данной группе относятся также КОМАНДЫ ИЗМЕНЕНИЯ ПРИЗНАКОВ, т. е. разрядов 00...03 ССП. Используются они редко.

- CLN — ОЧИСТКА N.
- CLZ — ОЧИСТКА Z.
- CLV — ОЧИСТКА V.
- CLC — ОЧИСТКА C.
- CCC — ОЧИСТКА ВСЕХ РАЗРЯДОВ.
- SEN — УСТАНОВКА N.
- SEZ — УСТАНОВКА Z.
- SEV — УСТАНОВКА V.
- SEC — УСТАНОВКА C.
- SCC — УСТАНОВКА ВСЕХ РАЗРЯДОВ.

(Существуют и другие машинные команды, обеспечивающие очистку или установку одновременно двух или трех признаков, но они не имеют сокращенной ассемблерной записи. Их коды приведены в Приложении 4 к книге: *Осетинский Л. Г., Осетинский М. Г., Писаревский А. Н.* ФОКАЛ для микро- и мини-компьютеров. Л.: Машиностроение, 1988. — *Прим. ред.*)

Псевдооператоры

Как уже неоднократно говорилось, каждый оператор с принадлежащими ему

операндами (если они имеются) при трансляции переводится в машинный код, образуя машинную команду, или инструкцию. Но имеется ряд специфических задач, для решения которых обычных операторов недостаточно (например, резервирование места под константы, переменные и массивы, запись в память текстов (вы помните, как неуклюже мы делали это в наших прежних примерах?), символьных и цифровых последовательностей и т. п.) Этой цели служат ПСЕВДООПЕРАТОРЫ, или псевдокоманды.

В отличие от обычных операторов, псевдокоманды при трансляции не переводятся в машинные коды, они представляют собой лишь указания транслятору, что записать в память с текущего адреса или что выполнить при трансляции. Заносимая в память информация, если она имеется, следует после псевдооператора. Рассмотрим, какие псевдооператоры имеются в ассемблерах «МИКРО». Все они начинаются с символа «ТОЧКА», который в ассемблере означает ТЕКУЩИЙ АДРЕС. Таким образом, любой псевдооператор можно истолковать как указание транслятору: «записать с текущего адреса...».

- .Е — ОБНУЛЕНИЕ СЛОВА ИЛИ БАЙТА И ПРИВЕДЕНИЕ ТЕКУЩЕГО АДРЕСА К ЧЕТНОМУ. Как известно, команды ассемблера после трансляции занимают от одного до трех машинных слов. Но слово — это два байта, а значит, любая команда может после трансляции начинаться лишь с ЧЕТНОГО адреса. Однако бывают ситуации, когда при написании программы мы не можем с определенностью сказать, четным ли будет текущий адрес. Это случается, например, если в текст программы вписана последовательность символов или байтовых кодов. С другой стороны, при выводе, скажем, текстовых сообщений удобно отмечать конец текста нулевым байтом. Обе задачи решаются одновременно, если в конце текста (или последовательности кодов) стоит псевдооператор .Е . Встречая его, транслятор заносит в память нулевой байт, если адрес нечетный, или нулевое слово, если четный. Адрес, таким образом, всегда приводится к четному. Кроме того, данный псевдооператор можно применять для резервирования отдельных слов памяти под переменные, если их исходное значение — ноль.

- .+X — ЗАРЕЗЕРВИРОВАТЬ X БАЙТ ОЗУ. Встречая в тексте этот псевдооператор, транслятор обнуляет указанное количество байтов (вместо X записывается конкретное восьмеричное число) и увеличивает на X текущий адрес. Применяется для выделения памяти под массивы, приведения текущего адреса к нужному, «подгонки» длины программ под «круглое» число и т. п. Пусть, например, мы с помощью оператора EMT 10 будем вводить текст максимальной длиной 20 байт, а начало буфера текста соответствует метке Т1. Тогда для выделения буфера под текст достаточно записать: Т1: .+20.

- .#X или .#МЕТ — ЗАПИСЬ КОНСТАНТЫ В СЛОВО ПО ТЕКУЩЕМУ АДРЕСУ. Константой может быть как восьмеричное число (без знака или со знаком «минус»), так и имя глобальной метки (можно и с коррекцией адреса в виде выражений МЕТ+X или МЕТ-X, где X — восьмеричное число). В составе псевдокоманды может быть через запятую перечислено несколько констант, например: .#14563, ТЕХ + 140, Т27, -20, 24, 0, 532.

Каждая из перечисленных таким образом констант записывается в очередное машинное слово. Если константой является адрес метки, то его абсолютное значение транслятор заносит только после компоновки программы. В менее совершенных версиях ассемблеров допускается запись после .# только одного значения константы, но зато часто разрешается писать по несколько псевдооператоров в строке.

Этот псевдооператор широко применяется для записи констант, переменных (с заданием их исходного значения), а также

строк абсолютных адресов меток, которые можно затем использовать для адресации различных модулей программы.

Приведем пример такой записи. Пусть в программе имеется 10 модулей, помеченных метками M0, M1, M2,... M9, а директивы передачи управления этим модулям — вводимые с клавиатуры цифры «0»,«1»,« 2»,...«9» соответственно. Как оптимально организовать блок управления такой программой? Запишем:

```
BEG: EMT 6              Ввод директивы
     ASL R0             Умножить на 2
                        код цифры
     JMP @ADR-140(R0)   Передача
                        управления модулю
ADR: .#M0,M1,M2,M3,M4,M5,M6,M7,M8,M9
```

Первая строка программы вполне понятна, мы заносим с клавиатуры в R0 код директивы, причем нулю соответствует код 60, единице — 61, и т. д. В последней же строке записаны адреса меток модулей, на которые мы должны передать управление. Адрес каждой метки, естественно, занимает одно слово, а значит, адреса следуют с интервалом два байта, а коды управления 60, 61... и т. д. — с интервалом в один байт. Чтобы привести последний в соответствие с интервалом следования меток, во второй строке мы умножаем код на 2. А вот в третьей строке, которая, собственно, и передает управление, использованы все возможные ухищрения адресации. Мы передаем управление по адресу, который, в свою очередь, записан по адресу (двойная косвенность!), определяемому выражением: R0+#ADR−140. А число 140 есть не что иное как удвоенный код символа «0». Таким образом, если, например, мы вводим директиву «3», в результате у нас получается 63X2+#ADR−40 = #ADR+6. А по адресу #ADR+6 записан адрес метки M3, на нее и будет передано управление. Отметим, что наша программа не имеет защиты от ошибок — если мы введем директиву, код которой больше кода клавиши «9» или меньше, чем у «0», результат будет непредсказуем. Следовало бы после оператора EMT 6 проверять содержимое R0 на соответствие заданному диапазону кодов, например так:

```
BEG: EMT 6
     MOV #DIR,R4
0:   CMPB R0,(R4)+
     BEQ 1
     TSTB @R4
     BNE 0
     BR BEG
1:   SUB #DIR,R4
```

```
     CMPB '0',R0
     BHI BEG
     CMPB '9',R0
     BLO BEG
```

(В первоначальном тексте примера мы этот фрагмент не привели, заботясь о его максимальной простоте.)

- .B: — ЗАПИСЬ КОНСТАНТЫ В БАЙТ ПО ТЕКУЩЕМУ АДРЕСУ. В качестве констант могут использоваться только восьмеричные числа от 0 до 377. Несколько чисел после одного псевдооператора может быть записано через запятую (в ранних версиях — только одно число). Эта псевдокоманда применяется в основном для записи строк кодов управления режимами вывода на экран или для простейшей генерации звука (код 7 — щелчок как при нажатии на клавишу). Можно также, записав строку кодов клавиш управления, найти в ней заданный код (директиву) и передать управление на модуль программы, пользуясь, в свою очередь, строкой меток, например так:

```
Ввод директивы
Начало таблицы кодов директив
Код совпадает?
Да - передать управление
Конец таблицы?
Нет - продолжать поиск кода
Иначе (кода в таблице нет) идем к началу
Номер кода директивы
```

```
        ASL R4                  Умножить на 2
        JMP @ADR-2(R4)          Передать управление по таблице
                                ; меток, в соответствии с номером
                                ; кода директивы, с поправкой

DIR: .B:10,22,31,32,33.E
ADR: .#M1,M2,M3,M4,M5
```

В данной программе мы используем две таблицы. В первой, под меткой DIR содержится список кодов директив (используются, для примера, коды клавиш управления курсором). Введя директиву, мы ищем ее код в таблице. Если он найден, вычисляем его номер от начала таблицы. Но коды директив — байты, а адреса меток передачи управления — слова, поэтому удваиваем номер и вычисляем адрес соответствующей метки в таблице ADR (как адрес начала таблицы плюс номер метки). По этой метке и передается управление. Поясним, что запись @ADR-2 появилась в программе потому, что после нахождения директивы содержимое регистра R4 инкрементируется и указывает на адрес следующей директивы, из-за чего при обращении к метке мы должны ввести поправку. Псевдокоманда .E в конце таблицы DIR приводит адрес метки ADR к четному, а нулевой байт отмечает конец таблицы. Чтобы все было окончательно ясно, покажем, как можно организовать модули программы, к которым мы обращаемся таким образом. Пример выбран простейший — по нажатиям клавиш-директив наша программа будет выдавать имена меток (но так можно построить любые модули). После программы, введенной в предыдущем примере, допишем:

```
M1:  MOV 'M1',R0        M1 - «стрелка влево»
     BR TYP
M2:  MOV 'M2',R0        M2 - «стрелка в начало экрана»
     BR TYP
M3:  MOV 'M3',R0        M3 - «стрелка вправо»
     BR TYP
M4:  MOV 'M4',R0        M4 - «стрелка вверх»
     BR TYP
M5:  MOV 'M5',R0        M5 - «стрелка вниз»
     BR TYP
TYP: EMT 16             Печать на
     SWAB R0            экране имени
     EMT 16             метки
     BR BEG             К началу
     END
```

Данный способ передачи управления интересен тем, что таблицы могут расширяться неограниченно, по мере написания новых модулей программы. Если же вместо оператора JMP @ADR-2(R4) использовать CALL @ADR-2(R4), то в качестве модулей можно использовать подпрограммы, после исполнения которых управление будет возвращаться в основную программу, — иногда такой способ удобнее.

- .@MET — СМЕЩЕНИЕ К МЕТКЕ. При трансляции в слово по текущему адресу заносится разность между адресом метки и текущим адресом — СМЕЩЕНИЕ К МЕТКЕ. Как обычно, можно ввести в это смещение поправку, записав псевдооператор в виде .@MET+X или .@MET-X, где X — восьмеричное число. Эта псевдокоманда широко используется для получения абсолютных адресов меток в перемещаемых программах. В самом деле, когда мы заносим абсолютный адрес метки, например, оператором MOV #MET,R4, вто-

рое слово данной команды заносится в память один раз при компоновке, а его значение зависит от заданного начального адреса, по которому программа компонуется и в дальнейшем будет работать. Стоит только изменить адрес загрузки такой программы, и она работать не будет, так как действительные адреса меток перестанут совпадать с полученными при компоновке. Иначе дело обстоит, если задано смещение: как бы мы ни перемещали программу, «расстояние» от текущего адреса до метки будет неизменным. Как пользоваться данной псевдокомандой? Предположим, что нам нужно получить в каком-либо регистре абсолютный адрес метки МЕТ. Напишем программу:

```
MOV PC,R4       Занести текущий адрес
ADD (PC)+,R4    Прибавить смещение
.@MET+2         Смещение с поправкой на +2
```

Мы получили абсолютный адрес метки МЕТ в регистре R4. Каким образом это делается? Вначале мы записали в R4 текущий адрес. Во второй строке прибавили к этому адресу смещение, полученное с помощью псевдооператора .@ в третьей строке. Мы добились этого, косвенно обращаясь через регистр PC, в котором в этот момент находится адрес следующей команды, т. е. как раз нужной нам строки. Но в момент обращения к PC в первой строке мы записали адрес, на 2 меньший, чем адрес строки псевдооператора, и вынуждены исправить эту ошибку, внеся поправку в смещение («+2»). То, что мы во второй строке не просто косвенно обращаемся к PC, а с инкрементом, служит важной цели — увеличить содержимое PC на 2 и таким образом «перескочить» строку с псевдооператором: передавать на нее управление нельзя, там не команда, а произвольное число. Пользуясь таким приемом, можно получить адрес метки независимо от того, находится она до или после псевдооператора. Как всегда в таких случаях, метка должна быть глобальной, а не локальной. Возможны и иные приемы получения абсолютных адресов меток с помощью данного псевдооператора, но предложенный — самый экономичный по расходу памяти и быстродействию.

Все вышеперечисленные псевдооператоры могут быть записаны как в одну строку друг за другом в любой последовательности, так и в отдельных строках, но при этом надо учитывать, что псевдокоманды .# и .@ могут правильно работать только при их трансляции по четным адресам. В псевдооператоре .@ можно, в отличие от .#, указывать только одно имя метки, перечислять их через запятую нельзя. В записанной последовательности псевдооператоров недопустимы пробелы и прочие символы, разрывающие ее. Такой символ либо ведет к ошибке, либо считается концом строки — и все, следующее за ним, транслятор игнорирует.

- .A: — ЗАПИСЬ СТРОКИ СИМВОЛОВ В КОДЕ КОИ-8. Код КОИ-8 (а точнее, одна из его модификаций) — это «родной» символьный код БК-0010, включающий все знаки и коды управления в диапазоне 0...377. (Иногда этот код называют «кодом ASCII» по аналогии с американским стандартом. Но, строго говоря, код ASCII — это совсем другой стандарт и никакого отношения к БК не имеет. Достаточно сказать, что он, естественно, не содержит русских символов.) Так вот, данный псевдооператор позволяет записать в строку любые символы БК-0010, а при трансляции их коды будут занесены в память начиная с текущего адреса, причем код каждого символа занимает один байт. Поскольку все, что следует за .A: , транслятор интерпретирует как символьный код, то этот псевдооператор может быть в строке только последним — после него недопустимы никакие операторы или комментарии. Во всех версиях «МИКРО.К» пробелы заносятся в строку символов в любом месте обычным образом, в версиях же «МИКРО.С» занесенные пробелы не всегда транслируются правильно (в частно-

сти, игнорируются пробелы в конце строки). Если в строку символов нужно ввести управляющие коды (перевод строки, сброс экрана и т. п.), их можно задать с помощью псевдооператора .B . Для самых распространенных кодов управления в «МИКРО.10К» последних версий введена возможность вставлять их в строку в виде символов «↑» (возведение в степень) и «\» (обратная косая черта), которые обозначают, соответственно, код 12 (перевод строки) и код 0. Этот псевдооператор очень широко применяется для ввода в состав программ меню, текстовых сообщений, инструкций, таблиц кодов и т. п. Примеры его использования будут приведены чуть позже.

- .R: — ЗАПИСЬ СТРОКИ СИМВОЛОВ В КОДЕ RADIX-50. Как уже было сказано, при кодировании текста в коде КОИ-8 каждый символ занимает один байт. Это и понятно, ведь используются все коды в диапазоне 0...377. Но иногда достаточно и меньшего количества кодов, например только цифр и латинских символов. Простой расчет показывает, что если ограничиться только 40д символами, то в каждом машинном слове теоретически можно разместить до трех символов (если не верите, извлеките корень третьей степени из числа 65536д). Но такое размещение явно не пройдет «напрямую». В самом деле, если разделить 16 на 3, получится 5 и 1 в остатке. А в пяти битах можно закодировать не более 32д символов. Чтобы можно было полностью использовать информационную емкость машинного слова, прибегают к следующему приему. Весь кодируемый текст делят на триады символов, а затем в каждое слово записывают одну из триад по правилу: **ЗНАЧЕНИЕ = ((С1 × 50) + С2) × 50 + С3**, где С1, С2 и С3 — первый, второй и третий коды символов (все числа и вычисления — в восьмеричной системе). Десятичное число 40 равно восьмеричному 50, отсюда и название кода. Все это позволяет при ограниченном наборе символов достичь повышения плотности упаковки текста в полтора раза и применяется, когда необходимо упаковать длинный текст при максимальной экономии памяти. Упаковка и распаковка кода RADIX-50 не слишком простая задача, требующая заметного времени по сравнению с обработкой текстов в формате КОИ-8, что наряду с ограниченным набором символов является препятствием для широкого применения RADIX-формата. Приведем таблицу восьмеричных значений кода RADIX-50:

Символ	Код RADIX-50
ПРОБЕЛ	0
A...Z	1...32
$	33
.	34
РЕЗЕРВ	35
0...9	36...47

Код 35 не используется, а код 34 не задействован в «МИКРО», что называется «по техническим причинам». Символ, не входящий в состав кода RADIX, считается концом строки. Если количество преобразуемых символов не кратно 3, то последиие один или два символа автоматически дополняются до полной триады пробелами. Псевдооператор .R , как и .A , должен быть последним в строке.

★ ★ ★

Осталось рассмотреть последний оператор — END. Он вполне может быть отнесен к псевдооператорам, так как не транслируется, а только сообщает транслятору о конце текста программы. Весь текст, следующий за ним, транслятор игнорирует. Хотя этот оператор не обязателен в «МИКРО.10К» и в некоторых иных версиях ассемблера, все же лучше писать его в конце программы во всех случаях. Помимо основного назначения, он может применяться, если нужно оттранслировать только часть программы, в этом случае оператор END временно ставится в конце транслируемой части.

А теперь приведем примеры использования разобранных выше псевдооператоров.

1)
```
; Вывод на экран текста с предварительным сбросом экрана
; и переключением формата
      MOV #TEX,R1        Адрес начала текста
      CLR R2             Конец текста - нулевой байт
      EMT 20             Вывод текста
      HALT               Останов
TEX:  .B:14,233.A:¬      Привет от БК-0010,
      .A:¬ лучшего в мире компьютера!\
      END
```

2)
```
; Программа "мигания" экрана с выдачей звуковых сигналов
0:    MOV #COM,R1        Адрес начала команд
      CLR R2             Признак конца команд - 0
      EMT 20             Выдача команд звука и мигания
      EMT 6              Ждать нажатия клавиши
      BR 0               Программа зациклена
COM:  .B:7,7,7,235,7,7,7,235.E
      END
```

3)
```
; Перемещаемая программа вывода текста
0:    MOV PC,R1          Занесение физического
      ADD (PC)+,R1        адреса метки
      .@TEX+2             TEX в R1
      CLR R2             Конец текста - нулевой байт
      EMT 20             Вывод текста
      EMT 6              Ожидание нажатия клавиши
      BR 0               Программа зациклена
TEX:  .A:¬ Эта программа может работать по любому адресу,
      .A:¬ и не нуждается в компоновке!\
      END
```

4)
```
; Программа ввода текста в буфер с его последующим выводом
      MOV #TEX,R1        Адрес буфера текста
      MOV #5377,R2       Конец текста - 377 байт или "ВВОД"
      EMT 10             Ввод текста с клавиатуры
      MOV #TEX,R1        Адрес начала текста
      MOV #5377,R2       Ограничители текста
      EMT 20             Вывод текста
      HALT               Конец
TEX:  .+377              Буфер текста 377 байт
      END
```

5)
```
; Программа выдачи нескольких последовательных
; фрагментов текста
      MOV #TEX,R1     Адрес начала текста
      CLR R2          Признак конца - нулевой байт
      EMT 20          Выдать первый текст
      EMT 6           Ждать нажатия клавиши
      CLR R2
```

```
        EMT 20          Выдать второй текст
        EMT 6
        CLR R2
        EMT 20          Выдать третий текст
        EMT 6
        CLR R2
        EMT 20          Выдать четвертый текст
        HALT
TEX:  .A:¬ Нажимайте клавиши!\
      .A:¬ Вы нажали первую клавишу.\
      .A:¬ А теперь - вторую!\
      .A:¬  Наконец-то, последняя клавиша! До свидания.
      .E
      END
```

Заметим, что в последнем примере при вызове EMT 20 используется не только входная информация, но и выходная — для выдачи последующих фрагментов текста их адрес в R1 уже не заносится, так как там содержится адрес байта, следующего за концом предыдущего текста.

Нельзя не упомянуть и о чрезвычайно оригинальном и очень полезном на практике способе вывода текстовых сообщений, также использующем выходную информацию EMT 20. Этот способ предложен учеником X класса Денисом Батранковым (Московская обл.). Поясним его смысл на примере:

```
        JSR R1,PRI              Печать текста
TE1:  .A: Привет!¬
        .E
        JSR R1,PRI              Печать текста
TE2:  .A: Два привета!¬
        .E
        JSR R1,PRI              Печать текста
TE3:  .A: Три привета!¬
        .E
        ;...............
        ; Подпрограмма печати
PRI:  CLR R2                    Признак конца текста
        EMT 20                  Вывод текста
        INC R1                  Привести содержимое R1
        BIC #1,R1               к четному значению
        RTS R1                  Выход
        ;...............
```

Здесь используется та особенность, что при обращении к подпрограмме JSR RN,MET в регистр RN заносится адрес возврата, т. е. адрес следующей после оператора JSR команды, а там у нас — начало текста. В подпрограмме выполняется EMT 20 по содержимому R1 (с предварительной установкой признака окончания по нулевому байту или слову), а затем содержимое R1 (в котором после выполнения EMT 20 хранится адрес следующего байта после текста) приводится к ближайшему большему четному адресу, т.е. в R1 оказывается адрес возврата из подпрограммы на следующую после текста команду. Данный способ дает заметную экономию памяти, если текстовых сообщений в программе много, и при этом обеспечивает ее перемещаемость. К его недостаткам можно отнести необходимость помещать текстовые сообщения в программе ТОЛЬКО по ходу обращений к ним (хотя в большинстве случа-

ев это как раз и требуется) и недопустимость использования в качестве ограничителя текста символа «\» (допускается только оператор .E). В остальном же этот способ имеет неоспоримые преимущества перед всеми остальными, включая и то, что после вывода текста содержимое R1 восстанавливается, как при всяком выходе из подпрограммы по RTS RN. (Кстати, обозначать метками начала текстовых сообщений, как это сделано в нашем примере для наглядности, совсем не обязательно. Это может пригодиться, только если потребуется еще раз вывести то же сообщение в другом месте программы.)

Контрольные вопросы и задания

1. Напишите программу, которая бы «отслеживала» положение символьного курсора по горизонтали, перемещая наравне с ним стрелку в служебной строке.

```
0: EMT 6       Ввод символа (команды)
   EMT 16      Вывод символа на экран
   EMT 26      Позиция курсора (R1 = X)
   CLR R0      Очистка
   EMT 22       служебной строки
   MOV #33,R0 Код "стрелка вниз"
   EMT 22      Символ в служебную строку (позиция - в R1)
   BR 0        Программа зациклена
   END         Конец
```

2. Напишите программу, которая бы, определяя установленный режим дисплея, «приказывала» пользователю печатать символы только в русском регистре.

```
0:    EMT 6        Ввод символа
      TSTB @#43    Регистр "РУС"?
      BNE 1        Да - печать символа,
      MOV #TEX,R1   иначе - выдать
      CLR R2        предупреждение
      EMT 20        о вводе не русского символа
      BR 0         К началу
1:    EMT 16       Символ на экран
      BR 0         Программа зациклена
TEX:  .A:¬ Нажмите клавишу "РУС"!¬
      .E
      END          Конец
```

3. Напишите простейший копировщик — программу, которая бы после запуска позволяла скопировать последний загруженный с магнитной ленты по «естественному» адресу файл без задания его имени, адреса и длины с клавиатуры. Программа должна быть перемещаемой, с целью размещения ее в зоне ОЗУ, не занимаемой копируемым файлом.

Подсказка: блок параметров EMT 36 для записи не обязательно начинается с адреса 320, а программа должна получиться ОЧЕНЬ короткой — всего несколько операторов.

```
MOV #344,R1    Адрес блока параметров послед-
               ; него загруженного файла минус 2
               ; используем как блок параметров файла
               ; для записи
MOV #2,@R1     Команда "Запись"
```

```
EMT 36          Запись файла
HALT            Конец
END
```

4. Что произойдет, если после запуска следующей программы нажать любую символьную клавишу:

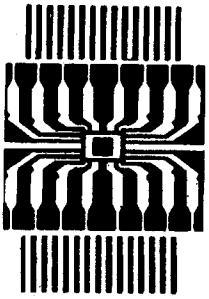
```
MTPS #340
WAIT
EMT 6
EMT 16
```

— Ничего, так как приоритет ЦП установлен равным 7, а приоритет клавиатуры — 4. Программа остановится на команде WAIT, и ее работа будет продолжена после нажатия на клавишу «СТОП».

(Продолжение следует)

ВНИМАНИЕ! ОПЕЧАТКА

В таблице, содержащей сведения об источниках, приоритетах и векторах прерываний, приведенной во внутризаводской документации завода «Экситон» и повторенной в статье «Техническое описание БК-0010» (№ 3 за 1994 г., табл.4), а также в примечании редактора к статье Ю. А. Зальцмана «МикроЭВМ БК-0010. Архитектура и программирование на языке ассемблера» (№ 2 за 1994 г., табл. 2), в графе «Приоритеты» ошибочно указаны порядковые номера. Приносим читателям свои извинения.



Уважаемые читатели! Вы держите в руках исходный текст 326-й прошивки в блоке КНГМД с подробными комментариями. Он получен дизассемблированием программы, зашитой в микросхеме КР1801РЕ2—326. Этот текст появился на свет благодаря дизассемблеру DIZAKS4 v4 17.04.88 (KSA corp.), текстовому редактору EDALT3E (фирма ALTEC) и конечно, дисководу.
Основная работа по дезассемблированию и написанию комментариев выполнена К. В. Гаршениным (г. Москва), исправления и дополнения внесли А. А. Войлуков , А. С. Рузаков и Р. Аскеров, любезно предоставивший эти материалы редакции для публикации. Редакция благодарит также В. И. Хмелькова из г. Иошкар-Олы за поправки, касающиеся, использования драйвера расширенной арифметики из прошивки КНГМД.

Описание прошивки ПЗУ КНГМД КР1801РЕ2-326

Сегодня все больше и больше пользователей приобретают дисководы и подключают их к БК. Но мало кто знает, как работает драйвер дисковода. (На БК стало дурной традицией сделать что-нибудь хорошее, а всю информацию «как и с чем это едят» припрятать или передать очень узкому кругу лиц. Кстати, это повелось с начала выпуска самой БК. Ее пустили в продажу, не включив в документацию ни схем, ни описания принципов работы. А ведь если бы это было сделано, то еще в 1989 г. мы имели бы на БК дисковод, расширение памяти, и, следовательно, много хороших программ. То же произошло с контроллером дисковода.) Имеются, конечно, какие-то обрывки, но полной информации нет. Вот почему автор принялся за эту каторжную работу. Понять и прокомментировать драйвер было довольно сложно, так как в своем распоряжении он имел всего лишь распределение ячеек в рабочей области драйвера, известное по руководству системного программиста для БК-0011.

Конечно, приведенный здесь ассемблерный листинг — это не «исходник», но надеемся, что с его помощью вы сможете кое-что узнать об этой прошивке и составить о ней собственное мнение. Думаем также, вы поймете, что эта программа далека от совершенства, многие операции можно было сделать лучше и в некоторых случаях применялись не самые хорошие алгоритмы.

Самое главное, что теперь это положение уже не исправить, так как данная прошивка стала стандартной для БК и ее изменение приведет к несовместимости с ней некоторых программ. (Хотя, конечно, это относится только к программам, вызывающим подпрограммы прошивки не через стандартные точки входа.)

Регистры КНГМД

Регистр состояния КНГМД (@#177130).

Формат регистра по чтению

Разряд	Назначение
00	Признак «дорожка 00»
01	Признак готовности накопителя
02	Запрет записи (дискета заклеена)
07	Требование съема информации с регистра данных или записи в него нового кода
14	В режиме чтения: признак записи циклического контрольного кода на диск
15	Датчик индексного отверстия
3—6	Не используются
8—13	Не используются

Формат регистра по записи

Разряд	Маска	Назначение
00	#1	Выбор накопителя с номером 0
01	#2	Выбор накопителя с номером 1
02	#4	Выбор накопителя с номером 2
03	#10	Выбор накопителя с номером 3
04	#20	Включение электродвигателя
05	#40	Выбор головки накопителя (стороны диска): 0 — верхняя, 1 — нижняя
06	#100	Направление шага
07	#200	Шаг
08	#400	Признак «НАЧАЛО ЧТЕНИЯ»
09	#1000	Признак «ЗАПИСЬ МАРКЕРА»
10	#2000	Включение схемы прекоррекции
11—15	—	Не используются

Регистр данных КНГМД (@#177132). Все разряды регистра доступны по чтению и записи.

Драйвер НГМД

Драйвер НГМД размещен в ПЗУ, находящемся в контроллере НГМД, и доступен при подключении контроллера к микроЭВМ. Для работы драйвера необходима область ОЗУ размером 66 байт для размещения параметров и рабочих ячеек.

Формат рабочей области драйвера:

Смещение от начала области	Длина, байт	Символическое имя	Начальное значение	Назначение ячейки
0	2	CSRW	000000	Копия по записи регистра состояния КНГМД
2	2	CURTRK	—	Адрес текущей дорожки (адрес одного из последующих байтов)
4	4	TRKTAB	177777, 177777	Таблица текущих дорожек (*)
10	2	TDOWN	020000	Задержка опускания головки
12	2	TSTEP	—	Задержка перехода с дорожки на дорожку
14	1	TRKCOR	000036	Дорожка начала предкомпенсации
15	1	BRETRY	000030	Число попыток повтора при ошибке
16	1	FLAGS	000000	Рабочая ячейка драйвера (**)
17	1	FILLB	—	Код заполнения при форматировании
20	2	FLGPTR	—	Указатель на байт признаков (один из следующих байтов)
22	4	FLGTAB	000000, 000000	Таблица признаков (***)
26	2	ADDR	—	Адрес начала массива данных в ОЗУ (обязательно четный!)

Смещение от начала области	Длина, байт	Символическое имя	Начальное значение	Назначение ячейки
30	2	WCNT	—	Количество слов для пересылки
32	1	SIDE	—	Номер стороны диска
33	1	TRK	—	Номер дорожки
34	1	UNIT	—	Номер привода
35	1	SECTOR	—	Номер сектора
36	2	WRTVAR	(****)	Значение, записываемое при форматировании
40	2	MARKER	(****)	Буфер маркера при записи
42	2	FREE	(****)	Длина пустого остатка сектора
44	2	INTIME	(****)	Счетчик длительности индекса
46	2	BUF4	(****)	Буфер для сохранения вектора @#4
50	2	BUFSP	(****)	Буфер для сохранения SP
52	2	BUFPSW	(****)	Буфер для сохранения PSW
54	1	CRETRY	(****)	Счетчик повторов при ошибке
55	1	TURNS	(****)	Число оборотов диска при поиске сектора
56	1	SECRET	(****)	Число повторных попыток поиска сектора
57	1	ERRNUM	(****)	Буфер для номера ошибки
60	2	MAXSEC	000012	Число секторов на дорожке
62	2	HOLTIN	—	Время задержки после индекса (до первого сектора)
64	2	SECLEN	000400	Длина сектора в словах (*****)

Примечания

(*): старший бит каждого байта при установке в «1» задает возврат на соответствующем дисководе к дорожке 00 перед операцией позиционирования (для случая, когда положение головки заранее неизвестно).

(**): флаги (биты) имеют следующие значения:

0 (001) — разрешение читать/записывать скрытые данные,

2 (004) — двигатель уже был включен перед операцией,

4 (020) — признак записи информации на диск,

5 (040) — текущее состояние индекса,

6 (100) — отмена двойного шага при возврате на 00.

(***): таблица признаков содержит по одному байту на каждый привод (дисковод) и используется для обработки нестандартных дисков. В каждом байте задействовано четыре бита для указания параметров операции:

0 — при установке в «1» драйвер производит пропуск дорожек при поиске, т.е. фактически обрабатываются лишь четные дорожки. Эта возможность используется для обработки на 80-дорожечном дисководе 40-дорожечных дисков,

1 — при установке в «1» драйвер производит обращение только к нижней стороне диска. Эта возможность используется при подключении одностороннего дисковода, а также при обработке на двустороннем дисководе односторонних дисков,

2 — при установке в «1» обрабатываются сектора длиной 2000 байт, иначе — 1000 байт **(Внимание! Работа с секторами длиной 2000 байт реализована неверно!)**,

3 — при установке в «1» начало первого сектора задается задержкой HOLTIN от индекса, в противном случае — своим заголовком.

(****): значения заносятся в процессе работы драйвера.

(*****): внимание! Следите, чтобы при записи ячейка SECLEN содержала соответствующее значение, в противном случае портится разметка диска!

Перед самым первым вызовом драйвера нужно проинициализировать рабочую область, вызвав специальную подпрограмму инициализации.

После завершения операций обмена с диском драйвер оставляет двигатель дисковода включенным. Для остановки двигателя программа пользователя должна обнулить ячейки @#177130 и CSRW. (Просьба не выключать двигатель командой RESET: подумайте о тех, кто имеет БК-0010 с расширенным ОЗУ 32 Кб!)

Диагностика ошибок в драйвере НГМД

При возникновении ошибки бит С в слове состояния процессора устанавливается в 1, и в байт @#52 помещается код ошибки:

1 — при чтении: ошибка CRC в зоне данных, при записи: установлена защита записи,
2 — ошибка CRC в зоне заголовка сектора,
3 — нет позиционирования на нулевую дорожку,
4 — ошибка позиционирования,
5 — не найден сектор,
6 — нет диска или диск не вращается,
7 — прерывание по вектору @#4 во время выполнения операции,
10 — не найден адресный маркер,
11 — не найден маркер данных,
12 — недопустимые параметры команды или нестандартный формат диска,
13 — попытка считывания скрытых данных без соответствующего разрешения.

Рабочие подпрограммы драйвера

Драйвер имеет несколько точек входа (указаны адреса обращения к подпрограммам):

160000 — автоматическая загрузка,
160002 — загрузка с выбранного привода,
160004 — чтение-запись по номеру блока,
160006 — чтение-запись по номеру сектора,
160010 — инициализация рабочей области драйвера,
160012 — форматирование дорожки.

Автоматическая загрузка

Вызов: **JSR PC,@#160000**. (Можно также произвести загрузку непосредственно из пускового монитора: «СТОП» С160000 «ВВОД».)

Входиые параметры отсутствуют.

Производится попытка загрузить операционную систему с привода 0. При неудаче драйвер пытается загрузить систему с привода 1. Если все попытки оказались неудачными, драйвер возвращает управление с установленным битом С в слове состояния процессора.

Алгоритм загрузки:

1) инициализируется рабочая область драйвера в ОЗУ (базовый адрес 2000) с параметрами для НГМД 6022,

2) в ОЗУ, начиная с адреса 1000, считывается сектор 1 дорожки 0 выбранного дисковода;

3) если в приводе нет диска, дисковод неисправен или его замок не закрыт, если диск не отформатирован, а также при ошибке чтения или отсутствии начального загрузчика попытка загрузки считается неудачной.

При успешной загрузке драйвер передает управление по адресу 1000, при этом в R0 помещается номер привода, с которого производилась загрузка.

(Драйвер прошивки 253 пытается загрузиться со всех приводов 0—3. Поскольку он использует функции EMT с номерами 0, 16, и 122, их необходимо перехватывать: функция EMT 112 заменяется на NOP, EMT 0 должна восстанавливать стек.)

Загрузка с выбранного привода

Вызов: **MOV #N,R0**
JSR PC,@#160002

Действие аналогично предыдущей программе, но загрузка начинается сразу с выбранного привода. При неудаче попыток загрузиться с других приводов не предпринимается. **ИЗ ВСТРОЕННЫХ МОНИТОРОВ БК ВЫБОРОЧНАЯ ЗАГРУЗКА НЕВОЗМОЖНА.**

Чтение-запись логического блока

Вызов: **JSR PC,@#160004 ;BCC WORK или BCS ERROR**

Входные параметры:

R0 — номер блока на диске,
R1 — длина пересылаемого массива в словах,
R2 — начальный адрес массива данных,
R3 — базовый адрес рабочей области драйвера,
UNIT — номер привода (здесь и далее «нерегистровые» обозначения относятся к ячейкам рабочей области).

Блок является логической единицей информации на внешних носителях, его длина равна 1000_8 байт. В используемом формате записи длина блока совпадает с объемом сектора. Первый сектор нулевой дорожки нижней стороны соответствует блоку 0. Вычисление номеров сектора и дорожки по номеру блока производится с учетом установленных признаков в таблице FLGTAB. Алгоритм этого вычисления совпадает с алгоритмом драйвера MY для микроЭВМ ДВК.

Длина массива данных задает количество слов, подлежащих пересылке, а также тип операции: положительное число — чтение, отрицательное — запись. (При записи количество пересылаемых слов равно абсолютному значению длины.) Если заданное количество слов превышает размер сектора, осуществляется автоматический переход к следующему сектору, а если необходимо, то и на другую сторону и дорожку. Если при записи число слов не кратно объему сектора, то остаток последнего сектора заполняется нулями.

Перед самым первым вызовом драйвера поля CSRW, TRKTAB и FLAGS в рабочей области должны быть заполнены значениями, указанными в таблице, и не должны изменяться программой пользователя. Поля TDOWN, TSTEP, TRKCOR, BRETRY и FLGTAB рекомендуется заполнять значениями, указанными в таблице, однако они могут быть изменены в зависимости от конкретных условий.

Возврат из драйвера осуществляется командой RTS PC, содержимое регистров (кроме R3) не сохраняется.

Возникновение прерывания по вектору 4 при записи на диск приводит к появлению сбоя в секторе, в который производилась запись. Сбой может быть устранен повторной записью этого сектора. **ЕЩЕ РАЗ НАПОМИНАЕМ: НАЖИМАТЬ КЛАВИШУ «СТОП» ВО ВРЕМЯ ЗАПИСИ НА ДИСК КРАЙНЕ НЕЖЕЛАТЕЛЬНО!**

Чтение-запись по номеру сектора и дорожки

Вызов: **JSR PC,@#160006**

Входные параметры:

R3 — адрес рабочей области,
ADDR — начальный адрес массива данных,
WCNT — длина массива данных,
SIDE — номер стороны (0 — нижняя, 1 — верхняя),

TRK — номер дорожки,
UNIT — номер привода (0—3),
SECTOR — номер сектора (1—12).

Действие аналогично предыдущей подпрограмме.

Инициализация рабочей области

Вспомогательная подпрограмма, служащая для облегчения написания программ, использующих драйвер НГМД.

Вызов: **JSR PC,@#160010**

Входной параметр:
R3 — базовый адрес рабочей области.

Заполняются следующие поля рабочей области:
CSRW — 0,
TRKTAB — 177777, 177777,
TDOWN — 10000,
TSTEP — 10000,
TRKCOR — 32. (для прошивки 253 — 30.),
BRETRY — 20. (для прошивки 253 — 30.),
FLAGS — 0,
FLGTAB — 0, 0,
MAXSEC — 12 (только для прошивки 326).

Примечание. Чтобы отличать десятичные числа от восьмеричных, после них ставится десятичная точка. Восьмеричные же значения точкой не заканчиваются.

Форматирование дорожки

Вызов: **JSR PC,@#160012**

Входные параметры:
R3 — базовый адрес рабочей области,
FILLB — код заполнения секторов при форматировании,
SIDE — номер стороны,
TRK — номер дорожки,
UNIT — номер привода.

Форматируется указанная дорожка с указанной стороны, поле данных всех секторов заполняется кодом FILLB.

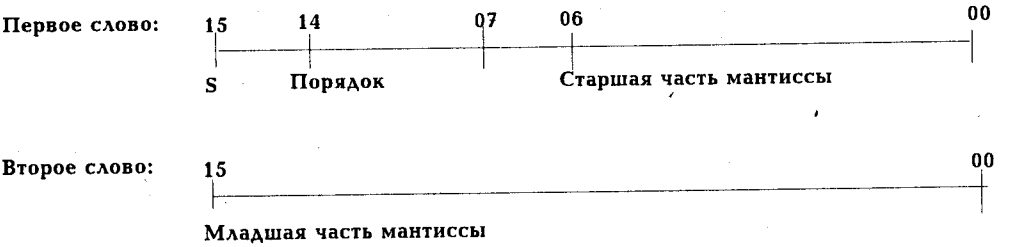
Команды расширенной арифметики

В прошивке 326 помимо драйвера дисковода содержится эмулятор команд расширенной арифметики процессора К1801ВМ2, т.е. на БК возможно использование программного обеспечения, написанного для этого процессора.

Для организации работы с блоком расширенной арифметики необходимо перед ее использованием в программе проинициализировать эмулятор. Инициализация производится занесением в ячейку @#10 числа 160016. При поступлении кода команды расширенной арифметики происходит прерывание по вектору @#10 и ее обработка.

Мнемоника команд расширенной арифметики здесь аналогична реализованной в ассемблерах MICRO В. С. Коренкова и TURBO А. М. Надежина:
MUL S,Rn — деление,
DIV S,Rn — умножение,
FMUL Rn — умножение чисел с плавающей запятой,
FDIV Rn — деление с плавающей запятой,
FADD Rn — сложение с плавающей запятой,
FSUB Rn — вычитание с плавающей запятой,
ASH S,Rn — арифметический сдвиг на заданное число бит,
ASHC S,Rn— арифметический сдвиг двойного слова на заданное число бит.

Формат чисел с плавающей запятой



Формат чисел с плавающей точкой в данном случае совпадает с принятым в вильнюсском БЕЙСИКе БК-0010.01 для чисел с одинарной точностью (признак типа «!»). Подробнее об этом формате и о перекодировании в него десятичных чисел читайте в статье на с. 42. (Хотя там говорится о числах двоичной точности, правила перекодирования одни и те же, но для использования функций из прошивки КНГМД нужно отсечь два последних слова мантиссы.)

Описание команд расширенной арифметики

DIV S,Rn (код 071RSS*)

Деление 32-разрядного слова RnRn+1 на число или содержимое регистра. (Здесь и далее обозначение RnRn+1 означает последовательную пару регистров: R0 иR1, R1 и R2, R2 и R3 и т.д.) Регистр результата (Rn) в команде должен быть четным.

```
MOV #75.,R1   ;Деление числа 75. на 10.
CLR R0        ;R0 — результат, R1 — остаток
DIV #10.,R0
```

MUL S,Rn (070RSS)

Умножение содержимого регистра на число. Если номер регистра нечетный, сохраняется младшая часть результата. Регистр результата (Rn) должен быть нечетным.

```
MOV #7,R1     ;умножение 7*10.=70. в регистре R1
MUL #10.,R1
```

Если бы в команде MUL был указан четный регистр, то результат был бы помещен в пару регистров RnRn+1.

ASH S,Rn (072RSS)

Арифметический сдвиг регистра вправо-влево на (-32...+32) позиции в зависимости от значения бита 15 числового аргумента. При «1» в бите 15 — сдвиг вправо, при «0» — влево.

```
ASH #5,R1     ;сдвиг регистра R1 на 5 позиций влево
```

При наличии «1» в пятнадцатом бите величина сдвига кодируется в дополнительном коде: ASH # -5,R1. Сдвиг арифметический, т.е. знаковый разряд распространяется вправо.

ASHC S,Rn (073RSS)

Арифметический сдвиг двойного слова, причем регистр с нечетным номером содержит младшую часть слова, а с четным — старшую. Остальное аналогично команде ASH.

---

* В обозначении кода: R — номер регистра (одна восьмеричная цифра), SS — источник (две восьмеричные цифры с учетом адресации).

FADD Rn (07500R)

Сложение чисел с плавающей запятой. Регистр указывает на адрес нахождения аргументов A и B. Результат помещается на место аргумента B, содержимое аргумента A сохраняется (см. пример к команде FSUB).

FSUB Rn (07501R)

Вычитание чисел с плавающей запятой. Результат в B, содержимое A сохраняется.

```
        FSUB R5         ;R5 указывает на адрес MET
        .....
MET:    .#A1            ;два слова аргумента A
        .#A2
        .#B1            ;два слова аргумента B
        .#B2
```

FMUL Rn (07502R)

Умножение чисел с плавающей запятой. Результат на месте аргумента B, содержимое A сохраняется.

FDIV Rn (07503R)

Деление чисел с плавающей запятой. Результат на месте аргумента B, содержимое A сохраняется. Если делитель (B) равен нулю, то результат в стек не записывается.

Ассемблерный листинг прошивки ПЗУ КНГМД

```
        BR    BOOT0     ;Автоматическая загрузка          160000
        BR    BOOT1     ;Загрузка с выбранного привода    160002
        BR    RWBLK     ;Чтение-запись по номеру блока    160004
        BR    RWSEC     ;Чтение-запись по номеру сектора  160006
        BR    INIT      ;Инициализация рабочей области    160010
        JMP   FORMAT    ;Форматирование дорожки           160012
        JMP   EXT       ;Расширенная арифметика           160016

            *** 160022 П/п автоматической загрузки ***

BOOT0:  CLR   R0        ;Начнем с нулевого привода
        CALL  BOOT      ;Пробуем загрузиться
        MOV   #1,R0     ;Там не получилось, попробуем с первого
        CALL  BOOT      ;Загружаем...
        BR    STOP      ;Загрузчика нигде нет, выходим

           *** 160042 Загрузка с выбранного привода ***

BOOT1:  MOV   #1000,SP  ;Установим стек
        CALL  BOOT      ;Попробуем загрузиться

      *** 160052 Остановка двигателя НГМД и выход из п/п ***

STOP:   CLR   (R3)
        CLR   @#177130
        CLR   @#177760
        RET

        *** 160066 П/п загрузки и запуска загрузчика ***

BOOT:   MOV   #2000,R3  ;Установим адрес рабочей области
        CALL  INIT      ;    и инициализируем ее
        MOVB  R0,34(R3) ;Занесем номер привода
        CLR   R0        ;Читаем нулевой блок
```

```
        MOV  #400,R1        ;   длиной 400 слов (1000 байт)
        MOV  #1000,R2       ;   с адреса 1000
        CALL RWBLK          ;Прочитаем...
        BCS  1$             ;   не получилось - выход
        CMP  @#1000,#240    ;Загрузчик начинается с NOP?
        BNE  1$             ;Нет - выход
        CMP  @#1002,#5      ;Вторая команда - RESET?
        BEQ  1$             ;Да - диск не системный, выход
        MOVB 34(R3),R0      ;Вспомним, откуда это мы загрузились
        CALL @#1000         ;Запускаем загрузчик...
1$:     RET                 ;Вот и все

          *** 160154 Чтение-запись по номеру сектора ***

RWSEC:  BR   SUBRW

        *** 160156 П/п инициализации рабочей области ***

INIT:   CLR  (R3)           ;Очистим копию РС КНГМД
        MOV  #177777,4(R3)  ;Заполним таблицу текущих дорожек
        MOV  #177777,6(R3)
        MOV  #23420,10(R3)  ;Время опускания головки
        MOV  #23420,12(R3)  ;Время перехода дорожки
        MOVB #40,14(R3)     ;Дорожка начала предкомпенсации
        MOVB #24,15(R3)     ;Число повторов при ошибке
        CLRB 16(R3)         ;Очистим флаги
        CLR  22(R3)         ;   и таблицу признаков
        CLR  24(R3)
        MOV  #12,60(R3)     ;Занесем число секторов на дорожке
        RET

        *** 160250 П/п чтения-записи по номеру блока ***

   Входные параметры:
      R0 - номер блока
      R1 - длина массива в словах (отрицательная - запись)
      R2 - адрес начала массива
      R3 - адрес рабочей области

RWBLK:  MOV  #10,R4
        MOVB 60(R3),R5      ;число секторов на дорожке
        SWAB R5             ;делим ст. байт R0 на ст. байт R5
1$:     CMP  R5,R0
        BHI  2$
        SUB  R5,R0
        SEC
2$:     ROL  R0             ;Частное накапливается в мл. байте R0
        SOB  R4,1$
        MOVB R0,R4          ;В мл. байте номер дорожки, сохраним его
        CLRB R0
        SWAB R0             ;Получили остаток от деления
        INC  R0             ;Номер сектора на единичку больше
        CLRB 32(R3)         ;Пока мы на нижней стороне
        CMP  R0,60(R3)      ;Этот сектор за пределами дорожки?
        BLE  3$             ;Нет - полный порядок
        INCB 32(R3)         ;Да - значит, он на другой стороне
        SUB  60(R3),R0      ;   и номер у него другой
3$:     CALL SETPTR         ;Установим указатель на байт признаков
        BITB #2,@20(R3)     ;Диск односторонний?
        BEQ  4$             ;Нет - оставим все как есть
        ASRB 32(R3)         ;Да - сторона всегда 0,
        ROLB R4             ;   а дорожка уже другая
4$:     MOVB R4,33(R3)      ;Запишем результаты: дорожка,
        MOVB R0,35(R3)      ;   сектор,
```

```
        MOV  R1,30(R3)      ;   число слов,
        MOV  R2,26(R3)      ;   адрес буфера

      *** 160372 П/п чтения-записи по номеру сектора ***

   Входные параметры:
      R3 - базовый адрес рабочей области
      ADDR, WCNT, SIDE, TRK, UNIT, SECTOR - соответственно

SUBRW:  MOV  SP,50(R3)      ;Сохраняем SP,
        MFPS 52(R3)         ;   PSW,
        MOV  #4,R0
        MOV  (R0),46(R3)    ;   @#4 в соответствующих буферах
        MOV  PC,(R0)        ;Переназначаем вектор @#4 на себя,
        ADD  #1076,(R0)     ;   теперь он равен 161512
        TSTB 35(R3)         ;Какой сектор нужен?
        BEQ  ERR12          ;Нулевых у нас нет, ошибка 12
        CMPB 35(R3),60(R3)  ;Сектор больше максимального?
        BGT  ERR12          ;Таких тоже нет, ошибка 12
        CLRB @#52           ;Пока ошибок нет
        BICB #30,16(R3)     ;Пока не записываем
        TST  30(R3)         ;Сколько слов будем пересылать?
        BEQ  QUIT           ;0 - ничего делать не надо, выход
        BPL  READ           ;больше 0 - будем читать

              *** 160460 П/п записи массива ***

        NEG  30(R3)         ;Нормализуем длину
        BISB #20,16(R3)     ;Установим признак записи
        MOV  #175641,40(R3) ;Запомним маркер данных
        BITB #1,16(R3)      ;Записываем скрытно?
        BEQ  1$             ;Нет - идем дальше
        MOV  #174241,40(R3) ;Да - запомним специальный маркер
1$:     CALL ENGINE         ;Включим двигатель, запомним привод
        BIT  #4,(R4)        ;Защита записи есть?
        BEQ  MAIN           ;Нет - продолжить работу
        MOV  #1,R0          ;Есть - ошибка 1
        BR   ERR

              *** 160536 П/п чтения массива ***

READ:   CALL ENGINE         ;Включим двигатель, запомним привод
        MOV  #400,64(R3)    ;Сектора стандартной длины
        BITB #4,@20(R3)     ;А они ли нам нужны?
        BEQ  MAIN           ;Да - продолжить работу
        MOV  #1000,64(R3)   ;Нет - секторы удвоенной длины

         *** Основная часть п/п чтения/записи массива ***

MAIN:   CALL GOTRK          ;Позиционируем на нужную дорожку
        CALL CORR           ;Включим схему прекоррекции
        MOVB 15(R3),54(R3)  ;Число попыток - в буфер

              *** Получение заголовка сектора ***

        MOVB #10,57(R3)     ;Предполагаем ошибку 10
        MOVB #24,55(R3)     ;На операцию отводим 20. оборотов
1$:     CALL FINDH          ;Найдем маркер заголовка
        BCC  3$             ;Нашли - идем дальше
2$:     MOVB 57(R3),R0      ;Не нашли - выходим с ошибкой
        BR   ERR
3$:     TSTB (R4)           ;Информация получена?
        BPL  3$             ;Нет - подождем
```

```
        MOV  (R5),R0        ;Теперь прочитаем ее
4$:     TSTB (R4)           ;Хотим еще информации!
        BPL  4$             ;Дождемся ее появления
        MOV  (R5),R1        ;    и заберем ее
        CALL TSTCRC         ;Проверим CRC
        BNE  5$             ;Ошибки нет - идем дальше
        MOVB #2,57(R3)      ;Ошибка 2!
        DECB 54(R3)         ;Остались еще попытки?
        BNE  1$             ;Да - повторим чтение
        BR   2$             ;Видно, не судьба! Выходим с ошибкой
5$:     BIC  #177774,R1     ;Выделяем код длины сектора
        BITB #4,@20(R3)     ;Какие сектора нам были нужны?
        BNE  LONG           ;Длинные - посмотрим...
        CMPB R1,#2          ;Короткие. А на диске какие?
        BEQ  WORK           ;Тоже короткие. Порядок, идем дальше
        DECB 54(R3)         ;Как там дела с попытками?
        BNE  1$             ;Есть еще - повторим все сначала

              *** 160724 Выход с ошибкой ***

ERR12:  MOV  #12,R0         ;Выходим с ошибкой 12
ERR:    JMP  ERROR          ;Идем на обработку и выход
QUIT:   JMP  EXIT           ;Идем на нормальный выход

LONG:   CMPB R1,#2          ;Нужны длинные сектора, а какие здесь?
                            ;    ОШИБКА: На самом деле код длинных
                            ;секторов (1024 байта на сектор) - 3, а не 2
        BNE  ERR12          ;Длинные - выходим с ошибкой 12

              *** Собственно п/п чтения ***

WORK:   MOVB 15(R3),54(R3) ;Восстановим счетчик по BRETRY
        MOVB #6,56(R3)      ;На поиск сектора - 6 попыток
        CLR  42(R3)         ;Остаток сектора отсутствует
        CMP  30(R3),64(R3) ;Читать надо больше одного сектора?
        BHI  1$             ;Да - идем читать
        MOV  64(R3),42(R3)
        SUB  30(R3),42(R3) ;Вычислим длину остатка
1$:     MOVB #5,57(R3)      ;Прогнозируем ошибку 5
2$:     MOVB #24,55(R3)     ;На операцию отводим 20. оборотов
3$:     CALL FINDH          ;Ищем маркер заголовка
        BCC  4$             ;Нашли - идем дальше
        MOVB 57(R3),R0      ;Кончились обороты - выходим
        BR   ERR            ;    с ошибкой
4$:     TSTB (R4)           ;Готовность данных есть?
        BPL  4$             ;Нет - подождем
        CMP  32(R3),(R5)    ;Нужная дорожка и сторона?
        BEQ  5$             ;Да - идем дальше
        DECB 56(R3)         ;Минус одна попытка
        BEQ  6$             ;Кончились - выход с ошибкой
        CALL GOTO00         ;Позиционируем на дорожку 00,
        CALL GOTRK          ;    вернемся на нужную
        BR   3$             ;    и попытаемся еще раз
6$:     MOV  #4,R0          ;Выходим с ошибкой 4
        BR   ERR
5$:     TSTB (R4)           ;Ждем готовности данных
        BPL  5$             ;Не готовы - повторять...
        MOV  (R5),R1        ;Получим номер сектора,
        SWAB R1             ;    поместим его в младший байт
        CMPB 35(R3),R1      ;Этот сектор ищем?
        BEQ  7$             ;Да - будем его читать
        MOV  64(R3),R0      ;Нет - возьмем длину данных,
        ADD  #27,R0         ;    прибавим длину заголовка
        CALL FICT           ;    и пропустим их
        BR   3$             ;Повторим процедуру еще раз
```

```
7$:     CALL TSTCRC         ;Проверим CRC
        BNE  8$             ;Совпала - идем дальше
        MOVB #2,57(R3)      ;Если что, это будет ошибка 2
        DECB 54(R3)         ;Одной попыткой меньше
        BNE  2$             ;Еще остались - поищем снова
        MOVB 57(R3),R0      ;Нет - выходим с ошибкой
        BR   ERROR
8$:     MOV  64(R3),R1      ;Получим длину сектора
        SUB  42(R3),R1      ;Вычислим длину используемой части

              *** 161176 Чтение сектора ***

        BITB #20,16(R3)     ;Будем записывать?
        BEQ  9$             ;Нет - значит, читать
        JMP  WRSEC          ;Да - идем на запись сектора
9$:     CALL FINDS          ;Ищем зону синхронизации
        TSTB 56(R3)         ;Нашли?
        BEQ  11$            ;Нет - пробуем снова
        CALL STREAD         ;Начинаем чтение!
        MOV  #1130,R0       ;
10$:    TSTB (R4)           ;Данные готовы?
        BMI  12$            ;Да - посмотрим их
        SOB  R0,10$         ;Больше ждать нельзя?
11$:    MOVB #11,57(R3)     ;Да - прогнозируем ошибку 11
        DECB 54(R3)         ;Одной попыткой меньше
        BNE  2$             ;Попытки есть - повторим операцию
        MOVB 57(R3),R0      ;Кончились - выйдем с ошибкой
        BR   ERROR
12$:    TST  (R5)           ;Пропустим маркер A1A1
13$:    TSTB (R4)           ;Готовность данных есть?
        BPL  13$            ;Нет - ждем
        MOV  (R5),R0        ;Прочитаем данные
        CMP  #120773,R0     ;Обычный сектор?
        BEQ  14$            ;Да - идем дальше
        CMP  #120770,R0     ;Скрытый сектор?
        BNE  11$            ;Что-то иное - ошибка
        BITB #1,16(R3)      ;Можно читать скрытые данные?
        BNE  14$            ;Да - будем читать
        MOV  #13,R0         ;Нельзя - выходим с ошибкой 13
        BR   ERROR
14$:    CALL RDSEC          ;Прочитаем сектор
        CALL TSTCRC         ;Проверим CRC
        BNE  NEXT           ;Совпала - посмотрим следующий сектор
        MOVB #1,57(R3)      ;Не совпала - ошибка 1
        DECB 54(R3)         ;Попытки кончились?
        BNE  2$             ;Нет - попробуем еще раз
        MOVB 57(R3),R0      ;Кончились - выходим с ошибкой
        BR   ERROR

           *** 161362 Мультисекторные операции ***

NEXT:   SUB  64(R3),30(R3)  ;Сектор обработали, вычтем из длины
        BLE  EXIT           ;Массив кончился - выход
        INCB 35(R3)         ;Нет - следующий сектор
        CMPB 35(R3),60(R3)  ;Конец дорожки?
        BLOS 3$             ;Нет - продолжим обработку
        BIT  #40,(R3)       ;Верхняя сторона?
        BNE  1$             ;Да - следующая дорожка
        BITB #2,@20(R3)     ;Диск односторонний?
        BNE  1$             ;Да - продолжим обработку
        MOVB #1,32(R3)      ;Нет - установим верхнюю сторону
        BIS  #40,(R3)
        BR   2$             ;Перейдем к обработке
1$:     INCB 33(R3)         ;Следующая дорожка
        CALL GOTRK          ;Позиционируем на нее
```

```
        CLRB 32(R3)         ;Установим нижнюю сторону
        BIC  #40,(R3)
2$:     MOVB #1,35(R3)      ;Теперь сектор будет первым
        CALL CORR           ;Переключить прекоррекцию
3$:     ADD  64(R3),26(R3)  ;Получим новый начальный адрес
        ADD  64(R3),26(R3)
        JMP  WORK           ;Идем на чтение очередного сектора

      *** 161512 Обработка прерывания по вектору 4 ***

        TST  (R5)           ;Сбросить данные
        MOV  #7,R0          ;Заносим код ошибки 7

              *** Выход с ошибкой ***

ERROR:  MOVB R0,@#52        ;Код ошибки - на место!
        MOVB #1,56(R3)      ;Установим бит C в буфере
        BR   EXIT1          ;Идем на выход

              *** Выход без ошибки ***

EXIT:   CLRB 56(R3)         ;Сбросим бит C в буфере
EXIT1:  MOV  46(R3),@#4     ;Восстановить вектор 4
        MTPS 52(R3)         ;Восстановить PSW
        CCC                 ;Сбросить все признаки
        RORB 56(R3)         ;Бит C - из буфера в PSW
        MOV  50(R3),SP      ;Восстановить SP
        RET                 ;Возврат

            *** 161566 П/п проверки CRC ***

TSTCRC: TSTB (R4)           ;Ждем готовности данных
        BPL  TSTCRC
        MOV  #17,R2         ;CRC появится не позже 15. циклов
1$:     BIT  #40000,(R4)    ;CRC сошлась?
        BNE  2$             ;Да - выход
        SOB  R2,1$          ;Нет - очередной цикл
2$:     RET                 ;Если бит Z=1, то произошла ошибка

          *** 161610 П/п чтения сектора в буфер ***

   Входные параметры: R1 - число слов для пересылки

RDSEC:  MTPS #340           ;Маскируем прерывания
        MOV  26(R3),R2      ;Получим адрес буфера
1$:     TSTB (R4)           ;Ожидаем данные
        BPL  1$
        MOV  (R5),R0        ;Читаем два байта,
        SWAB R0             ;   делаем из них нормальное слово,
        MOV  R0,(R2)+       ;   которое помещаем в буфер
        SOB  R1,1$          ;Читаем заказанное число слов
        MOV  42(R3),R1      ;Получаем длину остатка
        BEQ  3$             ;Остатка нет - сразу выходим
2$:     TSTB (R4)           ;Ждем данные
        BPL  2$
        TST  (R5)           ;Вхолостую читаем остаток
        SOB  R1,2$          ;   требуемой длины
        MTPS 52(R3)         ;Восстанавливаем PSW
3$:     RET

      *** 161660 П/п поиска синхропоследовательности ***
```

```
FINDS:  TST  (R5)           ;Сбрасываем прочитанное значение
        MOVB #144,56(R3)    ;Будем искать 100. циклов
1$:     TSTB (R4)           ;Ждем готовности
        BPL  1$
        MOV  (R5),R0        ;Получим прочитанные данные
        BEQ  2$             ;Это 0 - на выход
        INC  R0             ;Это 177777?
        BEQ  2$             ;Да - на выход
        DECB 56(R3)         ;Циклы кончились?
        BNE  1$             ;Нет - повторяем поиск
2$:     RET

     *** 161714 П/п чтения двух слов (не используется) ***

01$:    TSTB (R4)           ;Ждем готовности
        BPL  01$
        MOV  (R5),R0        ;Получаем первое слово
02$:    TSTB (R4)           ;Ждем готовности
        BPL  02$
        MOV  (R5),R1        ;Получаем второе слово
        RET

             *** 161732 П/п запуска чтения ***

STREAD: BIS  #400,(R3)      ;Установим признак "начало чтения"
        MOV  (R3),(R4)      ;    и запишем в регистр
        BIC  #400,(R3)      ;Снимем признак
        MOV  #10,R0
1$:     SOB  R0,1$          ;Небольшая задержка...
        MOV  (R3),(R4)      ;    и снова запишем в регистр
        RET

   *** 161756 П/п поиска начала индекса (не используется) ***

01$:    TST  (R4)           ;Индекс есть?
        BMI  01$            ;Да - ждем исчезновения
        BIS  #400,(R3)      ;Установим признак "начало чтения"
        MOV  (R3),(R4)      ;    и запишем в регистр
02$:    TST  (R4)           ;Индекса еще нет?
        BPL  02$            ;Да - ждать появления
        MOV  62(R3),R0
03$:    SOB  R0,03$         ;Задержимся...
        BIC  #400,(R3)      ;Сбросим признак "начало чтения"
MOV  (R3),(R4)      ;    и запишем в регистр
        RET

          *** 162012 П/п поиска адресного маркера ***

FINDH:  MOV  #17,R0
1$:     TST  (R5)           ;Вхолостую прочитаем
        SOB  R0,1$          ;    15. слов
        CLR  44(R3)         ;Начинаем ожидание индекса
        BICB #40,16(R3)     ;Пока индекса нет
2$:     BITB #10,@20(R3)    ;Сектор задан задержкой?
        BEQ  7$             ;Нет - обрабатывать
        CMPB 35(R3),#1      ;Да - нужен первый сектор?
        BNE  7$             ;Второй и далее задаются заголовками
        BIS  #400,(R3)      ;Установим признак  начало чтения
        MOV  (R3),(R4)      ;    и перешлем в регистр
3$:     TST  (R4)           ;Индекс есть?
        BMI  3$             ;Да - ждем исчезновения
4$:     TST  (R4)           ;Индекс есть?
        BPL  4$             ;Нет - ждем появления
        DECB 55(R3)         ;Один оборот окончен
```

```
        BNE   5$ .              ;Не кончились - продолжаем
        SEC                     ;Кончились - выходим с битом С
        BR    18$
5$:     MOV   62(R3),R0         ;Получим задержку после индекса
6$:     SOB   R0,6$
        BR    12$               ;Сектор, похоже, найден, идем дальше
7$:     CALL  INDEX             ;Проверим наличие диска и обороты
        BCS   18$               ;Обороты кончились - выходим
        BEQ   7$                ;Какие данные? Нули пропустим
        COM   R0                ;177777?
        BEQ   7$                ;Да - тоже пропустим
8$:     CALL  INDEX             ;Проверим наличие диска и обороты
        BCS   18$               ;Обороты кончились - выходим
        BEQ   9$                ;Какие данные? Нули - синхро
        COM   R0                ;177777?
        BNE   8$                ;Нет - пропустим

              *** Проверка адресного маркера ***

9$:     MOV   #3,R1
10$:    MOV   (R5),R0           ;Читаем данные
        BEQ   11$               ;Нули - продолжаем
        COM   R0                ;177777?
        BNE   7$                ;Нет - повторяем поиск
11$:    SOB   R1,10$            ;Да - повторяем 3 раза
12$:    CALL  STREAD            ;Запускаем на чтение
        MOV   #74,R0            ;ОШИБКА! Здесь должно быть 2000
13$:    TSTB  (R4)              ;Готовность есть?
        BMI   14$               ;Есть - считываем данные
        SOB   R0,13$            ;Нет - ждем
        BR    2$                ;Надоело ждать - повторяем поиск
14$:    TST   (R5)              ;Пропускаем A1A1
15$:    TSTB  (R4)              ;Ждем готовности
        BPL   15$
        MOV   (R5),R0           ;Есть - берем данные
        CMP   #120776,R0        ;Адресный маркер (A1FE)?
        BEQ   17$               ;Да - нормальный выход
        CMP   #120773,R0        ;Нет - маркер данных (A1FB)?
        BEQ   16$               ;Да - пропускаем сектор
        CMP   #120770,R0        ;Нет - маркер скрытых данных (A1F8)?
        BNE   2$                ;Нет - повторяем поиск сначала
16$:    CALL  FRESEC            ;Да - пропускаем сектор
        BR    2$                ;Повторяем поиск сначала
17$:    CLC
18$:    RET

  *** 162246 П/п проверки наличия диска и числа оборотов ***

INDEX:  TST   (R4)              ;Индекс есть?
        BPL   2$                ;Нет - идем проверять
        BITB  #40,16(R3)        ;Индекс есть - он уже был?
        BNE   3$                ;Был - идем увеличивать время
        CLR   44(R3)            ;Только что появился - очистим время
        BISB  #40,16(R3)        ;    и запомним факт появления
        DECB  56(R3)            ;Закончился последний оборот?
        BNE   1$                ;Нет - нормальный выход
        SEC                     ;Да - выход с ошибкой
        RET
1$:     CLC
        MOV   (R5),R0           ;Забираем данные
        RET
2$:     BITB  #40,16(R3)        ;Индекса нет - а он был?
        BEQ   3$                ;Нет и не было - идем считать время
        CLR   44(R3)            ;Только что исчез - очистим время
        BICB  #40,16(R3)        ;    и запомним факт исчезновения
```

```
        BR    1$              ;Нормально выходим
3$:     INC   44(R3)          ;Еще один такт индекс не менялся
        CMP   44(R3),#60000   ;Ну сколько можно! Пора кончать?
        BLOS  1$              ;Не пора - нормальный выход
        MOV   #6,R0           ;Терпение кончилось -
        JMP   ERROR           ;   значит, диска нет - ошибка 6

             *** 162364 П/п выбора устройства ***

ENGINE: MOV   #177130,R4      ;Заполняем рабочие регистры
        MOV   R4,R5           ;   адресами PC и
        TST   (R5)+           ;   РД КНГМД
        BICB  #4,16(R3)       ;Очистим признак работы двигателя
        BIT   #20,(R3)        ;Двигатель уже включен?
        BEQ   1$              ;Нет - включим
        BISB  #4,16(R3)       ;Да - установим признак
1$:     BIS   #20,(R3)        ;Включим двигатель
        MOV   (R3),(R4)       ;   и запишем в регистр
        BIC   #57,(R3)        ;Сбросим все другие режимы
        TSTB  32(R3)          ;Сторона верхняя?
        BEQ   2$              ;Да - идем дальше
        BIS   #40,(R3)        ;Нет - включим нижнюю
2$:     BICB  #177774,34(R3)  ;Сбросим лишние биты
        MOVB  34(R3),R1       ;Получим номер привода
        MOV   PC,R0           ;Получим базовый адрес
        ADD   #56,R0          ;   таблицы масок для приводов
        ADD   R1,R0           ;Получим адрес нужной маски
        BISB  (R0),(R3)       ;Установим бит нужного привода
        ADD   R3,R1           ;Получим базовый адрес
        ADD   #4,R1           ;   таблицы TRKTAB
        MOV   R1,2(R3)        ;Настроим указатель CURTRK
        CALL  SETPTR          ;Настроим указатель FLGPTR
        BITB  #4,16(R3)       ;Мотор включен только что?
        BNE   5$              ;Нет - выходим
        CLR   R1              ;Да - подождем раскручивания
3$:     SOB   R1,3$
4$:     SOB   R1,4$
5$:     MOV   (R3),(R4)       ;Записываем установки в регистр
        MOV   10(R3),R1       ;Подождем опускания головки
6$:     SOB   R1,6$
        RET

         *** Таблица масок для включения приводов ***

        .B:1   ;A:
        .B:2   ;B:
        .B:4   ;C:
        .B:10  ;D:

        *** 162540 П/п настройки указателя FLGPTR ***

SETPTR: CLR   20(R3)          ;Очистим указатель FLGPTR
        BICB  #177774,34(R3)  ;Отбросим лишние биты
        MOVB  34(R3),20(R3)   ;Перешлем номер привода в FLGPTR
        ADD   R3,20(R3)       ;Прибавляя базовый адрес FLGTAB,
        ADD   #22,20(R3)      ;   получаем искомый адрес
        RET

            *** 162574 П/п позиционирования ***

GOTRK:  TSTB  33(R3)          ;Нужна дорожка 00?
        BEQ   GOT000          ;Да - позиционируем прямо на нее
        TSTB  @2(R3)          ;Положение головки известно?
        BPL   1$              ;Да - будем искать требуемую
```

```
        CALL GOTOOO         ;Нет - придется сориентироваться
1$:     CMPB @2(R3),33(R3)  ;Нужная дорожка совпадает с текущей?
        BEQ  3$             ;Да - так что же мы тут делаем?
        BHI  2$             ;Нет - искомая ближе к центру?
        CALL GOUP           ;Да - шагнем к центру
        BR   1$             ;Проверим, дошли ли
2$:     CALL GODOWN         ;Нет - шагнем к периферии
        BR   1$             ;Проверим, дошли ли
3$:     MOV  #23420,R0      ;Подождем успокоения головки
4$:     SOB  R0,4$          ;Теперь можно и выйти
        RET

                *** Шаг к центру ***

GOUP:   INCB @2(R3)         ;Номер дорожки станет на 1 больше
        BMI  1$             ;Таких дорожек нет - выходим с ошибкой
        BIS  #100,(R3)      ;Все в порядке - шагать будем вперед
        BR   STEP           ;Пойдем шагать
1$:     MOV  #4,R0          ;Выход с ошибкой 4
        JMP  ERROR

                *** Шаг к периферии ***

GODOWN: DECB @2(R3)         ;Номер дорожки станет на 1 меньше
        BIT  #1,(R4)        ;Дошли до дорожки 00?
        BEQ  1$             ;Нет - можно шагать
        CLRB @2(R3)         ;Дошли: теперь текущая - 00
        RET                 ;   и можно выйти
1$:     BIC  #100,(R3)      ;Шагать будем назад
STEP:   MOV  #310,R0        ;Подготовим задержку
        MOV  (R3),(R4)      ;Перешлем направление шага в регистр
3$:     SOB  R0,3$          ;Подождем, пока контроллер "переварит"
        BIS  #200,(R3)      ;Установим признак шага
        MOV  (R3),(R4)      ;   и запишем в регистр
        MOV  12(R3),R0      ;Получим задержку перехода дорожки
4$:     SOB  R0,4$          ;   и задержимся
        BITB #1,@20(R3)     ;Двойной шаг?
        BEQ  6$             ;Нет - можно выходить
        BITB #100,16(R3)    ;Мы возвращаемся на 00?
        BNE  6$             ;Да - возвращаться надо одинарным шагом
        MOV  (R3),(R4)      ;Нет - придется шагнуть еще
        MOV  12(R3),R0      ;Получим задержку перехода дорожки
5$:     SOB  R0,5$          ;   и задержимся
6$:     BIC  #200,(R3)      ;Очистим признак шага
        RET

    *** 163004 П/п возврата на дорожку 00 ***

GOTOOO: MOVB #200,@2(R3)    ;Установим счетчик дорожек
        BISB #100,16(R3)    ;Возвращаться надо одинарным шагом
1$:     CALL GOUP           ;Шагнем к периферии
        BIT  #1,(R4)        ;Дошли до 00?
        BNE  2$             ;Да - выходим
        TSTB @2(R3)         ;На какой дорожке должны находиться?
        BNE  1$             ;Не на нулевой - еще шагнем
        MOV  #3,R0          ;На нулевой, а ее нет - ошибка 3
        BICB #100,16(R3)    ;Сбросим режим одинарного шага
        JMP  ERROR          ;Выходим с ошибкой
2$:     CLRB @2(R3)         ;Дошли - текущая теперь нулевая
        BICB #100,16(R3)    ;Сбросим режим одинарного шага
        RET

        *** 163072 П/п пропуска сектора ***
```

```
FRESEC: MOV  64(R3),R0      ;Получим длину сектора
FICT:   TST  (R4)           ;Индекс есть?
        BMI  4$             ;Да - ждем начала оборота
1$:     TST  (R4)           ;Индекс все еще есть?
        BMI  3$             ;Да - оборот завершен
2$:     TSTB (R4)           ;Данные готовы?
        BPL  1$             ;Нет - снова
        TST  (R5)           ;Прочитаем данные вхолостую
        SOB  R0,1$          ;    требуемое число раз
        BR   6$             ;    и выйдем
3$:     DECB 55(R3)         ;Все обороты вышли?
        BNE  5$             ;Нет - попытаемся снова
        MOV  #5,R0          ;Да - выходим с ошибкой 5
        JMP  ERROR
4$:     TST  (R4)           ;Индекс исчез?
        BPL  2$             ;Да - оборот начался
5$:     TSTB (R4)           ;Данные готовы?
        BPL  4$             ;Нет - проверим индекс
        TST  (R5)           ;Прочитаем данные вхолостую
        SOB  R0,4$          ;    требуемое число раз
6$:     RET                 ;    и вернемся к основной работе
```

*** 163154 П/п записи сектора ***

Входные параметры:

R1 - число слов для записи

```
WRSEC:  MOV  26(R3),R2      ;Получим начальный адрес массива
        MOV  #13,R0         ;В маркере будет 11. 4E4E
        MTPS #340           ;При записи нам не мешать!
        CALL WRMAR          ;Запишем маркер данных
1$:     TSTB (R4)           ;Готовность есть?
        BPL  1$             ;Нет - ждем
        MOV  (R2)+,(R5)     ;Запишем первое слово данных
        MOV  (R3),(R4)      ;Снимем признак записи маркера
        BR   3$             ;    и пойдем записывать данные
2$:     TSTB (R4)           ;Готовность есть?
        BPL  2$             ;Нет - дождемся
        MOV  (R2)+,(R5)     ;Запишем слово данных
3$:     SOB  R1,2$          ;    и так до конца
        MOV  42(R3),R1      ;Получим длину пустого остатка сектора
        BEQ  5$             ;Остатка нет - выходим сразу
4$:     TSTB (R4)           ;Готовность есть?
        BPL  4$             ;Дожидаемся
        MOV  #0,(R5)        ;Записываем нули
        SOB  R1,4$          ;    в нужном количестве
5$:     BIT  #40000,(R4)    ;Ждем записи CRC
        BEQ  5$
        MOV  #47116,(R5)    ;Записываем 4E4E
6$:     TSTB (R4)           ;Готовность есть?
        BPL  6$             ;Нет - ждем
        TST  (R5)           ;Прекращаем запись
        MTPS 52(R3)         ;Восстанавливаем PSW
        JMP  NEXT           ;Продолжаем запись массива
```

*** 163266 П/п записи маркера данных ***

Входные данные:

R0 - число записываемых слов 4E4E

```
WRMAR:  MOV  #47116,(R5)    ;Запишем 4E4E
        BR   2$             ;Идем на повтор записи
1$:     TSTB (R4)           ;Готовность есть?
        BPL  1$             ;Нет - дождемся
        MOV  #47116,(R5)    ;Запишем 4E4E
```

```
2$:     SOB  R0,1$          ;Повторим заданное число раз
        MOV  #6,R0          ;Запишем 12. нулевых байтов
3$:     TSTB (R4)           ;Готовность есть?
        BPL  3$             ;Нет - ожидаем
        MOV  #0,(R5)        ;Записываем нули
        SOB  R0,3$          ;   в нужном количестве
        BIS  #1000,(R3)     ;Подготовим признак записи маркера
4$:     TSTB (R4)           ;Контроллер готов?
        BPL  4$             ;Нет - ждем
        MOV  #120641,(R5)   ;Записываем маркер A1A1
        MOV  (R3),(R4)      ;Передаем признак в контроллер
        BIC  #1000,(R3)     ;Заранее снимем признак
5$:     TSTB (R4)           ;Готовность есть?
        BPL  5$             ;Нет - ожидаем
        MOV  40(R3),(R5)    ;Записываем маркер из буфера
        RET

        *** 163360 П/п установки прекоррекции ***
CORR:   BIC  #2000,(R3)     ;Снимем прекоррекцию
        CMPB 14(R3),@2(R3)  ;Надо включать на текущей дорожке?
        BHI  1$             ;Нет - пропустим
        BIS  #2000,(R3)     ;Пора - включим
1$:     MOV  (R3),(R4)      ;Перешлем управляющее слово
        MOV  #310,R0
2$:     SOB  R0,2$          ;Подождем, пока контроллер воспримет
        RET

            *** 163412 П/п форматирования ***
FORMAT: MOV  SP,50(R3)      ;Сохраним SP,
        MFPS 52(R3)         ;   PSW,
        MOV  #4,R0
        MOV  (R0),46(R3)    ;  @#4 в соответствующих буферах
        MOV  PC,(R0)        ;Переназначаем вектор @#4 на себя,
        ADD  #176056,(R0)   ;   теперь он равен 161512
        CALL ENGINE         ;Запустим двигатель
        CALL GOTRK          ;Позиционируем на нужную дорожку
        CALL CORR           ;Установим прекоррекцию
        BIT  #4,(R4)        ;Защита записи включена?
        BEQ  2$             ;Нет - начнем форматировать
        MOV  #1,R0          ;Да - ошибка 1
1$:     JMP  ERROR          ;Выходим с ошибкой
2$:     SWAB 32(R3)         ;Обменяем номера стороны и дорожки
        MOVB 17(R3),36(R3)  ;Делаем слово-заполнитель
        MOVB 17(R3),37(R3)  ;   из байта-заполнителя
        MOVB 60(R3),R1      ;Получим длину сектора
        MOV  #1001,R2       ;Запомним номер сектора и код длины
        MOV  #1750,R0       ;Время ожидания индекса
3$:     TST  (R4)           ;Индекс есть?
        BPL  5$             ;Исчез - начнем записывать
        SOB  R0,3$          ;Есть - ждем исчезновения
4$:     TST  (R5)           ;Так и не исчез - прекратим запись
        MOV  #6,R0          ;Диск не вращается - ошибка 6
        SWAB 32(R3)         ;Вернем все на место
        BR   1$             ;Выходим с ошибкой


            *** Запись на дорожку кода 4E4E ***
5$:     MOV  #6200,R0       ;Установим максимальную длину
        MTPS #340           ;При записи нам не мешать!!!
        MOV  #47116,(R5)    ;Записываем 4E4E
6$:     TST  (R4)           ;Индекс появился?
        BMI  8$             ;Да - начнем запись
7$:     TSTB (R4)           ;Готовность есть?
        BPL  7$             ;Нет - ждем
        MOV  #47116,(R5)    ;Записываем 4E4E
```

```
        SOB   R0,6$       ;И так до конца дорожки
        BR    4$          ;Дорожка длинная, ошибка!
8$:     MOV   #20,R0      ;В маркере будет 16. 4E4E (GAP1)
        MOV   #177241,40(R3) ;Адресный маркер (FEA1) - в буфер
9$:     CALL  WRMAR       ;Записываем адресный маркер
10$:    TSTB  (R4)        ;Готовность есть?
        BPL   10$         ;Нет - ожидаем
        MOV   32(R3),(R5) ;Записываем номер стороны и дорожки
        MOV   (R3),(R4)   ;Перешлем управляющее слово
11$:    TSTB  (R4)        ;Готовность есть?
        BPL   11$         ;Нет - дождемся
        MOV   R2,(R5)     ;Записываем номер сектора и код длины
        MOV   #13,R0      ;В маркере будет 11. 4E4E (GAP2)
        MOV   #175641,40(R3) ;Маркер данных (FBA1) - в буфер
12$:    BIT   #400000,(R4) ;Ждем записи CRC
        BEQ   12$
        CALL  WRMAR       ;Записываем маркер данных
        MOV   64(R3),R0   ;Получаем длину сектора
        DEC   R0          ;Одно слово запишем сейчас
13$:    TSTB  (R4)        ;Готовность есть?
        BPL   13$         ;Нет - ожидаем
        MOV   36(R3),(R5) ;Записываем заполнитель
        MOV   (R3),(R4)   ;Перешлем управляющее слово
14$:    TSTB  (R4)        ;Готовность есть?
        BPL   14$         ;Нет - дождемся
        MOV   36(R3),(R5) ;Записываем заполнитель
        SOB   R0,14$      ;И так до конца сектора
        INC   R2          ;Увеличим номер сектора
        MOV   #22,R0      ;В маркере будет 18. 4E4E (GAP3)
```

Примечание. Если увеличить GAP3 до 24-27 слов, то запись будет производиться так же быстро, как и чтение (обычно заголовок очередного сектора проскакивает, и дорожка записывается не за один, а за 10. оборотов). Однако при превышении этой величиной определенного значения конец последнего сектора дорожки может наложиться на ее же начало.

```
        BIT   #4,@20(R3)  ;Сектора длинные?
        BEQ   15$         ;Нет - все нормально
        MOV   #72,R0      ;Да - будет 116. 4E4E (можно увеличить
                          ;                      до 75)
15$:    MOV   #177241,40(R3) ;Адресный маркер (FEA1) - в буфер
16$:    BIT   #40000,(R4) ;Ждем записи CRC
        BEQ   16$
        SOB   R1,9$       ;И так все сектора до конца дорожки

              *** Окончание записи дорожки ***
17$:    TSTB  (R4)        ;Готовность есть?
        BPL   17$         ;Нет - ждем
        MOV   #47116,(R5) ;Запишем 4E4E
        TST   (R4)        ;Индекс есть?
        BPL   17$         ;Нет - продолжим запись до появления
18$:    TSTB  (R4)        ;Готовность есть?
        BPL   18$         ;Нет - дождемся
        MOV   #47116,(R5) ;Запишем 4E4E
        TST   (R5)        ;Прекратим запись
        MTPS  52(R3)      ;Восстановим PSW
        SWAB  32(R3)      ;Вернем все на место
        JMP   EXIT        ;Выходим
```

*** Конец текста драйвера НГМД ***

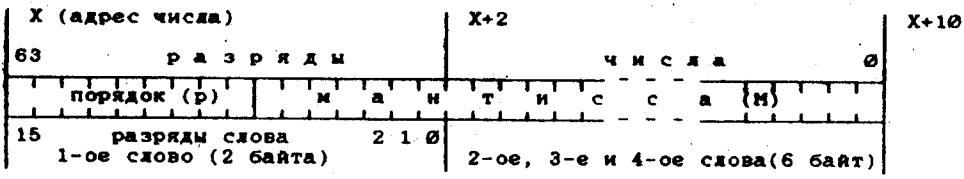
(Далее следует драйвер расширенной арифметики, его расшифровка не производилась)

Использование вещественной арифметики БЕЙСИКа при программировании в кодах*

Прежде всего следует заметить, что применение описываемого здесь метода делает программу в кодах машинно-зависимой: она может работать только на БК-0010.01 без подключения блока МСТД или контроллера дисковода (возможность работы с дисковым БЕЙСИКом не проверялась. — *Прим. ред.*). Кроме того, как и для любых подпрограмм из ПЗУ БЕЙСИКа, нужно учитывать, что область ОЗУ с адресами ниже 4000$_8$ используется БЕЙСИКом для служебных нужд.

Все расчеты в БЕЙСИКе производятся с вещественными числами двойной точности. Если в программе явно указан формат чисел с одинарной точностью, перед выполнением любой операции они все равно преобразуются к двойной.

Числа двойной точности занимают в памяти 4 слова (8 байт). При этом старший разряд первого слова (63 разряд всего числа) кодирует знак: «0»/«1» — «+»/«-», разряды 7—14 (разряды 55—62 числа) — порядок (p), а остальные 6 разрядов первого слова, а также второе, третье и четвертое слова (разряды 0—54 числа) — мантисса (M). Условно это показано на рисунке.



Тогда значение числа N определяется по формуле: $N = (M+1) \times 2^{p-201} = (1+2^{-1} \times a_{54}+2^{-2} \times a_{53}+2^{-3} \times a_{52}+ \ldots +2^{-54} \times a_1+2^{-55} \times a_0) \times 2^{p-201}$, где a_i — значение i-го бита числа (0 или 1). Число 201 восьмеричное, остальные десятичные.

Как же перевести число из привычного для нас десятичного вида в двоичное представление двойной точности? Для этого берется исходное число и делится на 2 до тех пор, пока его целая часть не будет равна 1.

Если исходное число меньше 1, то вместо деления производится умножение на 2, пока целая часть не станет равна 1. Затем дробная часть числа переводится в двоичный код — мантисса готова. К 201 (восьмеричному) прибавляем (или вычитаем, если производилось умножение) количество делений (или умножений) — получим порядок числа.

Осталось «скомпоновать» число — последовательно слева направо записываются: знак («0» — для «+», «1» — для «-»), порядок, мантисса. Остальные биты мантиссы заполняются нулями, либо мантисса укорачивается справа так, чтобы общая длина числа равнялась 64 битам.

Рассмотрим несколько примеров перевода десятичных чисел в двоичные двойной точности.

1. Исходное число: 9.5 . Делим на 2:
9.5
4.75 — результат после первого деления,
2.375 — после второго деления,
1.1875 — после третьего деления.

Дробная часть числа, равная **0.1875**, переводится в двоичный код посредством последовательного умножения на 2. При этом если после умножения число стало больше 1, то целая единица отбрасывается:

---

* Печатается по материалам книги: *Митрюхин В. К., Донской А. Н., Михайлов А. В., Немов А. М.* Программирование на БК-0010.01. Чебоксары, 1993.

0.1875 — умножается на 2,
0.375 — умножается на 2,
0.75 — умножается на 2,
1.5 — отбрасывается 1 и умножается на 2,
1 — все.

Тогда двоичный код мантиссы — **0011** (последовательно сверху вниз переписывается левый столбец цифр (целые части), кроме первого нуля). Было произведено три деления на 2, поэтому порядок числа равен **201+3=204** (числа восьмеричные) или **10000100** (двоичное). Осталось «скомпоновать» число: так как оно положительное, 63 бит равен 0, тогда число имеет следующий вид в двоичном коде: **0 10000100 0011 00000000000000000000000000000000 00000000000000**.

Если X — адрес числа, то код этого числа записывается в словах памяти следующим образом:

Адрес	Содержимое (восьм.)
X	**41030**
X+2	**0**
X+4	**0**
X+6	**0**

2. Исходное число: −13.25. Отмечаем, что оно отрицательное, и все остальные операции проделываем с его абсолютным значением. Делим на 2:

13.25
6.625
3.3125
1.65625

Берем дробную часть числа **0.65625** и переводим в двоичный код:

0.65625
1.3125
0.625
1.25
0.5
1

Получаем **10101**. Порядок равен **201+ 3 = 204 (10000100)**. Тогда число в двоичном коде будет равно: **11000010010101000000000000000000 00000000000000000000000000000000**, или четыре восьмеричных слова: **141124, 0, 0, 0**.

3. Исходное число: 0.0234375. Умножаем на 2:

0.0234375
0.046875
0.09375
0.1875
0.375
0.75
1.5

Переводим дробную часть числа 1.5, равную **0.5**, в двоичный код: это будет **1**. Порядок равен **201 − 6 = 173** (числа восьмеричные). Знак — **0** («+»). Вид числа в двоичном коде: **00111101110000000000000000000000 00000000000000000000000000000000**, или четыре восьмеричных слова: **36700, 0, 0, 0**.

Теперь остается научиться использовать числа с плавающей точкой. Для начала надо поближе познакомиться с тем, как транслируется программа на БЕЙСИКе.

Исходная программа после запуска командой RUN преобразуется в последовательность адресов подпрограмм, реализующих соответствующие операторы БЕЙСИКа, и исходных данных: констант, адресов переменных и т. д. Эти подпрограммы, как и сам транслятор, делающий такое преобразование, зашиты в ПЗУ.

После того как указанная последовательность (ее называют шитым кодом) получена в ОЗУ, в регистр R4 помещается адрес ее начала и командой **JMP @(R4)+** начинается обращение к подпрограмме, адрес входа в которую записан в указанной последовательности первым. Передача управления от одной подпрограммы к другой осуществляется также по команде JMP @(R4)+, содержащейся в конце каждой подпрограммы.

Адреса подпрограмм БЕЙСИКа

Арифметические операции и функции	Адреса подпрограмм	Количество операндов
+ (сложение)	167144	2
− (вычитание)	167124	2
/ (деление)	170776	2
× (умножение)	170210	2
SQR (квадратный корень)	171350	1
SIN (синус)	173614	1
COS (косинус)	173566	1
TAN (тангенс)	174306	1
ATN (арктангенс)	174434	1
EXP (экспонента)	171762	1
LOG (натуральный логарифм)	173052	1
FIX (целая часть)	176212	1
INT (целое значение, меньшее чем аргумент)	176340	1
RND (случайное число)	175176	1
Подпрограмма преобразования результата в целый тип	166760	1

Если в программе на языке ассемблера нам необходимо получить значение какой-либо функции (или арифметической операции), то можно организовать обращение к зашитым в ПЗУ готовым подпрограммам. (Далее под словом «операция» подразумевается арифметическая операция или функция, а под словом «операнд» — операнд арифметической операции или аргумент функции.)

В таблице приводится список адресов подпрограмм, при обращении к которым вычисляется результат соответствующей функции (или арифметической операции). Операнды и результаты операций — числа двойной точности.

Необходимо помнить, что для нормальной работы подпрограмм, указанных в таблице, область памяти с адресами от 2000_8 до 4000_8 должна быть свободной (эта область предназначена для системных переменных БЕЙСИКа).

Следующая небольшая подпрограмма облегчит вызов нужных подпрограмм БЕЙСИКа. Для этого в регистрах процессора должны быть подготовлены следующие данные:

R0 — количество операндов (1 или 2),
R1 — адрес первого операнда,
R2 — адрес второго операнда (для операций с двумя операндами; при операциях с одним операндом значение несущественно),
R3 — адрес подпрограммы операции (из таблицы),
R5 — адрес результата.

Эта вспомогательная подпрограмма работает аналогично транслятору с БЕЙСИКа после команды RUN — формирует требуемый для вызова «шитый код» и передает ему управление. Последовательность адресов подпрограмм и данных («шитый код») начинается с адреса A.

```
FUN:  MOV R0,-(SP)          ; сохранение
      MOV R1,-(SP)          ;  содержимого
      MOV R2,-(SP)          ;  регистров
      MOV R3,-(SP)          ;  в стеке
      MOV R4,-(SP)          ;
      MOV R5,-(SP)          ;
      MOV #A,R4             ; последовательность адресов
                            ;  начинается с адреса A
      MOV #156170,(R4)+     ; адрес подпрограммы пересылки в стек
      MOV R1,(R4)+          ; адрес первого операнда
      CMP R0,#1             ; если операция с одним операндом,
      BEQ C                 ;  то идти на метку C
      MOV #156170,(R4)+     ; засылка в стек второго операнда
      MOV R2,(R4)+          ; адрес второго операнда
C:    MOV R3,(R4)+          ; адрес нужной подпрограммы
                            ;  записывается в последовательность
      CMP R3,#166760        ; если это не подпрограмма перевода
      BNE G                 ;  вещественного в целое, идти на G
      MOV #156350,(R4)+     ; подпрограмма пересылки слова
      BR D                  ;  из стека (результат целого типа)
G:    MOV #156334,(R4)+     ; либо пересылка вещественного числа
D:    MOV R5,(R4)+          ;  из стека по адресу, указанному в R5
      MOV #B,(R4)+          ; организовать выход из шитого кода
      MOV #A,R4             ; запустить шитый код на выполнение
      JMP @(R4)+            ;
B:    MOV (SP)+,R5          ; восстановление
      MOV (SP)+,R4          ;  содержимого
      MOV (SP)+,R3          ;  регистров
      MOV (SP)+,R2          ;  из стека
      MOV (SP)+,R1          ;
      MOV (SP)+,R0          ;
      RTS PC                ; возврат из подпрограммы
A:    .+22
```

По команде **MOV #156170,(R4)+** в эту последовательность записывается адрес подпрограммы, пересылающей значение переменной в стек, а по команде **MOV R1,(R4)+** — адрес самой переменной, указанный в R1. Аналогично в стек засылается второй операнд, если он нужен.

После выполнения любой арифметической подпрограммы результат находится в стеке. Его необходимо извлечь оттуда и переслать по адресу, заданному в R5. Для этого используется подпрограмма @#156334, пересылающая вещественное число (4 слова), или, если результат целый, — @#156350, пересылающая одно слово.

Пример цепочечных вычислений:

```
; вычисляем выражение (4+7)*3.0625-1.25
      MOV #2,R0             ; двухоперандная команда
      MOV #AR1,R1           ; первое слагаемое
      MOV #AR2,R2           ; второе слагаемое
      MOV #167144,R3        ; адрес подпрограммы сложения
      MOV #REZ,R5           ; адрес результата
      JSR PC,FUN            ; складываем
      MOV R5,R1             ; результат используем
```

```
                        ; как первый сомножитель
        MOV #AR3,R2     ; второй сомножитель
        MOV #170210,R3  ; адрес подпрограммы умножения
        JSR PC,FUN      ; умножаем (результат там же   в REZ)
        MOV R5,R1       ; результат используем как уменьшаемое
        MOV #AR4,R2     ; вычитаемое
        MOV #167124,R3  ; адрес подпрограммы вычитания
        JSR PC,FUN      ; вычитаем
; выводим полученный результат на экран
        ADD #10,R5      ; засылаем результат в стек
        MOV -(R5),-(SP) ;
        MOV -(R5),-(SP) ;
        MOV -(R5),-(SP) ;
        MOV -(R5),-(SP) ;
        JSR PC,@#164710 ; вызываем подпрограмму перевода
                        ;  числа в символьную строку
        ADD #4,SP       ; восстанавливаем стек
        MOV #3027,R1    ; адрес символьной строки
        MOV #20000,R2   ; ограничитель строки   пробел
        EMT 20          ; печатаем сформированную строку
; аргументы
AR1:    .#40600.#0.#0.#0    ; 4.0
AR2:    .#40740.#0.#0.#0    ; 7.0
AR3:    .#40504.#0.#0.#0    ; 3.0625
AR4:    .#40240.#0.#0.#0    ; 1.25
REZ:    .#0.#0.#0.#0        ; результат
```

Уважаемые читатели!

Напоминаем основные требования при подготовке материалов, присылаемых в редакцию.

- В письме или в конце статьи четким, разборчивым почерком обязательно укажите свой почтовый адрес (с индексом) и подробные паспортные данные.
- Кроме распечатки статьи и рисунков в двух экземплярах, желательно предоставить в редакцию файлы текста, листингов, иллюстраций и других материалов на дискете (ее можно переслать по почте или доставить в редакцию лично) или на кассете БК. Файлы можно также переслать по электронной почте.
- В письме или на вложенном вместе с дискетой сопроводительном листке необходимо подробно указать тип компьютера, на котором записана дискета (IBM или БК), тип операционной системы, число дорожек и секторов (если дискета записана не на IBM), а также названия текстового и графического редакторов, в которых были подготовлены файлы.

Подробнее о том, как готовить материалы для редакции, можно прочитать в №1 за 1994 г., с.77.
Справки по телефону: (095) 151-19-40
E-Mail: mail@infoobr.msk.su

СПРАВОЧНЫЙ ЛИСТОК

Преобразование двоичных чисел в восьмеричные и обратно

Каждой цифре восьмеричного числа, за исключением крайней слева* соответствует триада двоичных разрядов:

Восьмеричная цифра	0	1	2	3	4	5	6	7
Двоичная триада	000	001	010	011	100	101	110	111

Крайней слева цифре двухбайтного восьмеричного числа соответствует один (15-й, считая с нуля справа налево) двоичный разряд, равный 0 или 1. Аналогично для однобайтных восьмеричных чисел крайней слева цифре соответствуют два (6-й и 7-й) двоичных разряда:

0	1	2	3
000	001	010	011

Примеры:

	Два байта						**Один байт**		
Восьмеричные числа	0	0	1	7	5	3	2	5	0
Двоичные числа	0	000	001	111	101	011	01	101	000

Преобразование шестнадцатеричных чисел в двоичные и обратно

Каждой цифре шестнадцатеричного числа (один байт — одна шестнадцатиричная цифра) соответствует тетрада двоичных разрядов:

Шестнадцатиричная цифра	0	1	2	3	4	5	6	7
Двоичная тетрада	0000	0001	0010	0011	0100	0101	0110	0111

Шестнадцатиричная цифра	8	9	A	B	C	D	E	F
Двоичная тетрада	1000	1001	1010	1011	1100	1101	1110	1111

* С учетом незначащих нулей слева количества разрядов чисел равны: восьмеричное двухбайтное — 6, восьмеричное однобайтное — 3, двоичное двухбайтное — 16, двоичное однобайтное — 8.

Соответствие кодов RADIX50 двухбайтным восьмеричным числам

Кодировка RADIX50 предполагает запись трех символов (допустимы только латинские заглавные буквы, цифры и знаки «ПРОБЕЛ», «$» и « . ») в двухбайтном машинном слове. (В большинстве реализаций кода RADIX50 на БК-0010(.01) и БК-0011(М) знаки «.» и «$» заменены на «%» и «¤», а также добавлен символ «&» вместо кода, неиспользуемого в общепринятом стандарте.) При этом порядок записи символов RADIX в машинном слове аналогичен способу нумерации двоичных разрядов — справа налево, т. е. первому символу из триады RADIX соответствует младшая треть числа, а последнему — старшая.

Таблица

Символ RADIX	Позиция в триаде		
	Первая	Вторая	Третья
Буквы			
пробел	0	0	0
A	1	50	3100
B	2	120	6200
C	3	170	11300
D	4	240	14400
E	5	310	17500
F	6	360	22600
G	7	430	25700
H	10	500	31000
I	11	550	34100
J	12	620	37200
K	13	670	42300
L	14	740	45400
M	15	1010	50500
N	16	1060	53600
O	17	1130	56700
P	20	1200	62000
Q	21	1250	65100
R	22	1320	70200
S	23	1370	73300
T	24	1440	76400
U	25	1510	101500
V	26	1560	104600
W	27	1630	107700
X	30	1700	113000
Y	31	1750	116100
Z	32	2020	121200
Знаки			
¤	33	2070	124300
%	34	2140	127400
&	35	2210	132500
Цифры			
0	36	2260	135600
1	37	2330	140700
2	40	2400	144000
3	41	2450	147100
4	42	2520	152200
5	43	2570	155300
6	44	2640	160400
7	45	2710	163500
8	46	2760	166600
9	47	3030	171700

В приведенной таблице указаны восьмеричные числа, соответствующие символам RADIX, стоящим в первой, второй и третьей позициях, когда два других символа — пробелы (набор символов, принятых для БК; все числа — восьмеричные). Для произвольной трехсимвольной комбинации восьмеричный эквивалент равен сумме указанных в таблице чисел для каждого символа с учетом его позиции. Исходя из этого правила, легко видеть, что RADIX-последовательностям от «пустой» (т.е. трех пробелов) до «максимальной» («999») соответствуют числа от 0 до 174777. Числа, большие 174777, на БК (в частности, в отладчике DEBU10 А. Г. Прудковского) означают повтор RADIX-последовательности с начала, т. е. 175000 — три пробела (эквивалентно 0), 175001 — символ А в первой позиции и два пробела (эквивалентно 1),... 177777 — буква О, цифра 8 и пробел (эквивалентно 2777).

* Незначащим нулям слева в восьмеричном числе соответствуют один или два незначащих пробела в тексте RADIX.

Домашние компьютеры БК-0010 — БК-0011М снискали большую популярность благодаря высоким эксплуатационным качествам, низкой цене, надежности и, главным образом, наличию огромного количества высококачественных программ и дополнительных аппаратных средств.
В этом обзоре пойдет речь о новинках программного и аппаратного обеспечения, появившихся на рынке осенью 1994 г.

А. М. Надежин,
г. Москва

Новинки программного и аппаратного обеспечения БК

Сегодня, спустя десять лет после начала серийного выпуска первой модели БК, программное обеспечение вышло на новый уровень, приближаясь по качеству и оформлению к имеющимся на профессиональных компьютерах, а арсенал аппаратного обеспечения приблизился к набору аналогичных средств для компьютеров IBM.

Программное обеспечение

Дисковая операционная система ANDOS V3.1.

Дисковая операционная система ANDOS V2.30 (а практически V2.50) была описана в первом выпуске журнала «Персональный компьютер БК-0010 — БК-0011М» за 1993 г. Новая версия имеет несколько, можно сказать, революционных нововведений:

- реализована структура подкаталогов неограниченной вложенности, при этом обмен с диском производится даже быстрее, чем в предыдущей версии;
- для удобства работы с прикладными программами имеется встроенный файлер, позволяющий в любой программе (например, в текстовом редакторе), вместо того, чтобы вводить имя файла вручную, выбирать его в появляющемся меню;
- для повышения надежности хранения информации на диске и увеличения скорости работы в новой версии используется принципиально новый алгоритм записи, исключающий фрагментацию файлов, пока диск не заполнится на 80—90%;
- в системе появился настраиваемый драйвер принтера;
- введена работа с датой;
- добавлен встроенный драйвер виртуального диска для БК-0011М, особенностью которого является проверка целостности информации при чтении;
- расширена и улучшена система программных запросов, что значительно облегчает создание утилит для ANDOS.

Кроме того, расширен перечень сообщения об ошибках, добавлен ряд новых команд, улучшена работа многих функций.

Большие изменения произошли и в оболочке *Disk MASTER*:

- появилась возможность работы с подкаталогами на дисках формата как ANDOS, так и MicroDOS;
- функции копирования и перемещения позволяют обрабатывать не только отдельные файлы, но и подкаталоги со всем содержимым;
- скорость копирования увеличена в 4.5 раза;

- при копировании предусмотрено самовосстановление (при зависании компьютера во время выполнения операции достаточно нажать «Reset», и копирование будет продолжено);
- при ошибке записи имеется возможность пометить ошибочный кластер как сбойный и продолжить работу;
- при копировании на одном дисководе реализована автосмена диска (после смены диска в дисководе копирование продолжается автоматически, не требуя дополнительного нажатия на клавиши) с полной защитой от ошибочной установки не того диска;
- значительно улучшена функция просмотра текста (добавлена возможность просмотра строк любой длины, а также режимы постраничного листания, произвольного позиционирования (мгновенного перемещения в любую часть текста), символьного дампа (что облегчает поиск текстов в программах), просмотра и печати на любых принтерах текста в кодировке для любых компьютеров в режимах 64 или 80 символов в строке);
- в оболочке появилось множество новых функций (изменение адреса и даты файла, сортировка каталога по дате, сохранение (с последующим восстановлением) каталога на последних дорожках дискеты, инициалнзация диска, сохранение состояния оболочки и др.);
- значительно ускорен процесс запуска файлов;
- скорость вывода на экран увеличена более чем в два раза.

Комплект поставки ANDOS V3.1 расширен по сравнению с предыдущей версией: теперь в него входят 45 файлов общим объемом 430 кб. Помимо основного набора программ и описаний, в комплект вошли новые версии утилит UNERASE (восстановление удаленных файлов), XEROX (создание резервных копий дисков), текстового редактора EDALT3M, программ DBASIC (дисковый БЕЙСИК для БК-0010.01, обеспечивающий, в отличие от предыдущих версий, полностью корректную работу с файлами данных и с .ASC-файлами), а также ANFOCAL8 (дисковый ФОКАЛ).

Для полноценной работы ANDOS V3.1 с оболочкой Disk MASTER на БК-0010(.01) необходимо дополнительное ОЗУ объемом 16 кб.

Система управления базами данных BASIS V2.10.

Система BASIS позволяет создавать различные справочники, досье, картотеки и т.п. Вызов всех функций производится с помощью интерактивных меню и общепринятых команд текстового редактирования, поэтому для работы с BASIS не требуется какой-либо специальной подготовки.

Новая версия BASIS работает в ANDOS V3.1, причем в отличие от предыдущей версии, базы данных являются обычными файлами на диске ANDOS. Суммарный объем базы ограничен только емкостью диска, размер каждой записи в ней — 512 байт.

Имеется возможность автоматической и ручной сортировки данных. Новые записи можно вставлять не только в конец базы, но и на произвольное место.

Дополнительно производится математическая обработка числовых данных, т. е. BASIS может выполнять функции электронной таблицы.

Имеется двухуровневая система защиты информации от несанкционированного доступа (с помощью пароля и полной шифрации данных).

Возможности BASIS могут быть расширены за счет подгружаемых модулей.

Система качественной печати COMFORT.

Система COMFORT предназначена для печати текста с использованием как встроенных шрифтов принтера, так и специальных графических. Прежде всего она ориентирована на использование струйного принтера МС6312, но может работать и с любыми другими принтерами, имеющими аналогичные команды (МС6313, Robotron CM6329, CM6337, D100M, EPSON), правда, при этом изображения символов для поставляемых дополнительных шрифтов на бумаге будут увеличенными. В комплект поставки входят: сама программа печати COMFORT, редактор шрифта COMFONT и 62 шрифтовых файла (в том числе национальных алфавитов).

COMFORT позволяет создавать и использовать новые шрифты (при этом высота и ширина символов шрифта может меняться). Предварительно текст можно разбить на страницы (с возможностью их нумерации). Перед печатью результат можно просмотреть на экране и при необходимости скорректировать разбивку. Параметры печати (интервал между строками, отступ от левого края листа, масштаб шрифта) также задаются пользователем.

Программа COMFORT работает в среде ANDOS V3.1, используя системный драйвер принтера.

Аппаратное обеспечение

Мультирежимный контроллер дисковода для БК-0010.

Летом 1994 г. начато серийное производство нового контроллера дисковода, разработанного Вадимом Новаком («Альтек»). В контроллер встроено дополнительное ОЗУ объемом 16 кб (в ближайшем будущем планируется начать производство контроллеров с ДОЗУ на 64 и 128 кб). По сравнению с традиционными моделями новый контроллер имеет следующие преимущества:

- автоматическая загрузка при включении питания;
- возможность работы встроенного БЕЙСИКа БК-0010.01 с диском;
- встроенный тест ДОЗУ и системного таймера;
- подключение ОЗУ вместо ПЗУ монитора.

Аппаратно-программный комплекс «МОДЕМ».

Данный аппаратно-программный комплекс предназначен для передачи данных по обычным телефонным линиям (как внутригородским, так и междугородным). В комплект поставки входят: Hayes-совместимый модем американской фирмы XTRON со скоростью обмена 1200 бод, контроллер модема, разветвитель для подключения двух контроллеров к разъему МПИ, дискета с программным обеспечением и описаниями, соединительные кабели.

Комплекс «МОДЕМ» позволяет:

- вести текстовый диалог и обмен файлами между двумя оснащенными модемами компьютерами серии БК, а также между БК и другой ПЭВМ со стандартным Hayes-совместимым модемом;
- работать с системами типа «электронная доска объявлений» (BBS);
- работать с любыми, в том числе и международными, компьютерными сетями (FIDO, Relcom, InterNET и др.)

Для поддержки модема на БК Сергеем Камневым создана программа TRANSIT, функционирующая в среде ANDOS V3.1 и обеспечивающая:

- работу с модемами, имеющими скорость 1200 или 2400 бод;

- работу в режиме терминала (стандартный или дуплексный 80-символьный терминал с альтернативной кодировкой и полным набором псевдографики IBM);
- прием и передачу файлов по стандартному протоколу XMODEM;
- прием и передачу файлов по специализированному протоколу SMODEM (между двумя БК);
- работу с «записной книжкой», позволяющей хранить в памяти имена, телефоны и другую информацию об абонентах (50—200 записей);
- автоматическое дозванивание до одного или нескольких абонентов из занесенных в «записную книжку»;
- возможность подключения дополнительных программных модулей.

Контроллер винчестера.

В настоящее время завершается разработка контроллера винчестера для БК, серийное производство которого предположительно начнется уже в декабре 1994 г. Контроллер позволяет подключать к БК-0011М (а впоследствии — и к БК-0010) любой IDE-винчестер емкостью 20—120 Мб. При использовании однодюймовых винчестеров контроллер и сам винчестер могут быть размещены в одном корпусе с контроллером дисковода (стандартный корпус блока МСТД).

* * *

Итак, вопреки многим предсказаниям, компьютеры серии БК не умирают, а напротив, продолжают развиваться, подтверждением чему, я надеюсь, и служит этот обзор.

По вопросам приобретения различных программных и аппаратных средств обращайтесь по адресу: 129329, Москва, а/я 103, Надежину А.М. В заявку вложите конверт с обратным адресом. Более подробную информацию вы можете получить у автора этого обзора по телефону: (095) 180-30-02.

Приобрести некоторые из описанных выше разработок можно и в редакции журнала. Наш адрес: 125315, Москва, а/я17. Телефон: (095) 151-19-40.

Возможно, не все БКманы знают, что среди журналов, публикующих (или, по крайней мере, делавших это раньше) материалы о компьютерах семейства БК, числится и издаваемый в Минске «Радиолюбитель». В нем опубликовано немало интересных статей, в том числе касающихся схемотехники БК, принципов ее функционирования, недостатков и их устранения. Однако, к большому сожалению, найти эти журналы можно лишь с огромным трудом — даже в центральных столичных библиотеках нет полных подшивок, не говоря уже о возможности их приобретения.

Публикуемая ниже статья (перепечатка из журнала «Радиолюбитель» №9 за 1991 г.) содержит, на наш взгляд, весьма полезные сведения о реализации на БК-0010 обмена информацией с магнитофоном, о причинах сбоев при чтении файлов и о возможностях их устранения.

А. Гречихин, М. Сандлер,
г. Нижний Новгород

О надежности обмена информацией БК-0010 с магнитофоном

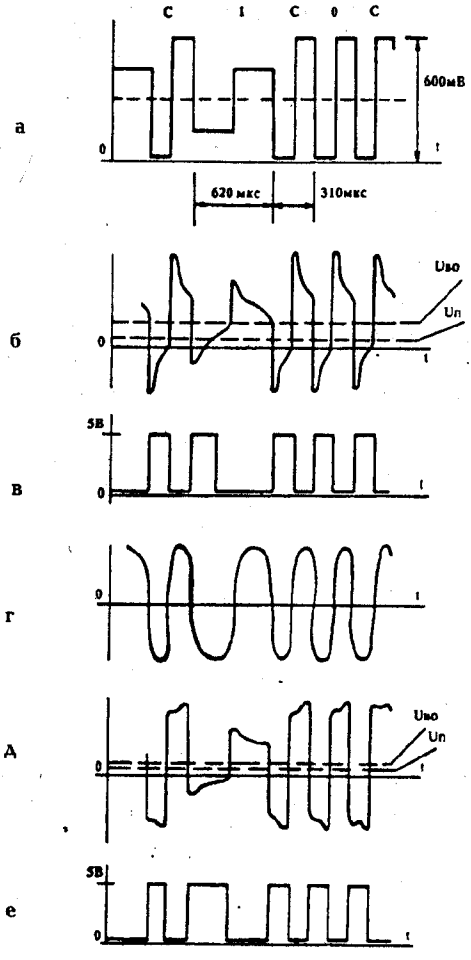


Рис. 1

Опыт показывает, что обмен информацией между БК и магнитофоном не всегда надежен. Иногда не помогают ни юстировка головки, ни чистка ее рабочей поверхности, ни усиление прижима ленты. Не читаются даже записи, сделанные на том же самом, вполне исправном магнитофоне.

Чтобы понять причины этих сбоев и найти способы их устранения, вначале нужно изучить сам метод обмена информации с магнитофоном, реализованный на БК-0010.

Сигнал на выходе БК (рис. 1,а) перед каждым импульсом-битом информации («1» — длинный период, «0» — короткий) содержит импульс синхронизации («С»). Если вслед за синхроимпульсом следует импульс примерно такой же длительности, то при чтении файла он распознается как «0», а если вдвое большей, то как «1». Постоянная составляющая при приеме такого сигнала должна быть исключена, что и делается в тракте записи-воспроизведения.

При воспроизведении сигнал с линейного выхода магнитофона поступает в БК на узел формирования — компаратор (рис. 2, слева показана типовая схема линейного выхода магнитофона). Характеристика компаратора приведена на рис. 3,а. При отсутствии сигнала на его входе имеется постоянное

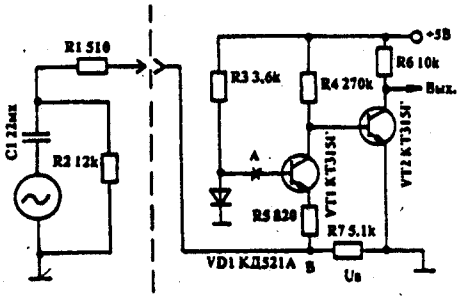


Рис. 2

смещение порядка 20—30 мВ. Из-за существенной несимметричности входа компаратора (при положительной полярности напряжения на входе ток идет только через резистор с сопротивлением 5.1 кОм, а при отрицательной — дополнительно через цепь эмиттер—база VT1) смещение зависит от уровня подаваемого сигнала (кривая 1 на рис. 3, б).

Для передачи без искажений линейный тракт, как известно, должен иметь равномерную в пределах передаваемой полосы частот амплитудно-частотную характеристику (АЧХ) и линейную фазочастотную характеристику (ФЧХ). Исследования сквозных частотных характеристик ряда кассетных магнитофонов («Легенда М-404», «Романтик-306», «Спутник 403», магнитол «Вега 331», «Эврика 302») показали, что главная причина искажений — нелинейность ФЧХ. (Типичный вид кривой ФЧХ приведен на рис. 4.) На рис. 1,б и 1,в показано совместное влияние искажений и смещения на входе компаратора $U_{во}$ относительно порога $U_п$, по которому производится оценка длительностей.

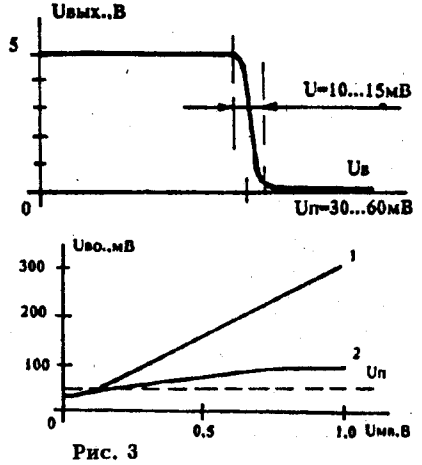


Рис. 3

Таким образом, причинами сбоев при обмене информацией могут быть как непригодность бытовых магнитофонов для записи-воспроизведения широкополосных импульсных сигналов (фазовые искажения, как известно, для правильной передачи речи и музыки несущественны), так и несовершенство входных цепей БК.

Из рис. 1,б видно, что сбои наиболее вероятны при приеме импульса низкого уровня. В то же время АРУ записи (автоматическая регулировка усиления) работает по пиковым уровням коротких импульсов. Отключение АРУ, рекомендованное некоторыми авторами (например, в [1]), иногда помогает, но может привести к несимметричным нелинейным искажениям, которые делают чтение невозможным и в ряде случаев не поддаются коррекции.

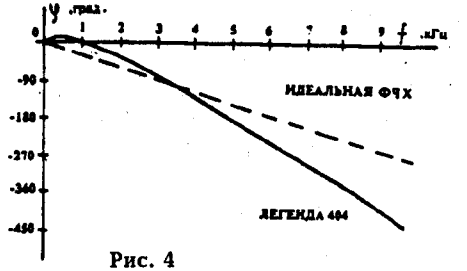


Рис. 4

Зачем же тогда в сигнале БК импульсы с коротким периодом имеют двойной уровень? Известно, что подобные предыскажения во время передачи при соответствующей коррекции при приеме позволяют улучшить отношение сигнал/шум. Однако ни в одном из испытанных нами магнитофонов такой коррекции до линейного выхода не предусмотрено.

Для успешного обнаружения и устранения причин сбоев нужны осциллограф (лучше с возможностью однократного запуска и с памятью) и милливольтметр. Отличным индикатором читаемости файлов может быть и сам БК, если воспользоваться программой, разработанной С. Гуторенко [2]. Приведем модификацию этой программы, позволяющую использовать ее при повышенных и пониженных скоростях воспроизведения (в два раза по сравнению со скоростью при записи).

```
НАЧАЛЬНЫЙ АДРЕС: 020000          ДЛИНА: 000160

012705 177716 042715 000200 012700 020142 012701 000040
005020 077102 005002 011501 021501 001776 011501 005202
102771 021501 001774 020227 000040 002364 105262 020142
126227 020142 000357 103756 012700 000014 104016 005000
005001 005002 104030 005200 005003 116302 020142 042702
177400 104032 005203 062701 000020 020127 001000 002765
000717 000000 000000 000000 000000 000000 000000 000000

КОНТРОЛЬНАЯ СУММА: 161465
```

Программа загружается любым способом в оперативную память БК с адреса 20000$_8$ и запускается, после чего компьютер становится анализатором временных интервалов. При хорошей читаемости подаваемой на вход БК записи на экране дисплея видны два импульса (второй примерно в три раза меньше первого по уровню), разделенные более или менее широким интервалом на уровне «нуля». При плохой идентификации «нулей» и «единиц» импульсы на всех или нескольких графиках, повторяющихся на экране, частично сливаются. Простейшим и эффективным средством согласования сигнала БК с трактом записи-воспроизведения является частотная коррекция, заключающаяся в обеспечении наклона сквозной АЧХ примерно минус 6 dB на октаву в области частот выше 1.5 кГц. Это можно осуществить с помощью RC-цепи (по схеме на рис. 5) как при записи, так и при воспроизведении. В результате выходной сигнал магнитофона будет иметь сглаженную форму, но примерно одинаковые уровни «нулей» и «единиц» и достаточно крутые переходы через «нуль» (рис. 1, г). Лучше всего такое звено поставить в тракт записи (между выходом БК и входом магнитофона), при этом АРУ записи не будет ослаблять амплитуду длинного импульса «единицы».

Если запись сделана без коррекции, то частотная коррекция в тракте воспроизведения лишь понизит уровень сигнала и поэтому может оказаться малоэффективной (без нелинейной коррекции и подбора начального смещения, о чем будет сказано ниже). При этом возможна фазовая коррекция (например, по схеме на рис. 6), дающая несколько лучшие результаты (рис. 1, д).

Надежную читаемость записей, в частности, с малым уровнем можно обеспечить, если добиться совпадения напряжения смещения с серединой переходной области характеристики компаратора ($U_П$ на рис. 3, а) и независимости этого смещения от уровня сигнала. Нами испытаны два способа, первый из которых состоит в подключении к входу БК (параллельно ему) цепи, показанной на рис. 7 (точка В), где может быть применен любой германиевый диод. Резисторами R3 и R4 (точно и грубо) добиваются того, чтобы добавка постоянной составляющей за счет детектирования входного напряжения была минимальной

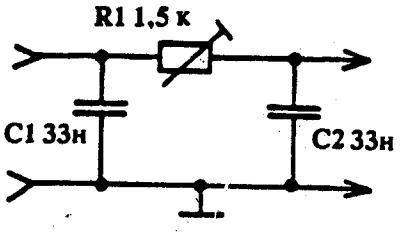


Рис. 5

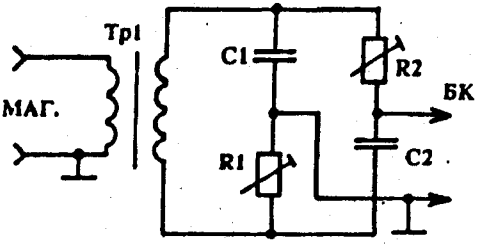


Рис. 6

(кривая 2 на рис. 3, б). Затем при помощи R1 устанавливают начальное смещение так, чтобы надежно читались записи минимально возможного уровня (контролируется милливольтметром, осциллографом и по графикам на экране БК). Невозможность подбора точной симметрии положительной и отрицательной ветвей вольтамперных характеристик входной цепи ограничивает эффективность первого способа.

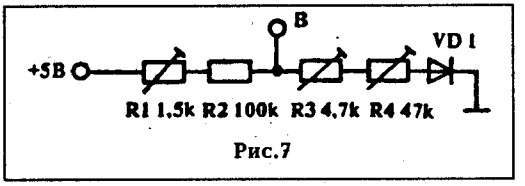


Рис.7

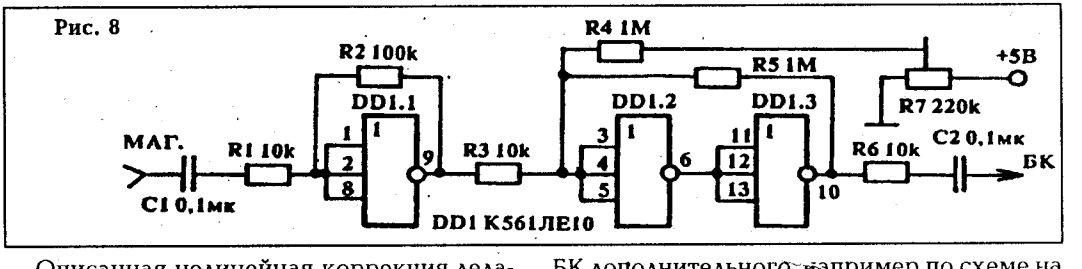
Несколько лучшие результаты получаются при втором способе, но он требует вскрытия БК (внутренняя коррекция). В цепь базы VT1, в разрыв в точке А (рис. 2) включают резистор 33—47 кОм, увеличивая таким образом сопротивление этой цепи в десятки раз. Для подбора начального смещения к точке В подключают резистор, соединенный с плюсом источника питания аналогично R1 на рис. 7. Целесообразно также в цепь магнитофонного входа БК включить последовательно неполярный конденсатор емкостью 1—2 мкФ.

Описанная нелинейная коррекция делает БК менее чувствительным и к линейным искажениям (частотным и фазовым). Тем не менее, целесообразно, наряду с нелинейной, сделать сразу и простейшую линейную (частотную) коррекцию при записи. Следует только помнить, что если цепь нелинейной коррекции — одна и та же для всех магнитофонов, то параметры цепи линейной коррекции зависят от характеристик магнитофона и должны подбираться в каждом конкретном случае индивидуально. Поэтому встраивать цепи линейной коррекции в корпус БК нецелесообразно. Их можно разместить в универсальной приставке с регуляторами или в корпусе разъема отдельно для каждого используемого магнитофона.

В ряде случаев надежность обмена можно повысить и более простыми средствами: добавлением последовательно в цепь воспроизведения резистора 1—3 кОм или шунтированием входа БК таким же резистором; включением последовательно между магнитофоном и БК трансформатора или конденсатора 1—2 мкФ (при воспроизведении) либо однозвенной интегрирующей RC-цепочки при записи (RC=20—100 мкс). Эффект, эквивалентный частотной коррекции, получается при уменьшении или увеличении скорости ленты при воспроизведении [2], а также при некотором нарушении правильного положения воспроизводящей головки (завал верхних частот). В этом смысле оправдан отказ от использования магнитофонов высокого класса. Лучше, наоборот, пользоваться старыми магнитофонами с изношенными головками.

К более сложным средствам относится полная замена существующего компаратора или установка между магнитофоном и БК дополнительного, например по схеме на рис. 8. Дополнительный компаратор, так же как и нелинейная коррекция, позволяет понизить уровень напряжения сигнала надежно читаемых записей от 100—200 мВ до 15—20 мВ.

Рис. 8



Литература

1. *Осетинский Л. Г. и др.* ФОКАЛ для микро- и мини-компьютеров. Л. : Машиностроение, 1988.

2. *Бойко А.* «Электроника БК-0010» работает с магнитофоном. //Информатика и образование. 1988. №5.

В. Л. Щекунов,
г.Полевской-11

Господа, сверим паши... магнитофоны!

В журнале «Информатика и образование», № 5 за 1988 г., была опубликована программа С. Гуторенко для подстройки магнитофона («БК считает периоды», с. 90). С появлением новых моделей компьютеров эта проблема, видимо, не исчезла.

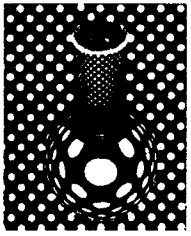
Так как у БК-0011 иначе реализовано обращение к ЕМТ, пришлось адаптировать вышеуказанную программу (конкретно ту ее часть, которая выводит на экран график). Алгоритм же анализа сигнала оставлен без изменений.

После запуска программы с адреса 1000 (здесь и далее все адреса и коды восьмеричные) очищается экран, включается режим «64 символа в строке» и гасится служебная строка. Компьютер ожидает поступления сигнала с магнитофона. Далее программа обрабатывает поступивший сигнал и строит график (подробности о работе с программой см. в статье С. Гуторенко).

Если работа была прервана клавишей «СТОП», повторный запуск необходимо производить с адреса 1006, так как блоки данных для ЕМТ64, ЕМТ30 и ЕМТ32 расположены по одним и тем же адресам.

```
1000)  12700  1216   104064  12700
1010)  14     104016 12705   177716
1020)  42715  200    12700   1232
1030)  12701  400    5020    77102
1040)  11501  5002   5202    102774
1050)  21501  1774   20227   1000
1060)  2367   105262 1232    126227
1070)  1232   357    103761  12700
1100)  14     104016 12700   1216
1110)  12701  5      5020    77102
1120)  12700  1216   104030  12737
1130)  377    1216   5003    116737
1140)  1232   1226   42737   177400
1150)  1226   12700  1216    104032
1160)  13737  1224   1220    13737
1170)  1226   1222   5203    62737
1200)  40     1224   23727   1224
1210)  1000   2751   677     100002
1220)  54202  132754 54202   135614
1230)  377
```

(В редакции работоспособность программы не проверялась.)



Многие пользователи БК-0011М сетуют, что для этого компьютера имеется слишком мало программных средств, в полной мере использующих его богатые ресурсы ОЗУ. Но похоже, ситуация меняется к лучшему: предлагаемый монитор использует резервы ОЗУ для расширения графических возможностей, и в частности, для создания рисунков с четкостью не хуже IBM-овского SVGA.

С. К. Румянцев,
пос. Локня, Псковская обл.

«NEWMON» – улучшенный монитор для БК-0011М

Программа «NEWMON» — это новый улучшенный монитор для БК-0011М. Пользователям этого компьютера, ностальгически сетующим на его несовместимость с БК-0010(.01), эта программа поможет воссоздать на БК-0011М среду работы на старой доброй БК-0010.

«NEWMON» занимает адресное пространство, соответствующее расположению монитор-драйверной системы БК-0010: начинается с адреса 100000_8 и имеет длину 20000_8 байт. Перед загрузкой необходимо подключить первую страницу ОЗУ на адрес 100000_8, выполнив команду **1;1C**. Кроме традиционных драйверов БК-0010, «NEWMON» включает в себя также монитор тестов, столь полюбившийся ветеранам БК (который ранее находился в ПЗУ МСТД), здесь он размещен на месте таблицы изображений символов (графем) знакогенератора БК-0010. (Для вывода символов на экран «NEWMON» использует таблицу графем, содержащуюся в ПЗУ БК-0011М.) Что же касается отладочного монитора, его новая версия обладает рядом преимуществ:

- директивы воспринимаются как в русском, так и в латинском регистре;
- по нажатию клавиши «СБР» происходит не только очистка экрана, но и его нормализация (то, что на БК-0010 приходится делать двукратным нажатием «АР2»+«СБР»);
- возможна пересылка массивов в любую сторону с любыми накладками с точностью до одного байта;
- в операциях обмена данными с магнитофоном реализована удобная система значений по умолчанию. Если в директиве чтения файла на запрос имени нажата клавиша «ВВОД», то читается первый встретившийся на ленте файл; если таким же способом пользователь отвечает на запрос адреса, то файл загружается по адресу, который записан в его заголовке. Аналогично в директиве записи по умолчанию производится запись файла с теми параметрами (адресом и длиной), которые использовались в предыдущей операции чтения или записи. При выводе информации о текущем файле на экране отображается не только имя, но и его адрес и длина;
- при остановке программы пользователя по клавише «СТОП», а также по HALT или по коду несуществующей команды выводится адрес, по которому произошло прерывание, что удобно при отладке.

Кроме того, в отличие от прочих аналогов «NEWMON» предоставляет новые возможности, более полно использующие ресурсы ОЗУ БК-0011М. Применение солидных запасов памяти этого компьютера здесь ориентировано на создание и обработку изображений большого размера, приближающихся по четкости к стандарту SVGA на IBM-совместимых ПЭВМ. Для их хранения в памяти выделено два буфера — рабочий и смежный. С помощью модернизированных драйверов построения точки и прямой (EMT 30 и EMT 32) в этих буферах могут быть сформированы два изображения с разрешением 512×768 точек каждое. При этом аргумент «цвет» в регистре R0

перед выполнением EMT 30 и EMT 32 может принимать значение из расширенного диапазона. Как и раньше, R0=0 кодирует стирание, а R0=1 — запись, к ним еще добавлено значение R0=2 — инверсия. Для функции построения точки EMT 30 допустимы также и большие значения аргумента. Так, R0=3 соответствует закраске контура с началом в данной точке.

При упомянутых аргументах все построения ведутся в рабочем буфере, а при значениях, больших 3, построения касаются обоих буферов изображений одновременно. Например, при R0=4 происходит закраска контура в рабочем буфере с инверсией соответствующих точек в смежном, при R0=5 — стирание смежных точек, а при восьмеричных значениях R0 от 6 до 41 закраска в смежном буфере производится различными псевдоцветами, имитирующими фактуры с разной яркостью и структурой.

С полученными изображениями и их фрагментами можно производить различные манипуляции: стирать, переставлять, инвертировать, накладывать одно на другое, распечатывать на принтере. Эти функции реализованы посредством драйвера вывода символов EMT 16, которому теперь можно передавать на исполнение ранее не использовавшиеся коды 1—6, 11 и 13. Так как аппаратная часть компьютера остается прежней, на экране отображается только фрагмент одного из изображений, режим же «SVGA» реализуется лишь частично. Для того чтобы увидеть картинку целиком, ее нужно распечатать на принтере или подать через EMT 16 специальный код, «сжимающий» суперизображение до размеров экрана БК (естественно, уже с меньшей четкостью). После распечатки на принтере изображение полностью занимает лист формата А4 в горизонтальном («альбомном») расположении. Программа рассчитана на принтер МС-6313М или любой другой, имеющий систему команд EPSON.

Приведем список восьмеричных кодов, выполняющих перечисленные выше преобразования:

1 — обнуление (очистка) рабочего буфера;
2 — инверсия рабочего буфера;
3 — обмен местами рабочего и смежного буферов;
4 — обнуление смежного буфера по маске рабочего (BIC);
5 — установка смежного буфера по маске рабочего (BIS);
6 — инверсия смежного буфера по маске рабочего (XOR);
11 — печать содержимого смежного буфера на принтере;
13 — преобразование смежного буфера в изображение, умещающееся на экране (располагается в средней части рабочего буфера).

Использовать новые графические возможности, предоставляемые «NEWMON», удобно при работе с каким-либо языком высокого уровня, загружаемым в оперативную память. Например, можно загрузить ФОКАЛ БК-0010 по адресу 120000_8, он не требует никаких переделок и безупречно работает с новыми аргументами функций построения точки FT и прямой FV. Коды экранных преобразований в ФОКАЛе можно передавать на исполнение при помощи команды X FCHR(Код,Код,...). Коды в скобках указываются десятичные, выполняются они в порядке записи.

К сожалению, «NEWMON» не может работать совместно с БЕЙСИКом из ПЗУ, так как последний рассчитан на взаимодействие с базовой операционной системой БК-0011М. Но можно использовать загружаемый БЕЙСИК87, работающий совместно с монитором БК-0010, а значит, и с «NEWMON». Этот вариант даже предпочтительнее ФОКАЛа, так как БЕЙСИК87 работает намного быстрее, хотя места он занимает чуть больше, чем ФОКАЛ. Правда, БЕЙСИК87 располагается в ОЗУ пользователя, но зато освобождает память, которую должен был бы занимать ФОКАЛ: с 120000_8 по 137777_8 и которая теперь может быть использована, например, для размещения данных.

Коды управления графической памятью можно передавать БЕЙСИКу при помощи команды PRINT CHR(Код,Код,...). При этом следует иметь в виду, что коды будут выполняться в порядке, обратном указанному в

скобках, т.е. справа налево. Такова особенность транслятора БЕЙСИК87 и ее необходимо учитывать во избежание путаницы. Например, при выполнении команды PRINT CHR(9,3,2) рабочий буфер будет сначала проинвертирован, потом обменян местами со смежным, а затем содержимое смежного буфера будет напечатано на принтере. Аргументы функции CHR, как и в ФОКАЛе, десятичные.

При использовании операторов БЕЙСИК87 «точка» (DOT) и «прямая» (LINE) с расширенным диапазоном аргумента «цвет» может выдаваться сообщение об ошибке, так как для этого аргумента считаются допустимыми лишь значения в диапазоне 0—4. Чтобы этого не происходило, БЕЙСИК87 необходимо настроить на новый диапазон. Сделать это проще всего, выполнив команду POKE 7304,266.

Восприятие построенных при помощи «NEWMON» изображений требует некоторой подготовки. Как уже говорилось, целая картинка существует только в памяти, а на экране отображается лишь ее центральная часть, к тому же повернутая на 90^{o} (так получается из-за особенностей распределения памяти). Вертикальная сторона экрана, которая раньше ассоциировалась с осью Y, теперь соответствует координате X от 256 (вверху экрана) до 511 (внизу). По горизонтали же теперь изменяется координата Y от 0 (справа) до 511 (слева).

Для того чтобы сэкономить память, «NEWMON» формирует изображения печатаемых букв из графем, хранящихся в ПЗУ БК-0011М. К сожалению, извлечь эти данные оказывается не так-то просто: ведь страницы ПЗУ могут быть подключены только к адресу 100000_8, по которому уже находится сам «NEWMON». Для получения графем программе приходится проделывать поистине акробатические пируэты. Вот как это происходит.

Пусть в регистре R1 находится адрес текущей графемы, как если бы на месте «NEWMON» была бы подключена БОС БК-0011М. В регистре R0 в нашем примере содержится маска вывода буквы: прямо или инверсно.

```
103006: MOV  #177716, R5   ; Адрес регистра управления памятью
        MOV  #5, R2        ; Количество слов в графеме символа
        MOV  SP, R4        ; Дублер указателя стека
        MOV  #4000, (R5)   ; Теперь страница памяти, на которой
                           ; находится "NEWMON", подключена и к
                           ; 100000, и к 40000
103024: JMP  43030         ; "NEWMON" перескакивает в свою копию,
                           ; на следующую команду. На самом
                           ; деле ее адрес на 40000 больше
 43030: MOV  #4002, (R5)   ; На то место, где только что был
                           ; "NEWMON", подключается ПЗУ
 43034: MOV  -(R1), -(R4)  ; Извлекаем графему,
        XOR  R0, (R4)      ; обрабатывая ее по маске
        SOB  R2, 43034     ; Адреса указаны "после перескока"!
        MOV  #4000, (R5)   ; ПЗУ отключено, "NEWMON" подключен на
                           ; обе области памяти
 43046: JMP  143052        ; Обратный перескок на свое законное
                           ; место, а фактически на следующую
                           ; команду
143052: MOV  #14000, (R5)  ; Восстановление стандартного
                           ; подключения памяти
```

Далее сохраненная в стеке графема извлекается и используется по назначению — для вывода в соответствующее место экрана, как и в стандартной подпрограмме монитора БК-0010. И так происходит при печати каждого символа! Конечно, вслед-

ствие этих сложных манипуляций с памятью символы выводятся чуть медленнее, чем в БК-0010.

Следует заметить, что такая технология хотя и позволяет экономить память, но довольно опасна в смысле зависаний. В самом деле, если нажать клавишу «СТОП» во время извлечения очередной графемы, достаточно высока вероятность того, что в данный момент на месте «NEWMON» окажется подключенной постоянная память, из которой извлекаются графемы, и обработка прерывания по вектору 4 будет передана на произвольные коды. Компьютер тогда, конечно же, зависнет. Чтобы этого не происходило, «NEWMON» настраивает свой вектор 4 на адрес 2348 в системной области, которая всегда находится «на своем месте». По этому адресу размещена небольшая программа, восстанавливающая стандартное подключение «NEWMON» и только затем передающая управление на обработку прерывания по «СТОП». Таким способом удается существенно повысить надежность защиты программы от зависаний.

При выполнении кодов управления графическими буферами «NEWMON» также приходится часто переключать страницы ОЗУ, но на этот раз в диапазоне экранной памяти, так что опасность зависаний при этом не грозит. Код управления, находящийся в R0, обрабатывается следующим образом:

```
114246: MOV  #110312, R2   ; Адрес таблицы кодов для
                           ; переключения страниц памяти
        ASL  R0            ; Преобразование кода управления
        ADD  PC, R0        ; в адрес в таблице переходов на
114256: ADD  #66, R0       ; соответствующую подпрограмму
114262: MOV  #177716, R4   ; Адрес регистра управления памятью
        TST  (R2)          ; Буфер весь?
        BEQ  114322        ; Да - на выход  (0 - конец таблицы)
        MOV  (R2), (R4)    ; Установить текущую страницу буфера
        MOV  #40000, R3    ; Адрес начала страницы
        MOV  #20000, R5    ; Количество слов в странице
        MOV  R0, -(SP)     ; Сохранить регистры
        MOV  R2, -(SP)     ;
        JSR  PC, (R0)      ; Подпрограмма отработки кода
        MOV  (SP)+, R2     ; Вернуть регистры
        MOV  (SP)+, R0     ;
        CMP  (R2)+, (R2)+  ; Продвинуться в таблице страниц
        BR   114262        ; и продолжать обработку буфера
114322: MOV  #14000, (R4)  ; Вернуть стандартное подключение
                           ; памяти, и на выход
```

Таблица переходов распределяет коды 1—21 на соответствующие подпрограммы, а если коду ничего не соответствует, то указывает на команду RTS PC — возврат из подпрограммы. Для трех кодов — 16, 17 и 21 — оставлен резерв:

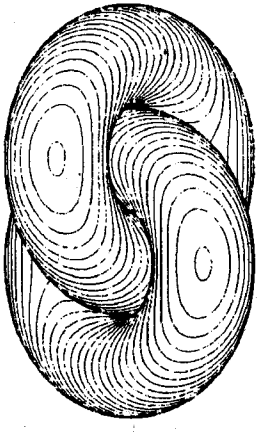
```
114414: JMP  114344        ; Код 16
114420: JMP  114344        ; Код 17
114424: JMP  114344        ; Код 20
```

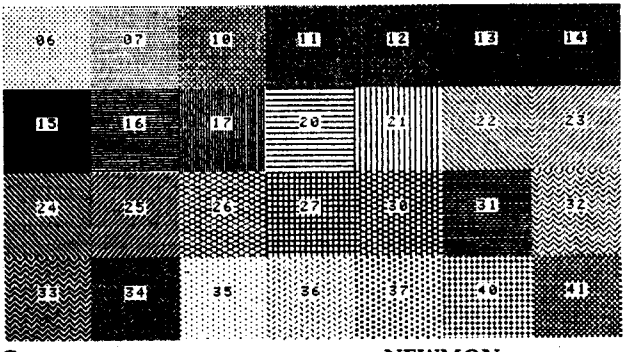
Хотя эти команды переходов тоже указывают на RTS PC по адресу 114344_8, их можно настроить на другие адреса, соответствующие подпрограммам обработки. Это возможно благодаря командам дальнего перехода JMP, в то время как остальные коды распределяются ближними переходами BR.

И еще несколько советов по модификации «NEWMON». По адресу 117620$_8$ расположена таблица структур закрасок, соответствующих значениям 6—41 аргумента «цвет» графической функции «точка». Каждая структура занимает 4 байта. При желании можно заменять эти структуры другими по своему усмотрению.

При необходимости подключения принтера другого типа можно попытаться изменить содержащийся в ней драйвер. Подпрограмма печати одной страницы смежного буфера (в цикле из трех) расположена по адресу 114554$_8$. Собственно драйвер принтера (т. е. подпрограмма передачи кода на принтер) находится по адресу 114736$_8$, код передается в регистре R0. В драйвере оставлен резерв — две пустые команды NOP. В случае инверсного входа принтера на этом месте можно поставить команду инверсии кода COMB R0 (или поместить с помощью отладчика по адресу 114742$_8$ число 105100$_8$).

Автор благодарит всех программистов, чьи творения были использованы при создании «NEWMON»: автора монитора БК-0010 М. И. Дябина; программистов из Вильнюса, написавших БЕЙСИК-ПЗУ, откуда была позаимствована подпрограмма закраски контура; автора программы печати экрана на принтере Ю. А. Зальцмана и надеется, что программа «NEWMON» будет полезна всем, кто занимается черчением и рисованием на компьютере.





Структуры закрасок в программе «NEWMON» и соответствующие им восьмеричные коды.

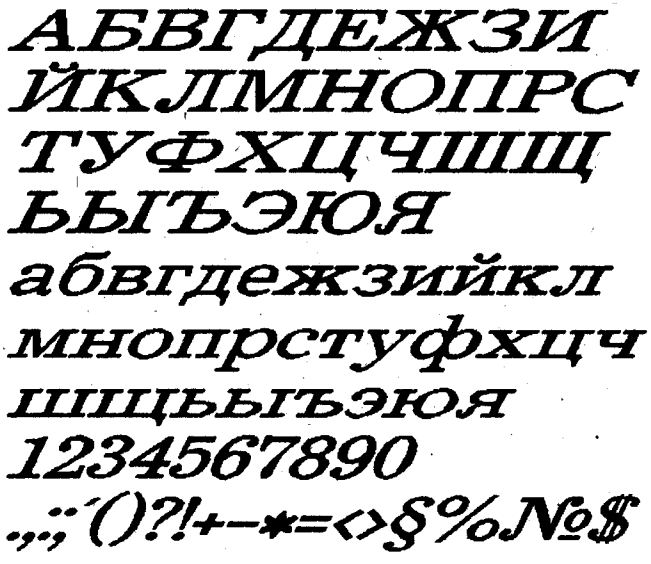
С. К. Румянцев,
пос. Локня, Псковская обл.

«Типографский» шрифт на БК-0010

Программа «АЛФАВИТ» позволяет создавать на БК-0010(.01) заголовки, используя крупный высококачественный «типографский» шрифт. Это может оказаться полезным при съемках титров для любительских кино- или видеофильмов, оформлении разнообразных рекламных листовок и т. д.

После запуска программы на экране появляется рабочее поле, ограниченное рамкой, в центре которого находится графический курсор. (Рамку можно включать и выключать нажатием на клавишу «КТ».) Рабочий режим — графический, при переходе в текстовый символы будут выводиться обычным («мелким») шрифтом. Графический курсор отмечает положение верхнего левого угла рисуемого символа и перемещается с помощью редактирующих клавиш (реализован автоповтор). После нажатия на символьную клавишу на экране появляется соответствующая буква, а курсор смещается вправо в следующую позицию. По нажатию клавиши «ВВОД» курсор перемещается в начало следующей строки (на расстояние, пропорциональное высоте букв). Размер шрифта можно менять, переключая режимы 32—64 символа в строке: в режиме 32 символа буквы в два раза крупнее, чем в режиме 64.

Независимо от установленного до запуска программы режима работы клавиатуры программа печатает только русские буквы (заглавные и строчные). Нажатие на клавишу «РУС» позволяет придать символам наклон (курсивный шрифт), в режиме «ЛАТ» начертание букв прямое. Ошибочно введенные символы можно затереть клавишей «ПРОБЕЛ», а сместить курсор на букву вправо без затирания — клавишей «ТАБ». Символы также могут формироваться в



незакрашенном, контурном виде, что достигается включением режима инверсии («АР2»+«,»).

Полученное изображение можно сохранить, записав экранную область памяти на магнитофон или диск. Для этого нужно нажать одновременно «АР2»+«СБР». Картинка записывается в виде файла, адрес

начала которого равен 42000, а длина — 36000, под именем «PICTURE» (здесь и далее все адреса и числа восьмеричные). Для печати же этого изображения на принтере следует нажать «АР2»+«ВВОД».

Программа рассчитана на работу с принтером МС-6313М или любым другим с системой команд EPSON. При использовании другого типа принтера может возникнуть необходимость изменения драйвера печати, который находится в программе по адресу 1332 (он написан на основе аналогичного драйвера Ю. А. Зальцмана). Подпрограмма передачи кода из регистра R0 на печать расположена по адресу 1534. В случае инверсного входа принтера вместо резервной команды NOP по адресу 1540 следует поставить команду инверсии **COMB R0** (код 105100).

Для интересующихся сообщаю также адреса основных подпрограмм:

1632 — запись экрана на магнитофон,
1720 — формирование инверсной рамки,
1770 — смещение курсора по клавише «ВВОД»,
2076 — отработка нажатия клавиш «ПРОБЕЛ» И «ТАБ»,
2262 — реакция на клавишу «СБР»,
2330 — формирование крупных букв,
2536 — извлечение координат узловых точек и их обработка,
2646 — вход в подпрограмму закраски,
2664 — сама подпрограмма закраски,
3326 — получение цвета пиксела для закраски,
3412, 3602, 3670 — дополнительные проверки контура букв для обнаружения недозакрашенных областей.

По вопросам приобретения программы «NEWMON» и «АЛФАВИТ» можно обращаться в редакцию по телефону: (095) 151-19-40 и E-Mail: mail@infoobr.msk.su

Пример рекламной продукции, созданной в программе «АЛФАВИТ»

А. Ю. Лобанов,
ИнтерСервер

База данных FastCard

Программа FastCard предназначена для ведения баз данных и дает возможность создавать картотеки, вводить, редактировать и удалять карточки, сортировать их, а также формировать табличные отчеты с возможностью вывода на экран, принтер или в файл.

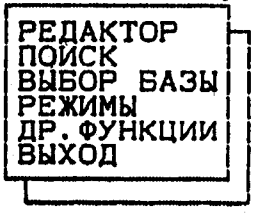
Карточка имеет фиксированное количество полей, которым присвоены заголовки. Размер картотеки также фиксирован и задается при ее создании. Информация в каждом поле представляет собой текстовую строку произвольной длины (не более 255 символов), общая длина всех полей не должна превышать размера карточки. Для каждой карточки запоминается дата и время ее редактирования.

Начало работы

Программа может быть запущена из командной строки командой **RUN FCARD**, если файл FCARD.SAV расположен на DK:, или просто **FCARD**.

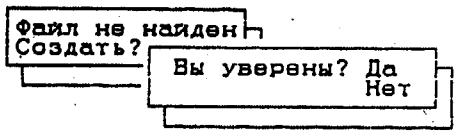
После запуска программы на экране появляется заставка. В правом верхнем углу выводится дата, установленная в операционной системе, и время 0:00. По желанию можно изменить дату и установить время с помощью стрелок управления курсором:
«←», «→» — переход к соседним позициям даты и времени,
«↑», «↓» — изменение численного значения.

При нажатии клавиши «ВК» на экране появляется основное меню программы:



Выбор или создание базы данных

Если база не открыта (ей не присвоено имя), курсор в виде инверсного изображения строки устанавливается в позиции «ВЫБОР БАЗЫ» и при нажатии клавиши «ВК» запрашивает имя файла. Оно состоит из имени устройства (по умолчанию «DK:»), собственно имени файла, состоящего из не более чем шести латинских букв и цифр, и отделенного точкой расширения имени — по умолчанию «.BSE». После нажатия клавиши «ВК» производится поиск файла. При обнаружении он становится текущим, и программа возвращается в основное меню. Если такой файл не был найден (не был создан ранее) — появляется сообщение:



Для создания новой базы с выбранным именем необходимо подтвердить свое желание переводом курсора (инверсной строки) в позицию «Да» и нажатием клавиши «ВК». При этом необходимо задать ее параметры, поэтому на экране появляется запрос количества полей (именованных граф) и максимально возможного количества знакомест в карточке, значения которых выбираются из предложенного ряда клавишами управления курсором «↑» и «↓», а затем подтверждается нажатием клавиши «ВК». После этого необходимо ввести заголовки граф в соответствии с выбранным количеством полей. По окончании ввода заголовка последнего поля запрашивается информационная емкость создаваемой базы для резервирования места под нее на дискете. В меню представлены различные способы задания размера файла,

которые перебираются клавишами « ↓ », « ↑ » и фиксируются клавишей «ВК»:

- по максимальному числу карточек — запрашивается их количество, после чего сообщается, сколько блоков выделено под данный файл;
- 1/2 свободной области — под файл выделяется половина объема наибольшей свободной области (или вторая по величине свободная область целиком) зоны текущего диска;
- вся свободная область — под файл отводится вся максимальная свободная область на диске;
- по числу блоков — запрашивается их количество, после чего сообщается, сколько их действительно выделено.

После определения размера файла программа запрашивает, следует ли проводить проверку диска в выделенной области, а затем возвращается в основное меню.

Редактирование карточек

После возврата в основное меню курсор (инверсная строка) установлен на пункт «РЕДАКТОР». Выбор этого пункта дает возможность создавать новые и редактировать существующие карточки. При этом на экране появляется карточка. В ее заголовке указывается имя базы, номер карточки, который присваивается после окончания ее ввода или очередного редактирования, а также дата и время ввода или последнего редактирования.

Основное поле карточки слева содержит названия выбранных полей (граф), а правая сторона оставлена для текста содержимого полей. Карточка заполняется и редактируется обычным образом, переход из одного поля в соседнее производится клавишами управления курсором « ↑ », « ↓ ». Нажатие «ВК» означает закрытие карточки (окончание ее редактирования). После этого можно воспользоваться подсказкой-меню, расположенной в нижней части экрана. Функции этого меню очевидны из их названий, а обращаться к ним следует, нажимая соответствующую цифру.

Кроме «0», вернуться в основное меню можно также, нажав клавишу «КТ». Вообще, клавиша «КТ» в большинстве случаев вызывает переход «на один уровень вверх» (откуда перед этим мы пришли) — аналогично «ESC» на IBM PC.

Поиск в базе данных

Выбор пункта «ПОИСК» позволяет просматривать базу (и создавать отчеты) не по отдельным карточкам, а в виде групповой таблицы, отобрав их по заданным параметрам:

- вывод результатов поиска: на экран, на печать, в файл с указанной спецификацией (устройство, имя, расширение);
- «маска», имеющая вид карточки, где на месте некоторого количества (в том числе нулевого) произвольных знаков ставится «*», а на место одного произвольного знака — «?»;
- длина каждого поля (графы) таблицы в количестве знакомест. Задание нулевой длины означает запрет вывода данного поля.

После ввода параметров выполняется просмотр базы, программа заносит в таблицу только те карточки, которые удовлетворяют указанной маске.

Настройка режимов

Выбор пункта «РЕЖИМЫ» позволяет:

- установить будильник, подающий сигнал при совпадении установленного в нем времени с текущим;
- установить количество символов в строке (64, 32 или 80);
- изменить установленные при входе в программу время и дату.

Кроме этого, возможна установка следующих режимов клавиатуры:

- звук при нажатии на клавиши. Комбинация СУ/@ циклически изменяет длительность звука от громкого писка до его полного отсутствия;
- преобразование регистра. «Переворачивает» коды строчных и заглавных букв русского регистра, что дает возможность пользоваться клавишей «НР» (слева вверху) для ввода заглавных букв. Нажатие СУ/U задает, а СУ/Q отменяет этот режим;
- расположение клавиш латинского регистра. Кроме традиционного для БК стандарта «ЙЦУКЕН» может быть задан стандарт «QWERTY», принятый на IBM PC.

Нажатие АР2/ЛАТ циклически переключает эти режимы.

Дополнительные функции

Выбор пункта «ДР. ФУНКЦИИ» дает доступ к следующим операциям:

- «ПОМОЩЬ» — вывод на экран справочной информации;
- «СОРТИРОВКА» — изменение порядка следования карточек в базе по заданному правилу.

Остальные пункты этого меню в данной версии программы не реализованы.

Сортировка базы данных

Правила при сортировке следующие. Карточки сравниваются по первой строке и сортируются по правилу, указанному правее нее, затем по второй строке и т. д. Для изменения порядка следования строк необходимо перевести курсор на переставляемую строку и нажать клавишу «ВК». При этом справа от передвигаемой строки появляется стрелка, которую нужно перенести в тот межстрочный промежуток, куда желательно переставить выбранную строку, и нажать клавишу «ВК».

По умолчанию карточки сортируются по возрастанию в алфавитном порядке, вместо этого можно установить сортировку по убыванию и/или по числам (для полей, содержащих числа). Чтобы изменить порядок сортировки, подведите инверсную строку к нужному полю и нажмите «←» для выбора «возрастание-убывание», или «→» — «алфавит-числа».

После задания критериев сортировки нажмите «ВК», когда инверсный курсор находится в строке «НАЧ. СОРТИРОВКИ».

Если вы хотите, чтобы карточки были отсортированы в хронологическом порядке их редактирования, нажмите «ВК» на строке «ДАТА И ВРЕМЯ».

После некоторой паузы, в течение которой программа сортирует базу данных, вновь появляется главное меню.

Д. Ю. Усенков,
г. Москва

ANIRAM-MIRIADA – универсальный резидентный драйвер для ANDOS

Универсальный резидентный драйвер ANIRAM-MIRIADA является представителем нового типа резидентных программных средств для БК-0010. Так называемые «резидентные программы», существовавшие до настоящего времени, были основаны на подмене векторов прерываний БК. Они, как правило, перехватывают EMT 14 и таким способом пытаются защищаться от переписывания векторов стандартными значениями при запуске программы пользователя. Однако если в последней предусмотрена замена содержимого вектора прерывания с помощью оператора MOV, «резидентная программа» общераспространенного типа «теряет управление», и ее нужно устанавливать заново.

В отличие от них, драйвер ANIRAM-MIRIADA встраивается непосредственно в систему ANDOS (имеется три версии драйвера — для ANDOS v2.30 с оболочкой SHELL8+, для ANDOS v2.50 с DiskMaster и для ANDOS v 3.1). Благодаря этому, даже если пользовательская программа переписывает значения векторов прерываний операторами MOV и «отключает» драйвер, после выхода из нее в ANDOS ANIRAM-MIRIADA вновь готова к работе. И так до тех пор, пока не будет произведена деинсталляция или полная перезагрузка ANDOS с системного

диска (порча же самого драйвера, размещающегося в старших адресах расширенного ОЗУ, в большинстве случаев маловероятна).

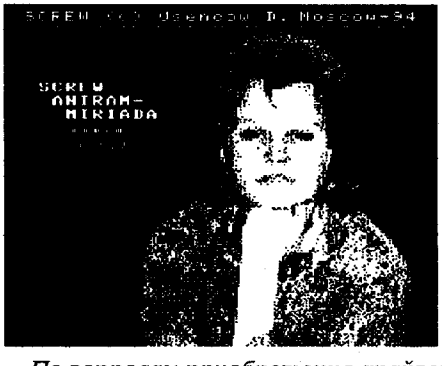
Какие же возможности предоставляет пользователю драйвер ANIRAM-MIRIADA? Он дополняет систему ANDOS следующими весьма полезными функциями:

- нажатие клавиши «АР2»+«ВВОД» вызывает запись копии экрана в файл на диск. (Каждый такой файл имеет типовое имя, заданное при инсталяции драйвера, и уникальный номер в расширении.) В случае же, когда пользовательская программа непосредственно обращается к регистру клавиатуры (вместо использования стандартной функции EMT 6) и «отлов» нажатия клавиш «АР2»+«ВВОД» невозможен, предусмотрен сброс копии экрана в файл по нажатию кнопки PrScn (как это сделано, например, в программе ALTEC Grafix). Кнопка PrScn представляет собой переключатель, замыкающий контакты IRQ2 (вектор прерывания 100_8, вывод B1 порта внешних устройств или A5 системного порта) и «общий»;
- клавиши «АР2»+«РУС» и «АР2»+«ЛАТ» обеспечивают включение и выключение соответственно режима сопровождения вывода текста на экран (только при использовании стандартной функции EMT 16) печатью тех же символов на принтере. Последний может быть любого типа (например, МС6312, Д100 и т. д.), обязательным является только использование в нем стандартных знаковых таблиц КОИ7—КОИ8 (т. е. годится любой принтер, нормально работающий, скажем, с операторами вильнюсского БЕЙСИКа LLIST и LPRINT);
- дополнительно к имеющимся в ANDOS устройствам A:, B:, M: и V:, драйвер ANIRAM-MIRIADA добавляет новое устройство P: — принтер. Теперь любая программа, работающая с текстом или с иными файлами, содержащими текст, но не имеющая собственных средств печати на принтер, может сделать это, если вызвать команду чтения/записи на диск и в качестве имени указать P: (другие символы, идущие после двоеточия, игнорируются). При этом вместо ввода-вывода на диск будет производиться печать на принтере. (Как и в предыдущем случае, подразумевается принтер типа МС6312, Д100 и т. п., для которого не требуется перекодировка.) Перед печатью текст «фильтруется»: символы с кодами, меньшими пробела, а также расположенные в промежутке между латинскими и русскими буквами, заменяются пробелами (в том числе и знаки полуграфики, которых нет в знаковых таблицах многих принтеров). В качестве указателя конца строки в тексте могут быть использованы коды 12_8 и 0.

Работать с драйвером ANIRAM-MIRIADA несложно. После запуска программы на экран выводится запрос: «Инсталляция/Деинсталляция(I/D):». Чтобы ИНСТАЛЛИРОВАТЬ драйвер (подключить его к ANDOS), нажмите «I». Клавиша «D» используется для ДЕИНСТАЛЛЯЦИИ, т. е. для приведения системы ANDOS в исходное состояние.

После ответа «I» программа запрашивает типовое имя длиной до восьми символов для файлов с копиями экрана (расширение в виде порядкового номера им присваивается автоматически).

После этого на экран выдается сообщение «--- Драйвер инсталлирован ---», и после нажатия любой клавиши производится выход в ANDOS, SHELL или DiskMaster (откуда программа была запущена). То же самое, но без запроса имени и с выводом соответствующего сообщения происходит при деинсталляции драйвера.



После инсталляции (драйвер размещается в расширенном ОЗУ по адресам от 155000_8 до 156000_8) можно использовать клавиши «АР2»+«ВВОД», «АР2»+«РУС», «АР2»+«ЛАТ», кнопку PrScn и устройство P: , как это описано выше.

Для перезадания типового имени файлов копий экрана или для установки порядкового номера на 000 нужно повторно инсталлировать драйвер обычным путем.

По вопросам приобретения драйвера ANIRAM-MIRIADA можно обращаться в редакцию по телефону: (095) 151-19-40 или E-MAIL: mail@infoobr.msk.su

ОБМЕН ОПЫТОМ

Д. Е. Лавриков,

г. Жуковский

Полезные советы

Восстановить программу на БЕЙСИКе, если произошел перезапуск процессора, можно одной строкой: **POKE &O2024,PEEK(&O2026)-KS*6**, где **KS** — количество строк в утерянной программе. (Если вы не помните их точное количество, лучше указать меньшее число, чем большее. Иначе возможно появление вместо листинга инверсных символов и прочего «мусора», а программа будет потеряна навсегда.)

И еще один совет. В БЕЙСИКе БК-0010.01 для программного определения режима вывода текста на экран (32/64 символа) используется ячейка системного ОЗУ с адресом 32 (восьм.). Для той же цели владельцы БК-0011М могут использовать вместо нее операторы PSET и POINT.

```
10 PSET (0,0),1
20 IF POINT (0,0)<>3 THEN ?CHR$(155)
```

(Корректное восстановление утерянной БЕЙСИК-программы по предложенному выше методу, как правило, возможно лишь в том случае, когда перезапуск произошел во время ее выполнения или после останова, если не производилось каких-либо изменений в листинге. Эти рекомендации проверены на БК-0010.01. Совет относительно БК-0011М редакцией не проверялся. — *Прим. ред.*)

◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆

Ю. П. Калмыков, А. В. Кузьмич,

г. Москва

Простой способ кодирования лабиринтов

При написании ряда игр на БЕЙСИКе возникает необходимость кодирования лабиринтов. Конечно, самый очевидный способ — использование операторов READ и DATA, но требуемый при этом объем памяти все же слишком велик. Более экономным является хранение закодированных лабиринтов в ОЗУ, выделенном с помощью CLEAR. Предлагаемая ниже простая программная конструкция позволяет декодировать и выводить такой лабиринт на экран.

В строке 10 организован цикл с начального адреса закодированного лабиринта в ОЗУ до конечного. В строке 20 двоичное значение очередной ячейки преобразуется в текстовую запись с дополнением нулями слева до 16 символов. Далее организуется цикл декодирования полученной строки (A$), в данном случае разбиение проводится по два символа. Выделенные пары символов анализируются, и производится вызов соответствующих подпрограмм для вывода элементов лабиринта на экран (строки 50—80).

```
10 FOR A%=<нач.адр> TO <кон.адр> ST 2%
20 A$=STRING$(16%-LEN(BIN$(PEEK(A%))),
   "0")+BIN$(PEEK(A%))
30 FOR B%=1% TO 16% ST 2%
40 B$=MID$(A$,B%,2%)
50 IF B$="11" TH GOS <обработка 1>
60 IF B$="10" TH GOS <обработка 2>
70 IF B$="01" TH GOS <обработка 3>
80 IF B$="00" TH GOS <обработка 4>
90 NEXT B%,A%
```

В данном примере может быть использовано 4 типовых элемента лабиринта (разбивка содержимого каждой ячейки памяти на пары битов). При разбивке по 3 бита допускается уже 8 типовых элементов, для 4 битов — 16, всего же их может быть до 65536.

«Всякое решение плодит новые проблемы»

Закон Мерфи

В. В. Денсель,
г. Каспийск

Игра «REVERSI»

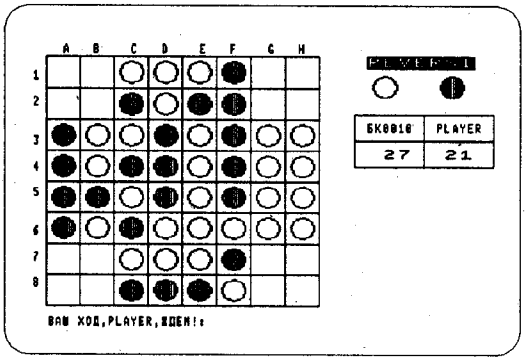
Предлагаю версию игровой программы «Reversi» на языке БЕЙСИК БК-0010.01. От имеющейся в комплекте, прилагаемом к БК при продаже (там эта игра называется почему-то «ШАШКИ» — *Прим. ред.*), она отличается значительно большей быстротой принятия решений (время на обдумывание хода компьютером составляет менее двух секунд по сравнению с прежним — до двух минут).

Для уменьшения объема программы в нее не введена подсказка о правилах игры. Однако отличия последних от правил для упомянутой игры «ШАШКИ» из покупного комплекта невелики. Следует уделить внимание только двум особенностям:

- вводимое в ответ на запрос БК имя играющего должно иметь длину ровно 6 символов (более короткие имена нужно дополнять пробелами). Ввод имени заканчивается автоматически после нажатия последней, шестой клавиши, нажатия на «ВВОД» не требуется;
- ввод хода играющего производится нажатием двух клавиш — вначале буквы (верхний ряд обозначений клеток поля), а затем цифры.

```
10 GOTO 230
20 CIRCLE (X0%,Y0%),8,,,,1.2
30 RETURN
40 PAINT (X0%,Y0%),U%,1
50 RETURN
60 ? AT(23,9)P1%AT(27,9)P2%CHR$(155)
   AT(44,9)CHR$(183) AT(53,9)
   CHR$(183) AT(62,9)CHR$ (183)
   AT (53,7)CHR$(183) AT(62,7)
   CHR$(183)CHR$(155);
70 RETURN
80 X0%=18*K1%+6
90 Y0%=24*K2%+8
100 RETURN
110 ? AT(2,22)CHR$(153)CHR$(156)
   CHR$(157)CHR$(157);
120 IF P1%<P2% THEN ?"ВЫИГРАЛ "A$;
   ELSE IF P1%=P2% THEN ?"НИЧЬЯ";
   ELSE ?"ВЫИГРАЛ БК0010";
130 ? CHR$(156)CHR$(155)
   "Повторить?-(D/N)"CHR$(155)
140 A$=INKEY$
150 IF A$=""THEN 140 ELSE IF A$="D"
   THEN 160 ELSE END
160 FOR I%=1 TO 8
170 FOR J%=1 TO 8
180 M%(I%,J%)=0
190 NEXT J%,I%
200 GOTO 280
210 DATA 8,-8,3,1,1,3,-8,8,-8,-8,-1,-1,-1,
   -1,-8,-8,3,-1,1.5,1.2,1.2,1.5,-1,-3,1,-1,1.2,
   0,0,1.2,-1,1,1,-1,1.2,0,0,1.2,-1,1,3,-1,1.5,
```

```
    1.2,1.2,1.5,-1,3,-8,-8,-1,-1,-1,-1,-8,-8,8,
    -8,3,1,1,3,-8,8
220 DATA 1,2,2,-1,-1,1,1,4,4,193,43,2,""
230 DIM U!(8,8),M%(9,9),K%(2),K$(2),A$(6)
240 FOR I%=1 TO 8
250 FOR J%=1 TO 8
260 READ U!(I%,J%)
270 NEXT J%,I%
280 RESTORE 220
290 READ L%,P1%,P2%,M%(4,4),M%(5,5),
    M%(4,5),M%(5,4),K1%,K2%,X0%,
    Y0%,U%,A$
300 ? CHR$(155)CHR$(12)
310 FOR I%=20 TO 212 STEP 24
320 LINE (30,I%)-(318,I%)
330 NEXT
340 FOR I%=30 TO 318 STEP 36
350 LINE (I%,20)-(I%,212)
360 NEXT
370 ? CHR$(155)AT(23,2)CHR$(156)
    "REVERSI"CHR$(156)
380 GOSUB 20
390 X0%=228
400 GOSUB 20
410 GOSUB 40
420 ? CHR$(155)AT(6,1)"A  B  C  D  E
    F  G  H"
430 ? AT(2,3) "1" AT(2,5) "2" AT(2,8) "3"
    AT(2,10) "4" AT (2,12)"5"AT(2,15)
    "6"AT(2,17) "7"AT(2,19) "8"
    CHR$(7)AT(4,22)"Введите ваше имя:";
440 FOR I%=1 TO 6
450 A$(I%)=INKEY$
460 IF A$(I%)="" THEN 450 ELSE ?A$(I%);
470 A$=A$+A$(I%)
480 NEXT
490 FOR I%=64 TO 104 STEP 20
500 LINE (355,I%)-(499,I%)
510 NEXT
520 FOR I%=355 TO 499 STEP 72
530 LINE (I%,64)-(I%,104)
540 NEXT
550 ? AT(46,7)"БК0010"AT(55,7)
    A$CHR$(155)
560 GOSUB 60
570 GOSUB 80
580 GOSUB 20
590 K2%=5
600 GOSUB 80
610 GOSUB 20
620 GOSUB 40
630 K1%=5
640 GOSUB 80
650 GOSUB 20
660 K2%=4
670 GOSUB 80
680 GOSUB 20
690 GOSUB 40
700 ? CHR$(155)AT(4,22)CHR$(153)"ВАШ
    ХОД,"A$",ЖДЕМ!:"CHR$(155);
710 FOR I%=1 TO 2
720 K$(I%)=INKEY$
730 IF K$(I%)="" THEN 720 ELSE ?K$(I%);
740 K%(I%)=ASC(K$(I%))
750 NEXT
760 K1%=K%(1)-64
770 K2%=K%(2)-48
780 IF K1%=-32 THEN 1210
790 IF K1%<1 OR K2%<1 OR K1%>8 OR
    K2%>8 THEN 810
800 IF M%(K1%,K2%)=0 THEN 850
810 ? AT(2,22)CHR$(153)CHR$(156)
    "ВЫ ОШИБЛИСЬ!ПЕРЕХОДИТЕ!"
    CHR$(156)
820 FOR I=1 TO 600
830 NEXT
```



```
840 GOTO 700
850 FOR P%=K1%-1 TO K1%+1
860 FOR T%=K2%-1 TO K2%+1
870 IF M%(P%,T%)<0 THEN 910
880 NEXT T%,P%
890 ? AT(2,22)CHR$(153)CHR$(156)"ВЫ
    НАРУШИЛИ ПРАВИЛА
    ИГРЫ!"CHR$(156)
900 GOTO 820
910 GOSUB 80
920 GOSUB 20
930 U%=2
940 GOSUB 40
950 M%(K1%,K2%)=1
960 P2%=P2%+1
970 FOR I1%=-1 TO 1
980 FOR J1%=-1 TO 1
990 I%=K1%
1000 J%=K2%
1010 I%=I%+I1%
1020 J%=J%+J1%
1030 IF M%(I%,J%)*L%>=0 THEN 1180
1040 I%=I%+I1%
1050 J%=J%+J1%
1060 IF M%(I%,J%)*L%<0 THEN 1040
    ELSE IF M%(I%,J%)*L%=0 THEN 1180
1070 I%=I%-I1%
1080 J%=J%-J1%
1090 IF M%(I%,J%)*L%=>0 THEN 1180
1100 X0%=18*I%+6
1110 Y0%=24*J%+8
1120 M%(I%,J%)=L%
1130 P1%=P1%-L%
1140 P2%=P2%+L%
1150 IF L%<0 THEN U%=0 ELSE U%=2
1160 GOSUB 40
1170 GOTO 1070
1180 NEXT J1%,I1%
1190 GOSUB 60
1200 IF P1%+P2%=64 OR P1%=0 OR
    P2%=0 THEN 110
1210 L%=-L%
1220 IF L%>0 THEN 820 ELSE 1310
1230 ? AT(2,22)CHR$(155)CHR$(153)
    "ОТВЕЧАЮ ТАК"CHR$(155)">"
    STRING$(6,7)CHR$(K1%+64%)
    CHR$(K2%+48%)"<"
1240 GOSUB 80
1250 GOSUB 20
1260 M%(K1%,K2%)=-1
1270 P1%=P1%+1
1280 FOR I=1 TO 500
1290 NEXT
1300 GOTO 970
1310 V!=-9
1320 FOR I%=1 TO 8
1330 FOR J%=1 TO 8
1340 IF M%(I%,J%)<>0 THEN 1580
1350 FOR P%=I%-1 TO I%+1
1360 FOR T%=J%-1 TO J%+1
1370 IF M%(P%,T%)>0 THEN 1400
1380 NEXT T%,P%
1390 GOTO 1580
1400 V1!=U!(I%,J%)
1410 FOR I1%=-1 TO 1
1420 FOR J1%=-1 TO 1
1430 S!=0
1440 P%=I%
1450 T%=J%
1460 P%=P%+I1%
1470 T%=T%+J1%
1480 IF M%(P%,T%)<=0 THEN 1540
1490 P%=P%+I1%
1500 T%=T%+J1%
1510 S!=S!+1
1520 IF M%(P%,T%)>0 THEN 1490 ELSE
    IF M%(P%,T%)=0 THEN 1540
1530 V1!=V1!+S!
1540 NEXT J1%,I1%
1550 IF V1!<=V! THEN 1580 ELSE V!=V1!
1560 K1%=I%
1570 K2%=J%
1580 NEXT J%,I%
1590 IF V!=-9 THEN ?CHR$(156)AT(2,22)
    CHR$(153)"ПРОПУСКАЮ ХОД!"
    CHR$(156) ELSE 1230
1600 GOTO 1210
```

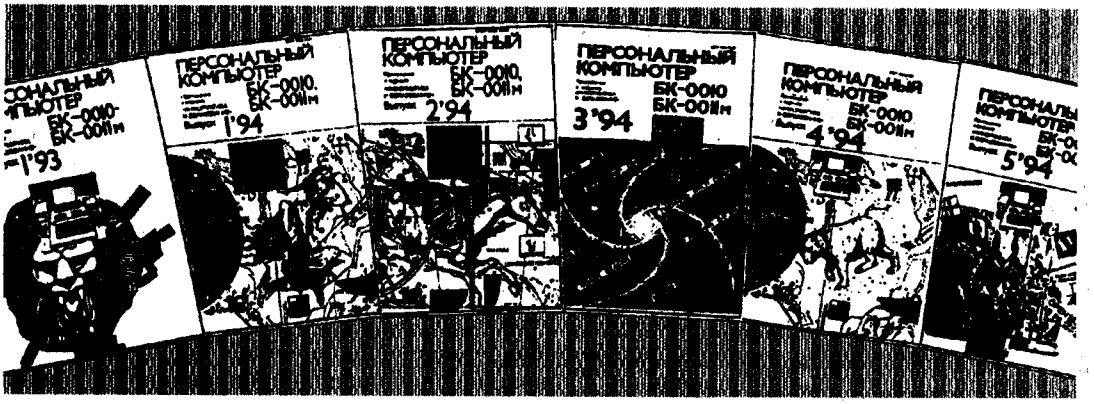
Задание кодов клавиш управления в «ТЕТРИСе»

В стандартном «ТЕТРИСе» из комплекта программ, прилагаемого к БК при продаже, для управления игрой задействованы «диагональные стрелки» (имевшиеся в старой БК-0010 с пленочной клавиатурой, но отсутствующие в БК-0010.01). Использование же дублирующих эти стрелки клавиш с цифрами не очень удобно.

Чтобы перенастроить игру на другие клавиши управления, нужно заменить соответствующие им управляющие коды. Адреса ячеек, содержащих эти коды, приведены в таблице. Каждая из ячеек содержит два байта, являющихся кодами пары дублирующих друг друга клавиш (скажем, для команды «влево» это «стрелка влево вверх» и «цифра 7»), за исключением двух последних ячеек с кодами отдельных команд.

Для перенастройки на другие клавиши достаточно заменить лишь первые байты в каждой дублирующей паре. Например, чтобы использовать для движения фигур стрелки влево и вправо, нужно занести в ячейки 14370 и 14374 коды 40067 и 14471 (все адреса и числа здесь и в таблице — восьмеричные)

Адрес ячейки	Исходное значение	Байты	Клавиши	Команда
14370	16067	34, 67	«↖», «7»	«влево»
14372	15070	32, 70	«↑», «8»	«поворот»
14374	16471	35, 71	«↗», «9»	«вправо»
14376	32440	65, 40	«5», «␣»	«сброс вниз»
14400	31063	62, 63	«2», «3»	«звук», «ускорить»
14402	30061	60, 61	«0», «1»	«пауза»



Содержание выпусков журнала за 1993 и 1994 гг.*

1993 г. (один выпуск)

1994 г. (пять выпусков)

№ 1

* Через тире указаны номера страниц.

№ 2

№ 3

№ 4

УВАЖАЕМЫЕ ЧИТАТЕЛИ!

Обратите внимание на изменение почтового адреса редакции журнала. Наш новый адрес для переписки:

125315, Москва, а/я17.

Отправивших корреспонденцию по прежним адресам (*103051, Москва, ул. Садовая Сухаревская, д. 16, комн. 9* и *125315, Москва, ул. Часовая, д. 21б, комн. 20*) просим не беспокоиться — все ваши письма, бандероли и т. п. дойдут по назначению.

Обращаем внимание читателей на то, что всего в 1993 г. был издан один, а в 1994 г. — пять выпусков нашего журнала (из них два — в первом полугодии 1994 г. и три — во втором). Начиная с 1995 г. планируется издание шести выпусков в год (по три в полугодие). Для удобства наряду с нумерацией в пределах каждого года на титульном листе в скобках указывается «сквозная» нумерация выпусков.

Кроме того, сообщаем об изменениях правил почтовой доставки. С первого полугодия 1995 г. рассылку выпусков подписчикам будет производиться редакцией в виде бандеролей. Не успевшие подписаться на почте могут оформить подписку в помещении редакции (как с почтовой доставкой, так и на льготных условиях по каталожной цене с получением выпусков в редакции лично). Здесь же можно приобрести недостающие выпуски начиная с 1993 г., журналы «Информатика и образование» и «Персональный компьютер УКНЦ», заказать ксерокопии статей, опубликованных в журналах «Информатика и образование» (с 1986 г.) и «Вычислительная техника и ее применение» (с 1988 по 1992 г.), а также приобрести разнообразные программы для БК-0010(.01), БК-0011(М) и IBM-совместимых компьютеров.

Справки по телефону: (095) 151-19-40 и E-Mail в сети RELCOM: mail@infoobr.msk.su

Наша анкета

Уважаемые читатели! Публикуя различные материалы о компьютерах семейства БК, редакции журнала хотелось бы знать ваше мнение о том, насколько полезны те или иные рубрики, статьи, советы, рекомендации и удовлетворяет ли вас подача содержащихся в них сведений.

На обороте приводится список основных рубрик и наиболее значительных статей, опубликованных в 1993 и 1994 г. г. Просьба оценить по десятибалльной системе необходимость данной рубрики (статьи) и качество подачи материала (от «10» — «бесценные, очень необходимые сведения», «высшее качество» до «0» — «бесполезная трата бумаги», «качество низкое»). Проставив в отведенной справа колонке две эти оценки, вырежьте купон анкеты и перешлите в редакцию по адресу: 103051, Москва, ул. Садовая-Сухаревская, д. 16, комн. 9.

В сопроводительном письме вы можете дополнительно сообщить пожелания и критические замечания. Редакция постарается учесть их при подготовке в печать дальнейших выпусков.

Уважаемые читатели!

Подписаться на журнал «Персональный компьютер БК-0010 — БК-0011М» можно в любом отделении связи или непосредственно в редакции. В каталоге ЦРПА «Роспечать» данные о журнале следует искать на букву «Б» — «Библиотечка журнала «Информатика и образование».

Чтобы приобрести отдельные выпуски журнала через редакцию:

- ☐ частным лицам необходимо перечислить за каждый выпуск 5000 руб., почтовые расходы (по пересылке бандероли из Москвы в пункт назначения) и 500 руб. (орграсходы);
- ☐ предприятиям и организациям необходимо перечислить на расчетный счет редакции за каждый выпуск 7300 руб. (включая почтовые расходы по пересылке бандероли из Москвы в пункт назначения) и 500 руб. (орграсходы).

Расчетный счет для Москвы и Московской области: 609602 в ММКБ, филиал «Интеллект», МФО 212199, уч.1Е

Расчетный счет для других городов России и ближнего зарубежья: 609602 в ММКБ, филиал «Интеллект», кор. счет 216161800 в ЦРКЦ ГУ ЦБ РФ, уч.С3, МФО 211004.

Перечисление денег необходимо подтвердить письмом с вложенной в конверт заявкой (см. на обороте) по адресу:
103051, Москва, ул.Садовая Сухаревская, д.16, комн.9.
Редакция журнала «Информатика и образование».
Справки по телефону: (095) 151-19-40
E-MAIL: mail@infoobr.msk.su

Рубрики	Для оценок	
HARD & SOFT		
ОБМЕН ОПЫТОМ		
СПРАВОЧНОЕ БЮРО		
СПРАВОЧНЫЙ ЛИСТОК		
НАЧИНАЮЩИМ ПОЛЬЗОВАТЕЛЯМ		
ПОТЕХЕ ЧАС		
ГЛЮКАДЕМИЯ		
ЕСТЬ ИДЕЯ		
Статьи		
МикроЭВМ БК-0010. Архитектура и программирование на ассемблере		
Обзоры графических средств и дисковых систем для БК		
Библиотеки подпрограмм на ассемблере (графика, окна и меню, подключение мыши		
Поиск и устранение неисправностей БК-0010		
Техническое описание БК-0010		
Материалы о БК, опубликованные в журналах		
Графопостроитель для БК - своими руками		

ЗАЯВКА

на журнал «Персональный компьютер БК-0010 — БК-0011М»

_____

(адрес подписчика с указанием почтового индекса)

_____

(фамилия, имя, отчество полностью)

_____

(номер выпуска и год издания)

_____

(общее количество экземпляров)

Перечислено на расчетный счет_____

_____руб.

(общее количество экземпляров, стоимость одного экземпляра)

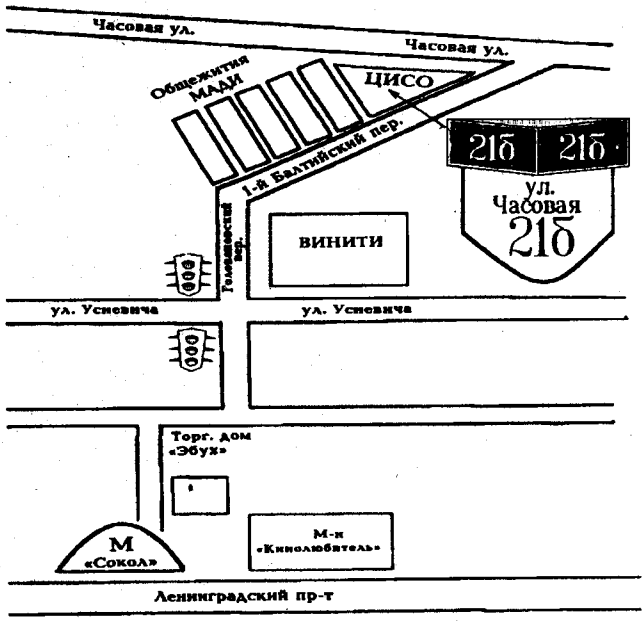
Платежное поручение № _____от _____199 ____г.

СОДЕРЖАНИЕ

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР БК-0010 – БК-0011М

Наш адрес: Москва, ул. Часовая, 21Б, помещение Центра Интерактивных Средств Обучения (ЦИСО), комн.20
Телефон: (095) 151-19-40

Как к нам добраться:



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

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР БК-0010 - БК-0011М

Главный редактор
Васильев Б. М.

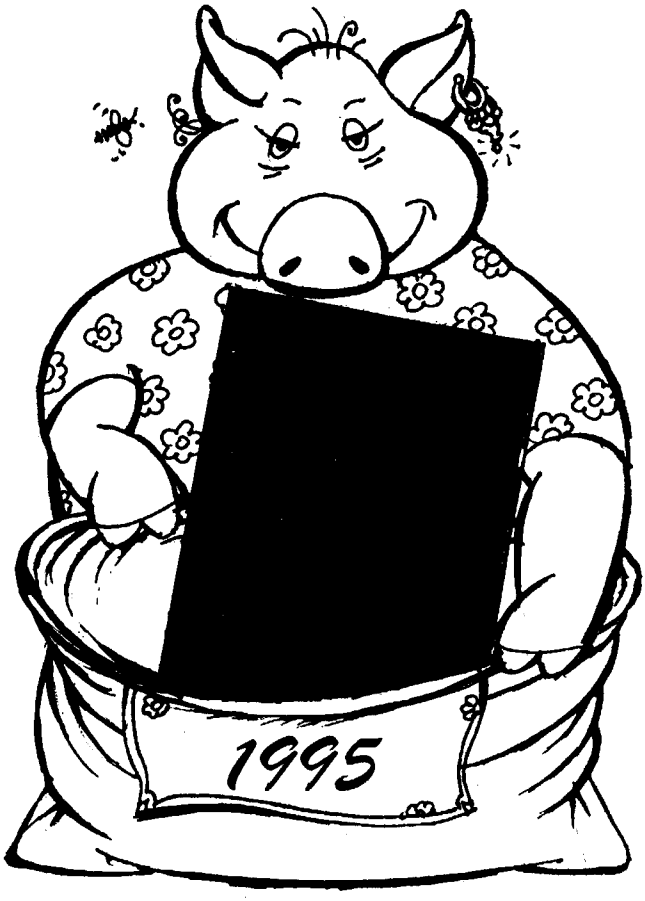
Редактор
Усенков Д. Ю.

Корректор
Антонова В. С.

Компьютерная верстка
Макеев А. Б.

Подписано в печать с оригинал-макета издательства «Информатика и образование» 25.12.94. Формат 70×100 1/16. Бумага офсетная. Усл.печ.л. 6,5. Заказ №3841 . Цена 4000 руб. (по подписке). В розничной продаже цена договорная. Тираж 1500 экз.
Ордена Трудового Красного Знамени Чеховский полиграфический комбинат Комитета Российской Федерации по печати.
142300, Чехов Московской обл.

Содержание выпуска 1, 95:



Редакционный коллектив журнала
«Персональный компьютер БК-0010 – БК-0011М»
поздравляет наших читателей

С НОВЫМ ГОДОМ!

Желаем всем подписчикам безаварийной, безвирусной и максимально совместимой эксплуатации ваших ПЭВМ.

Программирующим – плодотворной работы,
пользователям – удачного ввода и успешного запуска.

Всем нам обильного и дешевого компьютерного рынка.

И в год свиньи – без свинства шоковых терапевтов.