Содержание выпуска 3,95

- **Архитектура и программирование на языке ассемблера** *(окончание)*
- **Машинная графика:**

  «Негативная» графика
  Рисование окружностей и эллипсов средствами ассемблера
  Графика на ФОКАЛе
  Шрифты нестандартного вида
  Работа со спрайтами
- **Обмен опытом**
- **Глюкадемия:**

  Доработка программы БК-PAINT
- **HARD&SOFT:**

  Доработка ОЗУ 16 кб для работы программ в адресах 140000 — 174000 на БК-0011(М)
  Электронный диск для БК-0010.01
- **Справочный листок:**

  Шрифты принтера МС-6313

Подписка 1995 года, второе полугодие

Название журнала	Подписной индекс	Периодичность	Стоимость подписки на полугодие по каталогу «Роспечати»
«Информатика и образование»	70423 для индивидуальных подписчиков	1 раз в 2 мес.	24 000 руб.
	73176 для предприятий и организаций		75 000 руб.
Библиотека журнала «Информатика и образование»			
«Персональный компьютер БК-0010 —БК-0011М»	73177 для индивидуальных подписчиков	1 раз в 2 мес.	15 000 руб.
	73092 для предприятий и организаций		30 000 руб.
«Персональный компьютер УКНЦ»	Рассылка по заявкам, присылаемым в редакцию по адресу: 125315, Москва, а/я 17	1 раз в 3 мес.	Договорная

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР БК-0010 БК-0011м

Приложение
к журналу
«ИНФОРМАТИКА
И ОБРАЗОВАНИЕ»

2'95 (8)

Издается с 1993 г.

В НОМЕРЕ

Программирование на ассемблере

Передача массивов в USR-подпрограммы

Шитые коды БЕЙСИКа БК-0010.01

Увеличение объема БЕЙСИК-программ

RESTORE с переменным номером строки DATA

Винчестер на БК — уже реальность!

Дисковый БЕЙСИК — доработка КНГМД

Москва «Информатика и образование» 1995

Авторы выпуска

Бутырский Д.
Ермаков В. В.
Зальцман Ю. А.
Канивец И. В.
Королев М.
Котов Ю. В.
Милюков А. В.
Неробеев С. М.
Новак В. Е.
Сорокин А. В.
Усенков Д. Ю.

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

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

E-Mail: mail@infoobr.msk.su
Телефон: (095) 151-19-40
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—5 за 1994 г. и №1 за 1995 г.

Ю. А. Зальцман,

г. Алма-Ата

МикроЭВМ БК-0010.

Архитектура и программирование на языке ассемблера

Системная область БК-0010

Мы уже неоднократно упоминали этот термин — системная область. Пора, наконец, рассказать подробно, что это такое. Для БК-0010 системной областью называется область адресов от 0 до 777. В ней помимо уже известных нам векторов прерывания размещаются СИСТЕМНЫЕ ПЕРЕМЕННЫЕ и СТЕК. Стек — это область памяти, обычно адресуемая косвенно через регистр SP, и подробно мы ее рассмотрим позже (в приводимых ранее примерах мы ее уже использовали). А системные переменные — это специально выделенные ячейки памяти, используемые в качестве буфера для работы драйверов монитора БК-0010. В них запоминаются различные промежуточные данные, признаки режимов ЭВМ и т. п. Часть ячеек свободна и может быть задействована программистом для своих нужд. Однако при выборе таких ячеек надо иметь в виду, что некоторые из них могут использоваться в языках высокого уровня или некоторыми распространенными программами, в частности ассемблер-системой.

Приведем сведения о всех ячейках системной области в порядке возрастания их номеров, включая как уже известные нам, так и те, с которыми мы пока не встречались. Для всех ячеек будут даны сведения о хранимой там информации, по возможности с пояснениями, что это такое и как эти ячейки используются системой БК-0010 или ЦП. Если информацию, хранящуюся в ячейках, можно как-либо применять в своих программах, постараемся обратить внимание и на это. Для каждой ячейки будет приводиться ее адрес. Если используется машинное слово или несколько слов, будет приведен только адрес (адреса) ячеек. Если же задействован отдельный байт, это будет оговариваться особо.

У читателя может возникнуть вопрос, откуда автор получил сведения о системных ячейках. Во-первых, был использован авторский текст драйверного модуля БК-0010 (листинг DMBK, автор – М.И. Дябин, Москва, 1983—1985 гг.). Во-вторых, производилось дезассемблирование содержимого ПЗУ БК-0010. И наконец, со многими ячейками автор проводил эксперименты для определения их функционального назначения и возможного использования.

* * *

- 0, 2 — не используется. В эти и некоторые другие неиспользуемые ячейки при включении ЭВМ заносится адрес перезапуска системы — 100000. При случайном косвенном обращении по адресу такой ячейки произойдет инициализация системы.
- 4, 6 — вектор прерывания по клавише «СТОП» и команде останова HALT. Прерывание по клавише «СТОП» внеприоритетное. Значение первого слова вектора в ПМ — 100442, в МСД — 160722, в ФОКАЛе — 121110, в БЕЙСИКе — 120234 (если это значение не изменено программой пользователя).
- 10, 12 — вектор прерывания по резервному коду. ЦП выполняет данное прерывание при получении кода, отсутствующего в его наборе команд.
- 14, 16 — вектор прерывания по Т-разряду (при работе в пошаговом режиме).
- 20, 22 — вектор прерывания по команде IOT.

- 24, 26 — вектор прерывания по аварии сетевого питания ЭВМ (по входу ACLO ЦП). Это прерывание внеприоритетное.
- 30, 32 — вектор прерывания по команде EMT.
- 34, 36 — вектор прерывания по команде TRAP.
- 40—56 — ячейки (байты) слова состояния дисплея. Они уже были описаны в разделе, посвященном EMT 34, там же приведена таблица их значений. Напомним, что исходное содержимое всех этих ячеек (при перезапуске ЭВМ) равно нулю, а при включении соответствующего режима дисплея становится равным 377, за исключением ячейки 43 — индикатора регистра «РУС», куда заносится число 200 (маска регистра «РУС»). Ячейки могут быть использованы прежде всего для определения включенных в данный момент режимов с целью их изменения и для приведения изображения на экране к желаемому виду и формату. Запись числа 377 в некоторые из ячеек (44—47, 54—56) приводит к прямому включению данного режима, иногда — с некоторыми нюансами по сравнению с включением режима подачей кода через EMT 16. Например, при включении таким образом режима «БЛОКРЕД» и «ИНДСУ» отсутствует индикация в служебной строке; при записи числа 377 в ячейку 56 курсор со следующего выводимого символа исчезает, но остается его «призрак» на экране и т. п. Тем не менее эти ячейки могут в некоторых случаях очень успешно использоваться для переключения режимов, особенно когда это переключение должно выполняться с очень высокой скоростью, на несколько порядков быстрее, чем по EMT 16. Записав в такие ячейки число, отличное от 0 и 377, можно сделать режим «неотменяемым» с клавиатуры — никакие нажатия клавиш БК не позволят снять его.

Другие ячейки (40—43, 50—53) при записи в них какого-либо числа включают режим частично и «ненормально» — дальнейшая работа с дисплеем становится невозможной или затруднительной. Но и эти ячейки могут быть использованы для каких-либо целей. Например, регистр «РУС—ЛАТ» можно переключать через ячейку 43, но при этом поменяется на обратную и установка регистра «СТР»—«ЗАГЛ». Ячейка 41 проявляет себя только в отдельных случаях, например после переключения цветов. Ячейка 40 блокирует выведение на экран каждого второго символа (выводятся широкие символы), а ячейка 42 по достижении края экрана блокирует рулонный сдвиг и как бы организует режим «РП» на обычном экране. Ячейки графических режимов 50—52 хотя и переключают дисплей в режим текстовой графики, но при этом на экране остается тень символьного и графического курсоров, резко замедляется отработка команд текстовой графики и т. п.

- 57 (байт) — не используется.
- 60, 62 — вектор прерывания от клавиатуры* (кроме регистра «АР2», или «НР» на старой модели БК-0010 с пленочной клавиатурой). Приоритет данного прерывания — 4.
- 64—76 — не используются.
- 100, 102 — вектор прерывания от внешнего таймера. Вход данного прерывания выведен на контакт В1 разъема порта «УП» БК0010. Приоритет прерывания — 4.
- 104 (байт) — код последнего введенного с клавиатуры символа. Код заносится в ячейку, если разрешено прерывание от клавиатуры (в отличие от регистра 177662, где код появляется после нажатия клавиши независимо от того, разрешено ли прерывание). Ячейка широко используется в различных программах для организации режима «автоповтор» — многократного выведения символа на экран при удержании клавиши нажатой. Код символа в данной ячейке, в отличие от регистра 177662, — уже «обработанный», т. е. соответствующий регистру клавиатуры («РУС», «ЛАТ», «СТР», «ЗАГЛ»).
- 105 (байт) — признак записи кода в ячейку 104**. Используется командой EMT 6.
- 106 — буфер константы повтора (количество повторов «пустого» цикла SOB при нажатии клавиши «ПОВТ» перед выводом каждого следующего символа). В символьном режиме число в ячейке 106 равно 20000, в графическом — 2000, благодаря чему работа клавиши «ПОВТ» замедляется до приемлемого уровня. Пере-

* Стандартный адрес обработчика — 101136. — *Прим. ред.*

** При очистке данной ячейки командой CLRB @#105 достигается эффект «отмены» предыдущего нажатия клавиши, т. е. последующий вызов EMT6 всегда вызывает ожидание нового нажатия. — *Прим. ред.*

задав содержимое ячейки 106, можно изменить скорость работы в режиме «повтор» (минимальной скорости соответствует число 0*, максимальной — 1).

- 110 (байт) — признак повтора кода. Если значение в ячейке 110 не равно нулю, то код последнего введенного символа выводится повторно, после чего ячейка сбрасывается.
- 111 (байт) — счетчик табуляции. В соответствии с содержимым ячейки устанавливается количество пробелов, выводимых на экран. Используется для отработки табуляции по клавише «ТАБ».
- 112—120 — позиционный код маски табуляции. В соответствии с содержимым этих четырех слов (64д бита) устанавливаются метки табуляции в служебной строке и выполняется табуляция по клавише «ТАБ». Установленной метке табуляции соответствует единичный бит, сброшенной — нулевой. Это можно использовать для «принудительного» (программного) задания позиций табуляции, для чего достаточно занести в данные ячейки соответствующую маску. Для того чтобы метки табуляции индицировались в служебной строке, ее после записи маски в ячейки 112—120 необходимо «обновить», дав команду «установка индекса» (код 236).
- 122 (байт) — счетчик ключа. Количество символов, выводимое на экран при обращении к программируемому «ключу» клавиатуры (клавиши «0»...«9» по регистру «АР2»). В эту ячейку заносится длина текста текущего ключа.
- 123 (байт) — не используется.
- 124 — адрес очередного байта текста текущего «ключа».
- 126—150 — адреса (по порядку) начала текстов программируемых «ключей», соответствующих клавишам «0», «1»,...«9».
- 152 (байт) — признак нарушения рулона. Используется для организации рулонного сдвига экрана вверх и вниз.
- 153 (байт) — признак записи точки. Используется в программах командных прерываний EMT 30 и EMT 32.
- 154 (байт) — маска позиции графической точки (ГТ). Чтобы адресовать ГТ на экране в режиме текстовой графики, емкость машинного слова недостаточна, требуется еще как минимум три бита. Данная ячейка и является дополнением к машинному слову — адресу ГТ. Маска ГТ устанавливает положение графической точки в пределах выбранного байта экранного ОЗУ и может принимать значения 1, 2, 4, 10, 20, 40, 100, 200 (для черно-белого режима 512 точек) — единичный бит маски соответствует положению ГТ в пределах байта. В цветном режиме маска состоит не из одного, а из пары битов: 3, 14, 60, 300. Положение пары единичных битов кратно двум, т. е. в качестве маски используются попарно биты 00 и 01, 02 и 03 и т. д.
- 155 (байт) — начальная маска ГТ. Используется для задания начального значения маски в ячейке 154, которое может быть равно 1 или 3, в зависимости от режима (черно-белого или цветного).
- 156 — номер символа на экране. Номер первого символа (в верхнем левом углу) равен 0, последнего (в правом нижнем) — 2777 (для режима «64 символа в строке»). Нумерация позиций символов от сдвига рулона не зависит и всегда одинакова. Последний символ любой строки имеет номер, оканчивающийся на 77, первый — на 00. При переходе номера через нуль в двух младших разрядах драйвер дисплея организует перевод строки на экране. Проиллюстрировать эту функцию ячейки можно следующими программами:

```
0: CLR @#156
   EMT 6
   EMT 16
   BR 0

0: MOV #77,@#156
   EMT 6
   EMT 16
   BR 0
```

Функция перевода строк на экране нарушается: при работе первой программы курсор возвращается к левому краю экрана, но всего на одну телевизионную строку экрана ниже, а при работе второй символ вообще печатается только на одном месте в начале строки.

В цветном режиме содержимое ячейки 156 кратно двум. Его можно использовать в раз-

* Учитывая, что оператор SOB вначале вычитает единицу из значения регистра-счетчика цикла, легко понять, что исходное нулевое значение счетчика эквивалентно числу 200000 (т. е. 177777+1). Исходная единица, напротив, вызовет только один проход тела цикла. — *Прим. ред.*

личных программах, например, для индикации достижения текстом края экрана и организации автоматического перевода строки на принтере*.

- 160 — адрес текущего символа. В этой ячейке всегда присутствует абсолютный (с учетом рулонного сдвига) адрес верхнего левого байта формируемого символьного курсора. Если записать в эту ячейку другое значение, то последующие символы, выводимые по EMT 16 или по EMT 20, будут формировать строку начиная с этого адреса. Строка прервется, когда содержимое ячейки 156 в двух младших разрядах перейдет через нуль или в строке символов встретится невыполнимая в пределах одной строки команда, например «ВК» или «очистка экрана». При этом курсор вернется на прежнее место экрана (содержимое ячейки восстановится в соответствии с номером символа). Символ по заданному в ячейке 160 адресу может быть выведен и в ОЗУ пользователя, скажем, по адресу 30000. Хотя при этом он не виден, но может быть использован, например, для получения на экране его увеличенной «копии» (в различных заставках), «бегущей строки» и т. п.
- 162 — горизонтальный размер символа на экране, в байтах. В этой ячейке хранится величина горизонтального смещения курсора после формирования очередного символа. В режиме «64 символа в строке» она равна 1, в режиме «32» — 2. Меняя содержимое ячейки, можно получить оригинальные эффекты вывода символов «вразрядку», по вертикали, диагонали, назад:

```
0:    MOV #1001,@#162 Интервал вывода
      MOV #TEX,R1     Начало текста
      CLR R2          Признак конца
                      ;текста
      EMT 20          Выдать текст
      EMT 6     Ждать нажатия клавиши
      BR 0      Программа зациклена
TEX:  .A:Привет!
      .E
      END
```

Однако при использовании данного приема необходимо учитывать, что по достижении «физического» конца экрана (который при сдвинутом рулоне может прийтись на любое место) и попытке «вывода» символов в область ПЗУ компьютер зависнет, так как соответствие номера символа и рулона нарушается.

- 164 — количество знакомест экрана (3000 в обычном режиме и 400 в режиме «РП»).
- 166 — адрес (номер) байта текущей ГТ на экране. Не зависит от сдвига рулона, левый верхний угол экрана всегда 0, правый нижний — 35777. Может использоваться вместе с ячейкой 154 для вычисления (и, например, индикации) позиции курсора в режиме текстовой графики.
- 170 — абсолютный адрес байта текущей ГТ на экране. При начальном положении рулона меняется от 42000 до 77777. Меняется при сдвиге рулона.
- 172 — длина графического вектора. Используется при исполнении команд текстовой графики.
- 174 — счетчик телевизионных строк. Используется при перемещении графического курсора в пределах одного знакоместа, для организации рулонного сдвига экрана в режиме текстовой графики.
- 176 — буфер координаты X для EMT 30 и EMT 32.
- 200 — буфер координаты Y для EMT 30 и EMT 32.

Две последние ячейки могут широко использоваться в различных функциях графических редакторов и других программ с применением EMT 30 и EMT 32 для получения координат последней выведенной точки (конца вектора).

- 202 — адрес начала видеопамяти. Вне зависимости от рулонного сдвига экрана в этой ячейке всегда число 40000 (в режиме «РП» — 70000). Может использоваться как константа адреса конца ОЗУ пользователя, для определения режима (обычный или «РП») и т. п.
- 204 — относительный адрес начала экрана (или адрес конца служебной строки). В исходном положении рулона — 2000, меняется со сдвигом рулона. Иначе это число называется базой видеопамяти.
- 206 — длина видеопамяти в байтах, 40000 — в обычном режиме и 10000 — в «РП».
- 210 — длина рабочей области экрана в байтах (равна длине видеопамяти за вы-

* Содержимое ячейки 156 можно расценивать как значение координат текущей знаковой позиции, закодированной по формуле: $X+100_8*Y$ (для 64 символов в строке) или $2*X+100_8*Y$ (для 32 символов в строке). — *Прим. ред.*

четом длины памяти служебной строки), 36000 – в обычном режиме и 5000 — в режиме «РП» (так как в «РП» еще 1000 байт отводится на нижнюю черту — границу экрана). Все эти константы используются драйверами дисплея для адресации графических точек и символов на экране. В своих программах их можно использовать для автоматического учета границ экрана при переходе в режим «РП» и коррекции выводимых на экран изображений.

- 212 — маска цвета фона экрана. В БК-0010 реализованы четыре цвета — красный, зеленый, синий и черный, которым соответствуют маски 177777, 125252, 52525 и 0. Например, если фон экрана черный, то маска в ячейке 212 равна 0. Если экран инверсный в черно-белом режиме (или при красном цвете фона в цветном режиме), маска равна 177777 и т. д. Изменяя содержимое ячейки 212, можно менять цвет фона*, делать экран «полосатым» и т. п.
- 214 — маска цвета символа на экране. Маски символов точно такие же, как для фона. Помимо символов, текущая маска в ячейке 214 «отвечает» за цвета графических точек и векторов. Комбинируя различные маски (не только четыре вышеуказанные) по вертикали и горизонтали, можно значительно расширить цветовую палитру БК-0010, а задавая специально подобранные значения (например, 100001 в цветном режиме), можно «маскировать» выводимые символы. Вообще же, комбинируя маски фона и символа, можно получить самые разнообразные эффекты.
- 216 — маска цвета фона служебной строки.
- 220 — маска цвета символа служебной строки. Эти две ячейки играют для служебной строки ту же роль, что две предыдущие для всего экрана.
- 222 — счетчик кодов.
- 224 — счетчик установки индикаторов.

Две последние ячейки используются при выдаче в служебную строку индикаторов — надписей, указывающих на режимы дисплея: «РУС», «ЛАТ», «ГРАФ», «ЗАП», «СТИР», «БЛР», «ИСУ», «РЕД».

- 226—246 — не используются.
- 250—254 — буферы служебных констант драйвера телеграфного канала.
- 256 — ячейка, зарезервированная под копию порта вывода (драйверами монитора БК-0010 не используется).
- 260 — адрес дополнительной программы обработки прерывания от клавиатуры. Если занести в эту ячейку какой-либо адрес, то управление будет передаваться на него при нажатии на любую клавишу. При нормальном функционировании ЭВМ код в эту ячейку заносится автоматически драйвером клавиатуры (ячейка обнуляется, что означает отказ от вызова дополнительной программы и продолжение обработки драйвером нажатия клавиши).
- 262 — признак кода «ВК». Если в этой ячейке содержится нуль, то при нажатии клавиши «ВВОД» в программу передается код 12 (большинство режимов), иначе — код 15 (например, в ФОКАЛе).
- 264 — буфер стартового адреса. В эту ячейку заносится физический адрес последнего загруженного файла (независимо от адреса, указанного в оглавлении файла) и на него передается управление по директиве пускового монитора «S» (без указания адреса в явном виде).
- 266 — буфер длины массива. В эту ячейку заносится длина последнего считанного файла.
- 270, 272 — не используются (это адрес вектора прерывания IRQ3 ЦП, не задействованного на БК аппаратно).
- 274, 276 — вектор прерывания от клавиатуры по нижнему регистру** («АР2» на БК-0010.01).
- 300 (байт) — «флаг» фазы читаемого с МЛ сигнала, устанавливается при чтении равным 0 или 377 (прямой или инверсный сигнал).
- 301 (байт) — признак ошибки при работе драйвера магнитофона. Данная ячейка

* Отметим, что при записи в данную ячейку какого-либо нового значения фон экрана еще не изменяется. Изменится он лишь после того, как будет нажата, например, клавиша «СБР» (разумеется, прежняя информация при этом будет утеряна). Если же выводить на экран какие-либо символы, не очищая его, то фон каждого и этих новых символов будет соответствовать новому содержимому ячейки 212, а остальная часть изображения на экране не изменится. Этот прием весьма удобно использовать для организации меню и оконного интерфейса в вильнюсском БЕЙСИКе с помощью операторов POKE &O212,... и PRINT. Подробности см. в №3 за 1994 г., с. 37. — *Прим. ред.*

** Стандартный адрес обработчика — 101362. — *Прим. ред.*

дублирует ячейку 321 блока параметров EMT 36, причем она является первичной, а ячейка 321 — вторичной.

- 302 — признак фиктивного чтения файла драйвером магнитофона. При фиктивном чтении эта ячейка не равна 0.
- 304 — инкремент адреса массива при работе драйвера магнитофона. В БК-0010 содержимое этой ячейки при чтении файлов всегда равно 1 (при фиктивном чтении — 0). В случае его изменения в некоторых неграмотно написанных копировщиках возникает характерный дефект при загрузке файла — косая «сетка» на экране (отдельные байты, далеко отстоящие друг от друга) и порча отдельных байтов всех массивов, находящихся в памяти дальше начала загружаемого файла*.
- 306 — буфер адреса блока параметров EMT 36, дублирует содержимое регистра R1 и используется при работе драйвера магнитофона.
- 310 — буфер указателя стека, сюда заносится и отсюда восстанавливается (по клавише «СТОП» или по окончании работы EMT 36) адрес вершины стека при работе драйвера магнитофона.
- 312 — буфер контрольной суммы массива (файла). В эту ячейку заносится контрольная сумма при чтении или записи файла на МЛ. Кроме того, она используется как буфер загрузки при фиктивном чтении файлов.
- 314 — константа сравнения при чтении файла. Сюда заносится результат вычислений константы по установочной последовательности. Это число позволяет ЭВМ отличать при чтении файла нулевой бит от единичного. Используется драйвером магнитофона при чтении, может также использоваться для примерной оценки скорости при записи.
- 316 — не используется. Это единственная из неиспользуемых ячеек, содержимое которой не меняеется при исполнении EMT 14 (не считая ячеек стека)**. Особая просьба: не использовать эту ячейку в пользовательских программах, а оставить для системных драйверов, которые пишет автор этой статьи (драйвер спецформата магнитофона, драйвер дисковода и т. п.), иначе ваши программы просто будут несовместимы с этими системными программами и вам же будет хуже.
- 320—371 — блок параметров драйвера магнитофона. Назначение этих ячеек было подробно изложено при описании EMT 36.
- 372—777 — стек.

* * *

Наше повествование о системной области подходит к завершению. Осталось сказать несколько слов о стеке.

Как вы уже знаете, стек — это специально выделенная область памяти, куда сама ЭВМ или программы пользователя заносят данные, которые нужно сохранить (обычно временно). Адресуется стек чаще всего через специальный регистр R6, или SP, отличающийся от остальных регистров процессора тем, что его автоинкремент или автодекремент выполняется всегда на 2, независимо от того, работает команда со словом или с байтом.

Отсчет ячеек стека, в отличие от всех прочих участков памяти, ведется «сверху вниз», т. е. началом считается адрес 777 (или, в четных адресах, 776), а концом (или «дном») — адрес 372. Если программа, использующая стек, не работает с магнитофоном или организует блок параметров для него начиная не с адреса 320, а в другом месте ОЗУ, то стек может быть «продлен» до адреса 320.

Ячейка, адрес которой соответствует текущему содержимому SP, называется ВЕРШИНОЙ стека. Адрес вершины при работе со стеком не остается постоянным, а все время меняется. Но этот адрес всегда должен восстанавливаться при работе отдельных ветвей программы. Таким образом, если мы в начале программы заносим что-либо в стек, изменяя адрес его вершины, а затем программа ветвится, то мы должны восстановить адрес вершины до ветвления или предусмотреть одинаковый порядок его восстановления в каждой из ветвей (либо после выхода из них). Наруше-

* Ряд копировщиков, например HELP7, устанавливают значение этой ячейки равным 177777 (минус один), за счет чего реализуется загрузка файлов в экранное ОЗУ «в обратном порядке» — от правого нижнего угла к левому верхнему. — *Прим. ред.*

** В новой версии ANDOS v3.1 специально предусмотрены еще три пары ячеек, обладающие этим ценным свойством (в каждой паре первая ячейка сохраняет значение, а вторая — адрес, по которому это значение должно быть занесено): @#120244 и @#120246, @#120250 и @#120252, @#120254 и @#120256. — *Прим. ред.*

ние этого правила может привести к тому, что стек будет постоянно засоряться все новыми данными и рано или поздно затрет ячейки системной области. К чему это приведет, ясно без лишних комментариев*.

Стек в БК-0010 организуется по типу магазина для автоматического оружия, а обмен данных в нем — по принципу «первым вошел — последним вышел». Данные, введенные раньше, оказываются в стеке дальше от вершины (и «выше» в памяти), чем введенные позже, и данные, введенные последними, извлекаются из стека прежде всего. Но это всего лишь традиция и удобный прием обращения к стеку. Не составляет большого труда организовать хранение данных и по другому принципу, например «первым вошел — первым вышел» («очередь»), но это, как правило, менее удобно.

Обычно данные сохраняются в стеке с помощью стандартного приема — последовательной засылкой операторами MOV с автодекрементной адресацией регистра R6 (последовательное извлечение с помощью автоинкрементной адресации). По ходу изучения ассемблера приводилось столько примеров с использованием стека, что повторять их нет необходимости. Мы познакомились также еще с двумя возможностями сохранения и использования данных в стеке: с помощью косвенной индексации (чтобы «достать» данные, лежащие не на вершине) и малоупотребительной команды MARK. Но есть еще один способ использования стека для хранения данных — с помощью абсолютной адресации.

При реализации этого способа исходят из того, что для работы подавляющего большинства пользовательских программ такая «глубина» стека, которая имеется на БК-0010, не нужна. Вполне достаточно иметь «дно» стека по адресу 600, а иногда даже 700. При этом всю расположенную ниже область памяти можно рассматривать как системные ячейки, не используемые драйверами монитора БК-0010. Естественно, что любая из этих ячеек может быть использована программистом для своих нужд. Например, этот прием широко применяется в МИКРО.10К: начиная с адреса 400 по 600 многие ячейки используются для хранения констант и переменных ассемблер-системы. Перечислим некоторые из них, они могут быть полезными при работе с МИКРО.10К:

- 402 — адрес начала текста редактора,
- 404 — адрес текущего конца текста редактора,
- 426 — текущий адрес метки,
- 432 — текущий адрес курсора,
- 446 — адрес начала текущей страницы текста,
- 470 и далее — буфер маски поиска.

Разумеется, используются в МИКРО и другие ячейки. Благодаря такой организации системных переменных МИКРО.10К как программа, не имеющая внутри себя буфера переменных может с равным успехом работать и в ОЗУ пользователя, и в дополнительном ОЗУ, и в ПЗУ, что является примером рациональной организации инструментальных программ. Попутно укажем, как можно изменить МИКРО.10К таким образом, чтобы изменились параметры буферов текста и загрузочного модуля, но сначала сделаем одно замечание.

Многие решительно выступают против того, чтобы использовать выделенные ячейки стека (и вообще свободные ячейки системной области) для хранения переменных в программах пользователя, и не без оснований: если в памяти должны работать две программы и обе используют по роковому стечению обстоятельств одни и те же ячейки, то они будут мешать друг другу, и, скорее всего, ни одна из них нормально работать не сможет. Что можно на это возразить? Во-первых, вариант, когда в памяти БК-0010 работает сразу несколько программ, встречается не так уж часто. А во-вторых, где прикажете размещать переменные, если программа должна быть рассчитана на работу в ПЗУ? А этот вариант становится все более распространенным**. Словом, как уже было сказано раньше, вам самим решать, что и каким образом использовать. Автор же, придерживаясь демократических взглядов, от категорических рекомендаций и запретов предпочитает воздерживаться.

Но вернемся к обещанным рекомендациям по изменению МИКРО. В исходной версии МИКРО.10К-РП (или, равным образом, МИКРО.10-01К) адрес начала текста равен

* Еще хуже, если положение вершины стека нарушается внутри подпрограммы или драйвера обработки прерывания. Поскольку адрес возврата хранится в стеке, возврат в основную программу невозможен, если положение вершины стека перед выходом из подпрограммы (или драйвера) не будет тем же, что и после входа. — *Прим. ред.*

** Впрочем, если судить по все возрастающей популярности дополнительного (расширенного) ОЗУ, как в КНГМД, так и в виде отдельных блоков, этот вариант не станет единственно возможным. — *Прим. ред.*

17001, адрес загрузочного модуля (программы в кодах, получаемой в результате трансляции или трансляции и компоновки) — 13000. Таким образом, для текста программы выделено 20000 байт (последние 1000 байт ОЗУ ассемблер использует под таблицу меток), а под загрузочный модуль — 4000. Это достаточно рациональное распределение памяти, если ассемблер-система расположена в ОЗУ по адресу 1000. А если ассемблер размещается в ПЗУ (или дополнительном ОЗУ), например, по адресу 140000? Ясно, что первые 12000 байт ОЗУ пользователя при этом будут потеряны. Как перераспределить память?

В МИКРО.10К для этого есть три ячейки. Если обозначить адрес загрузки ассемблер-системы за 0, то в ячейке 6 содержится адрес начала текста редактора, в ячейке 32 — адрес начала загрузочного модуля, а в ячейке 36 — тот же адрес минус 1000 (используемый при вычислении адресов меток). При размещении МИКРО.10К в области ПЗУ целесообразно задать адрес текста равным 10001, адрес загрузочного модуля — 1000, а третью константу — 0. Разумеется, распределение памяти может быть и иным, в зависимости от целей применения МИКРО.10К. Например, если МИКРО используется только как текстовый редактор, можно задать адрес начала текста 1001, а две другие константы не изменять. При размещении МИКРО.10К в ОЗУ пользователя и применении его в качестве текстового редактора также можно увеличить буфер текста, задав адрес его начала (в ячейке 6), например, равным 13001. Кстати, единица в адресе начала текста МИКРО.10К совсем не обязательна, он может быть любым, просто такой адрес — опознавательный признак текстового файла МИКРО.10К и, кроме того, для нормальной работы текстового редактора желательно иметь перед буфером текста свободный байт, куда при запуске записывается код 12. После изменения параметров буферов МИКРО надо помнить, что для их реализации необходимо перезапустить МИКРО.10К с адреса загрузки.

* * *

Бывают случаи, когда стек, расположенный по адресам 372 ...777, начинает мешать работе. Допустим, нам надо скопировать файл длиной 77400 с адресом загрузки, равным 400 (такие игровые программы есть, в них подобный адрес и длина файла — своеобразный, но очень «слабый» способ защиты от копирования). Сделать это обычным копировщиком, размещаемым в ОЗУ, нельзя — все ОЗУ занимает копируемый файл. Но и обычный копировщик, размещенный в дополнительном ОЗУ или ПЗУ, тоже непригоден — мешает стек. Как быть? Нужен копировщик, размещаемый в дополнительном ОЗУ (например, с адреса 140000) и переносящий стек в эту же зону. А для переноса стека достаточно изменить содержимое регистра SP.

Еще один случай, когда надо представлять, что хранится в стеке, — это создание программ с автозапуском. Такие программы обычно имеют адрес загрузки меньше 1000 и после загрузки самозапускаются. Как это происходит? Довольно просто. При обращении к драйверу магнитофона по прерыванию EMT 36 в стек заносится адрес команды, следующей за вызвавшей прерывание. Если теперь мы загрузим программу, в начале которой (до адреса 1000) записан блок, состоящий из слов, содержащих адрес запуска, то адрес команды, следующей после вызова EMT 36, будет в стеке подменен. После возврата из прерывания (после загрузки файла) этот адрес по команде RTI попадает в PC и управление передается на запуск загруженного файла. В МСД исходный адрес вершины стека — 732, сюда обычно и заносят адрес запуска программы с автозапуском. В ПМ исходный адрес вершины стека — 1000 и адрес автозапуска обычно заносят по адресу 776 или 766 — в обоих случаях автозапуск «сработает». Автозапуск по ячейке 766 можно задать прямо в МСД, записав затем на МЛ файл с адреса, например, 760. Для задания автозапуска по другим ячейкам необходимы особые программные средства, например специальный копировщик*. Обычно такие средства заносят адрес запуска программы начиная с ячейки 732 (или даже с еще более «низких» адресов) до 1000, что обеспечивает автозапуск из любого режима.

Однако у автозапуска есть недостатки. Во-первых, после загрузки программа запускается без проверки на наличие ошибок чтения, а во-вторых, EMT 36, как правило, заканчивает при этом работу «нештатным» образом и дистанционное управление магнитофона не отключается, если только в запущенной программе пользователя не присутствует команда, делающая это взамен неоконченного EMT 36

* Впрочем, не обязательно «специальный». То же самое можно, при известном умении, сделать с помощью общераспространенных магнитофонных копировщиков HELP и HELP3, а также других. — *Прим. ред.*

(например, команда EMT 14). Разработаны (в том числе автором) несколько способов автозапуска, позволяющих устранить указанные недостатки, но на этом мы останавливаться не будем. Отметим только, что основаны они на том, что в начале программы помещается блок (при загрузке размещаемый еще «глубже» в стеке), на который автозапуск и передает управление, а уже этот блок делает все необходимое, в том числе и перемещение загруженной программы по рабочему адресу (что очень важно, если автозапуском снабжена программа, начинающаяся не с адреса 1000), а уже потом запускает ее.

В заключение следует сказать, что автозапуск довольно коварный прием и снабжать им все программы подряд не рекомендуется. Особенно это относится к системным и инструментальным программам, которые автозапуск может даже иногда сделать неработоспособными (например, если для этого требуется обязательный «штатный» выход из режима загрузки: запись в нужные ячейки всех переменных, байта ответа EMT 36, восстановление стека и т. п.).

Для чего предназначен ассемблер?

«Вот это да! — скажет иной читатель. — Сначала мы изучили язык, а потом задаем вопрос, зачем он нужен?!» Но вопрос этот не праздный. Хотя язык ассемблера относится к машинно-ориентированным языкам, а значит, по сути своей универсален, есть определенные классы задач, для решения которых он подходит больше, а для других — меньше. Что же это за задачи?

Поскольку программы, написанные на ассемблере, имеют максимальное быстродействие, ясно, что преимущества языка более выражены при операциях с большими массивами информации, а также при работе в реальном времени. Это может быть:

- пересылка массивов, в том числе текстов и изображений, их обработка и модификация;
- сортировка, перекодирование, перевод из одного формата в другой и обратно, упаковка и распаковка текстов и изображений, поиск в массивах заданной информации;
- логическая обработка информации (логические игры и головоломки, логическое моделирование процессов);
- скоростной анализ ситуаций и генерация динамических моделей (в играх и при управлении внешними объектами);
- связь с внешними устройствами в заданных форматах и с заданными скоростями обмена.

Теперь посмотрим, для решения каких задач ассемблер приспособлен меньше всего. Как известно, система команд процессора БК-0010 содержит только простейшие арифметические операции с целыми числами — сложение и вычитание. Не так уж трудно на их основе построить программы целочисленного умножения и деления, но даже построенные по оптимальным алгоритмам, эти операции будут выполняться на порядок медленнее, чем элементарные, а программы — занимать довольно большой объем памяти. Стоит ли говорить тогда о программах плавающей арифметики, вычисления функций и т. п.? Вряд ли удастся при их написании заметно улучшить алгоритмы аналогичных вычислений, имеющиеся в языках высокого уровня, а занимаемый объем ОЗУ при этом станет так значителен, что оставшаяся для других нужд память заметно сократится. В то же время выигрыш по быстродействию по сравнению с языками высокого уровня будет не так уж велик. Таким образом, ассемблер меньше всего подходит для решения задач вычислительного характера.

Но иногда включение вычислений в прочие программы на ассемблере бывает совершенно необходимо. Например, нередко в ходе управления объектами (или в играх) приходится вычислять траектории движения по формулам, генерировать псевдослучайные последовательности и т. п. Конечно, в наборе прикладных программ, разработанных для БК-0010, есть математические пакеты, реализующие любые функции плавающей арифметики (например, пакет ARPAC), и даже специальные языки — расширения ассемблер-системы (язык ALMIC, 1991 г.). Но, как уже отмечалось, программы плавающей арифметики работают относительно медленно (хотя, конечно, гораздо быстрее, чем, например, ФОКАЛ) и занимают много памяти. Чтобы не терять присущие ассемблеру преимущества — быстродействие и компактность программ, можно предложить несколько особых приемов.

Все вычисления нужно пытаться сводить к целочисленной арифметике в пределах одного машинного слова (однорегистровая арифметика). В самом деле, одно машинное слово может содержать число от 0 до 65535д, при этом его изменение на единицу дает относительную погрешность всего около 0,002% от конечного значения. Такая точность представления чисел (и даже на два порядка худшая)

вполне приемлема для подавляющего большинства практических задач. Но тогда остро встает вопрос о диапазоне представления чисел и порядке вычислений. Например для целочисленного умножения исходные числа не могут быть больше 377 каждое, а значит, точность их представления будет не выше 0,4%. Это пока еще допустимая точность для большинства задач управления объектами, особенно если учесть, что точность многих датчиков на порядок ниже. Но при операциях с такими числами возможна очень большая потеря точности, особенно при делении. Когда делимое и делитель одного порядка, погрешность результата в целых числах будет не меньше 10%. Поэтому при решении вычислительных задач в целых числах первостепенное внимание надо уделять порядку вычислений: вначале выполнять умножение, а потом деление, заботиться, чтобы не было деления меньшего числа на большее (что приведет к потере числа в дальнейших вычислениях) и т. п. Общее правило целочисленной арифметики вообще таково: надо стараться, чтобы при умножении и делении как исходные числа, так и результат лежали как можно ближе к границе переполнения (т. е. для двухбайтных чисел без знака — к 65536д), но переноса не происходило. Тем не менее при вычислении таким образом сложных функций, например тригонометрических, число последовательных умножений и делений будет так велико, что потеря точности станет значительной, а время вычислений — соизмеримым с таковым в языках высокого уровня. В этих случаях рекомендуется, когда это возможно, заменять вычисления функций по формулам на их табличное представление. Как это сделать? Предположим, необходимо вычислять значения синуса угла в диапазоне от 0 до 90°. Если точность представления аргумента не выше 1°, таблица значений такой функции займет всего 90 машинных слов (а если пойти на некоторое снижение точности, то 90 байт). Зная, что диапазон значений данной функции лежит в пределах от 0 до 1, сопоставим максимуму функции число 65535, а минимуму — 0. Промежуточные значения при заданном аргументе легко вычисляются умножением синуса на число 65535 и подставляются в таблицу. Например, синус 30° при таком представлении будет равен 0.5 * 65535 = 32768 (округленно). Вряд ли программа вычисления синуса с точностью до пятого знака уложится в такой же объем памяти, как таблица, а о сравнимом быстродействии не приходится и говорить. Если же необходимо большее число градаций аргумента, то не стоит увеличивать в десятки раз объем таблицы, а лучше всего, особенно для такой функции, как синус, прибегнуть к линейной интерполяции. Таким путем в большинстве случаев удается избегнуть громоздких и длительных вычислений по формулам. Если же, как часто бывает, нужен не весь диапазон представления аргумента, а только его часть (например, вычисление радиуса разворота самолета в зависимости от крена, допустимое значение которого не превышает 30°), то таблица станет еще короче либо может быть повышена точность. Не следует забывать и об элементарных преобразованиях функций, известных еще из курса средней школы, которые часто позволяют выразить одну функцию через другую и тем самым упростить вычисления или сократить количество функций, вычисляемых таблично.

Что же касается генерации случайных чисел, то тут есть очень простой и на первый взгляд неочевидный выход. Что представляет собой ПЗУ БК-0010, если не последовательность чисел, с точки зрения пользователя случайную? Специальные исследования показали, что если выбирать последовательные числа из ячеек ПЗУ, то после несложных преобразований их ряд хорошо удовлетворяет равномерному закону распределения (значительно лучше, чем, например, генератор случайных чисел ФОКАЛа). О возможностях же использования в качестве генератора случайных чисел системного таймера уже говорилось в разделе, посвященном системным регистрам.

Итак, хотя ассемблер — это язык, явно лучше приспособленный для логической, чем для математической, обработки информации, все же, прибегая к специальным приемам, вполне можно без потери присущего ему быстродействия выполнять с его помощью и вычисления.

«Стандартные» подпрограммы

Рассмотрим несколько подпрограмм на ассемблере, имеющих практическое значение и предназначенных для работы с числами. Для наиболее придирчивых (тех, кто любит искать чужие ошибки и неточности) автор сообщает, что эти программы (как и вообще все, приводимые в тексте) написаны им лично. (Хотя для простейших программ авторство — дело сомнительное и «скользкое»: уж больно велика вероятность попасть на проторенный путь, и тебя в два счета обвинят в плагиате... Да и вообще, существует ли авторское право на простейшие приемы программирования? Ав-

тор склонен думать, что нет, так как оно не имеет смысла — каждый легко может «переоткрыть» эти приемы сам, не «списывая» у другого. Тем не менее автор охотно выслушает замечания читателей.) Все эти программы тщательно проверены и отлажены на БК-0010, и, если не возникнет ошибок при их перепечатке, а затем при вводе с клавиатуры на ЭВМ читателя, все они должны работать. Как с наиболее простого примера, начнем, пожалуй, с генератора псевдослучайных чисел.

```
; Подпрограмма генератора
; случайных чисел
; Первое обращение к программе - RNZ,
; последующие - RND
; R4 - выход случайного числа,
; R5 - счетчик адреса
RND: ADD (R5)+,R4      Получение
     SWAB R4           очередного
     ADD 7(R5),R4      псевдослучайного
     DEC R4            числа
     ADD 15(R5),R4     в регистре R4
     CMP #137750,R5    Конец ПЗУ?
     BLO RNZ           Да - перезапуск
     RET               Выход
RNZ: MOV #100000,R5    Адрес начала ПЗУ
     BR RND            Генерация
```

Программа извлекает содержимое трех ячеек ПЗУ, «перемешивает» полученные числа и выдает результат в регистре R4. Используемый способ «перетасовки» трех чисел для получения псевдослучайной последовательности выбран в результате опытной проверки полученных рядов чисел на равномерность и случайность распределения. Программа дает непериодический ряд из примерно 8000 чисел, распределенных в диапазоне 0...177777. Обращение по метке RNZ необходимо для задания адреса начала ПЗУ, а также может использоваться при отладке, так как дает все время одно число.

Так же проста и программа умножения:

```
; Подпрограмма умножения
; 8-разрядных чисел
; R2-множимое, R3-множитель,
; R4-результат, R5-счетчик итераций
MUL: MOV #10,R5 Цикл из 8 итераций
     CLR R4     Очистить
                ;аккумулятор результата
1:   ASR R3     Очередной разряд
                ;множителя
     BCC 2      Если 0 - дальше
     ADD R2,R4  Иначе прибавить множимое
2:   ASL R2     Сдвиг множимого на
                ;один разряд
     SOB R5,1   Если не конец -
                ;в цикл
     RET        Выход
```

Алгоритм этой программы построен совершенно аналогично правилу умножения «в столбик» для двоичных чисел и, видимо, в пояснениях не нуждается.

Программа деления* уже заметно сложнее:

```
; Подпрограмма деления
; 16-разрядных чисел
; R2-делимое, R3-делитель, R4-частное,
; R5-счетчик итераций
DIV: CLR R5      Очистить счетчик
     CLR R4      и аккумулятор
                 ;результата
1:   INC R5      + итерация
     ASL R3      Делитель сдвинуть
                 ;влево
     BCC 1       Если не 1 - продолжать
                 ;искать начало делителя
2:   ROR R3      Возврат старшего
                 ;разряда
3:   INC R4      + 1 в разряд
                 ;частного
     SUB R3,R2   Вычитание делителя
     BCC 3       Если не конец -
                 ;продолжать
     ADD R3,R2   Возврат к
                 ;предыдущему
     DEC R4      значению делимого
     ASL R4      Следующий разряд
                 ;частного
     SOB R5,2    В цикл итераций
     ROR R4      Шаг назад на
                 ;один разряд
     RET         Выход
```

В строках 3...5 (имеются в виду номера строк по порядку, а не метки) происходит сдвиг делителя влево до первой единицы (т. е. до начала его значащей части) и одновременно определяется число итераций (последовательных вычитаний и сдвигов), необходимых для деления. Оно равно разности 16д и числа значащих разрядов делителя. Затем производится вычитание делителя из делимого, начиная со старших разрядов последнего, а когда результат становится отрицательным, происходит очередной сдвиг и вычитание продолжается в более младшем разряде. Количество

* Имеется в виду целочисленное деление, т. е. в R4 возвращается только целая часть результата. Остаток от деления после выхода содержится в регистре R2. — *Прим. ред.*

вычитаний, с учетом сдвига по разрядам, и есть частное. Остаток от деления сохраняется в R2. При желании можно по остатку произвести округление результата до целого, сравнив его с 1/2 делителя. Эту программу можно усовершенствовать и в другом направлении, например ввести защиту от деления на ноль. (Использованный алгоритм деления весьма распространен и носит название способа с восстановлением остатка.)

Надо отметить, что имеются частные случаи, когда нет нужды прибегать к столь сложным алгоритмам. Например, если заранее известно, что делимое и делитель будут отличаться не более чем на порядок, совсем не требуется производить последовательные вычитания со сдвигом. Вполне достаточно просто организовать вычитание до первого переноса. Программа при этом получится настолько простой, что ее нет необходимости здесь приводить. То же относится и к программе умножения, если множитель не превышает 10—20. Но необходимо помнить, что в неблагоприятных случаях такие простейшие программы работают очень плохо, например деление числа 177777 на 2 займет около 0,5 с*.

Однако все это — сами арифметические действия, а как ввести исходные данные с клавиатуры или увидеть результат на экране? Для этого тоже можно предложить ряд программ, например для ввода восьмеричного 6-разрядного числа с клавиатуры и вывода такого же числа на экран.

```
; Подпрограмма ввода восьмеричных
; чисел с клавиатуры
; Окончание ввода числа - клавиша
;  "ВВОД", число - в R4
INP:  CLR R4          Очистить буфер числа
1:    EMT 6           Ввод очередного
                      ;символа
      EMT 16          Выдать символ на
                      ;экран
      CMPB #12,R0     Символ "ВВОД"?
      BEQ 2           Да - выход
      ASL R4          Иначе - продвинуть
      ASL R4          предыдущее число
      ASL R4          в старший разряд
      SUB #60,R0      Перевести код
                      ;символа в число
      ADD R0,R4       Записать число в
                      ;буфер
      BR 1            Ввод следующего
                      ;символа
2:    RET             Выход
```

Программа эта простейшая и поэтому не лишена недостатков. Например, она не защищена от ввода нецифровых символов, не допускает исправлений, при вводе более чем шести символов действительны только последние шесть и т. д. При желании программу нетрудно дополнить всеми необходимыми сервисными функциями.

```
; Подпрограмма вывода на экран
; восьмеричных чисел в коде КОИ-8
; Исходное число - в регистре R4
OUT:  CLR R0          Очистить буфер цифры
      MOV #6,R5       Цикл чтения 6
                      ;разрядов
      BR 2            Чтение старшего
                      ;разряда числа
1:    CLR R0          Очистить буфер цифры
      ROL R4          Переслать 3-разрядное
      ROL R0          двоичное число,
      ROL R4          определяющее
                      ;очередную
      ROL R0          восьмеричную цифру,
2:    ROL R4          в буфер
      ROL R0          цифры
      ADD #60,R0      Перевести число в
                      ;код КОИ-8
      EMT 16          Выдать цифру на экран
      SOB R5,1        Цикл чтения разрядов
      RET             Выход
```

Эта программа также может быть усовершенствована. Например, иногда необходимо вместо незначащих нулей впереди числа выдавать пробелы, вместо числа без знака выводить число со знаком, интерпретируя его код как дополнительный, и т. п. Ценой некоторых (но не слишком больших!) усилий программиста все эти задачи могут быть решены. Часто нужен ввод и вывод не восьмеричных, а десятичных чисел. Реализовать такие программы несколько сложнее, но также вполне возможно. Вообще, как говорится, «лиха беда начало», и автор ничуть не сомневается, что читатели очень быстро добьются в программировании на ассемблере значительно больших успехов, чем он сам.

Можно отметить, что программы, вполне аналогичные двум последним, имеются в ПЗУ БК-0010. Например, подпрограмма ввода восьмеричного числа с клавиатуры имеется в ПЗУ монитор-драйверной системы, адрес обращения к ней — 100472. Она заносит число

* Напомним, что умножение и деление на 2 (или на число, равное 2 в степени N, где N — целое) на ассемблере реализуется сдвигом исходного числа вправо и влево командами ASL и ASR. — *Прим. ред.*

в регистр R5 и отличается от приведенной программы INP тем, что ввод любого «невосьмеричного» символа сбрасывает набранное число, так что набор нужно повторять. Эту подпрограмму можно применять во всех случаях, ведь ПЗУ МДС в БК-0010 имеется всегда. А вот подпрограмма вывода восьмеричного числа, хранящегося в R4, имеется только в тест-ПЗУ по адресу 163220 и при его отсутствии (без блока МСТД на БК-0010.01) программы с ее использованием работать не будут. Ясно, что ее применение не всегда корректно.

В результате разработки (обычно по мере необходимости) различных программ у каждого программиста не только появляются свои излюбленные приемы, «фокусы» и прочее, определяющие так называемый «почерк» программиста (чего почти лишены программы на языках высокого уровня), но и постепенно накапливаются библиотеки стандартных подпрограмм, подобных приведенным выше. Они могут быть использованы в разных программах по мере надобности, а хранить их лучше на МЛ отдельно, в виде листингов. Возможности ассемблера «МИКРО.10К» (других — в меньшей степени) позволяют при необходимости «догрузить» нужный программный модуль и «переставить» его в требуемое место программы, тем самым избавляя программиста от труда вводить его заново. Поэтому рекомендуется снабжать каждый такой модуль-подпрограмму уникальной меткой-именем, как это и сделано в наших примерах*.

Контрольные вопросы и задания

1. Принесло ли вам пользу знакомство с ячейками системной области БК-0010?

— Каждый отвечает так, как считает нужным.

*2. Напишите программу генератора случайных чисел, использующую системный таймер БК-0010. Выдача каждого следующего числа на экран должна происходить при нажатии клавиши, причем использовать EMT 6 для ожидания нажатия нельзя — найдите другой прием. Диапазон выдаваемых случайных чисел — от 0 до 377**.*

```
—       OUT=163220    Адрес подпрограммы
                      ;вывода
        RT1=177706    Регистры
        RT2=177710    системного
        RT3=177712    таймера
        SWU=177716    Регистр системных
                      ;устройств

        MOV #377,RT1      Исходное
                          ;число счетчика
        MOV #20,RT3       Множитель "x1"
0:      BIT #100,SWU      Клавиша
                          ;нажата?
        BNE 0             Нет - ждать
1:      BIT #100,SWU      Клавиша
                          ;отпущена?
        BEQ 1             Нет - ждать
        MOV RT2,R4        Извлечь число
                          ;из счетчика
        CALL OUT          Выдать число
                          ;на экран
        BR 0              Программа
                          ;зациклена
        END
```

Обратите внимание, что все адреса ПЗУ и системных регистров заданы с помощью операторов прямого присваивания.

3. Какой прием вы можете предложить для перевода восьмеричного числа в десятичное?

— Простейший прием такой. Нужно вычислить значения «круглых» десятичных чисел — 10, 100, 1000, 10000 в восьмеричном представлении и затем вычитать их из исходного числа, начиная с самого большого для одного регистра — с 10000, подсчитывая каждый раз количество вычитаний, пока еще нет переноса. Это количество, переведенное в код КОИ-8 (если его нужно выдать на экран), и будет десятичным разрядом. Затем нужно восстановить остаток и продолжить вычитание уже следующего числа, например 1000, и так до 1. (Восьмеричные эквиваленты приведенных десятичных чисел — 23420, 1750, 144, 12.)

4. Написать программу, обеспечивающую перекодирование десятичного числа из его текстовой записи (т. е. строки символов «0»..«9», вводимых один за другим с клавиатуры) в числовое содержимое регистра R2.

---

* Другой способ реализации библиотек стандартных функций, возможно более удобный на практике, — создание подгружаемых при компоновке объектных модулей. Подробнее об этом см. в №1 за 1994 г., с. 37. — *Прим. ред.*

** На некоторых экземплярах БК-0010(.01) системный (встроенный) таймер отсутствует или неисправен, в этом случае предложенная ниже в качестве ответа на данный вопрос программа не будет работать вовсе или могут наблюдаться различные отклонения от ее правильного функционирования (например, генерируемые случайные числа могут превышать #377). — *Прим. ред.*

```
—  PAC:CLR R2          Очистить
                      ;счетчик
   0:  EMT 6          Очередной
       EMT 16         символ
       CMPB #12,R0    Окончание
                      ;ввода?
       BEQ 1          Да!
       SUB #60,R0     ASCII-код ->
                      ;в цифру
       BCS 2          Код меньше
                      ;нуля
       CMP R0,#12     Цифра?
       BCC 2          Нет, код
                      ;больше девяти
       ASL R2         Число
                      ;умножаем на 2
       MOV R2,-(SP)   Сохраняем
       ASL R2         Умножено на 4
       ASL R2         Умножено на 8
       ADD (SP)+,R2   X*8+X*2 =X*10
       ADD R0,R2      Прибавить к
                      ;счетчику
       BR 0           Продолжать
   2:  CLR R2         Если
                      ;некорректно, то 0
   1:  RET            и выход
```

Трансляция и компоновка

В свое время мы уже кратко познакомились с тем, как подготовить программу к работе — оттранслировать и скомпоновать ее, и обещали еще раз вернуться к этой теме. Так мы и сделаем, но напомним, что наше описание базируется на ассемблер-системе «МИКРО.10К» и все приводимые далее адреса и директивы относятся прежде всего к ней. Для прочих версий они могут отличаться от указанных, хотя принципы работы те же самые.

Пусть у нас написан текст программы на языке ассемблера. В его состав могут входить:

- операторы прямого присваивания,
- метки,
- операторы,
- операнды с различными способами адресации,
- псевдооператоры с наборами символов,
- комментарии.

Весь этот чрезвычайно разнородный материал (размещенный в памяти по адресам 17001...37000) ассемблер должен превратить в программу в машинных кодах, готовую к исполнению, — ЗАГРУЗОЧНЫЙ МОДУЛЬ, — «собрать» из них программу. Отсюда и название системы: «ассемблер» означает «собиратель». В общих чертах мы уже знаем, как это делается, но рассмотрим этот процесс подробнее.

Обычно любой ассемблер выполняет трансляцию за два ПРОХОДА (или просмотра) исходного листинга программы. Первый проход называется ТРАНСЛЯЦИЕЙ и может сам состоять из ряда операций, но проще рассматривать его как одно целое.

Запуск транслятора осуществляется из его встроенного монитора по директиве «СО». В процессе трансляции производится перевод текста программы в машинные коды и присваивание меткам определенных адресов. По мере перевода текста в машинные коды производится запись последних в память ЭВМ (начиная с адреса 13000), и встречающиеся по ходу трансляции метки располагаются уже не просто в определенных строках программы, а по определенным физическим адресам. Эти адреса (вместе с именами меток в коде RADIX—50) заносятся в ТАБЛИЦУ МЕТОК, формируемую с адреса 40000 «вниз», т. е. в сторону меньших адресов. Если меток очень много, таблица может стать слишком длинной и затереть при трансляции конец текста программы раньше, чем он будет прочитан транслятором. Естественно, при этом возникает ошибка и трансляция прекращается. Избежать чрезмерного роста таблицы можно путем широкого использования ЛОКАЛЬНЫХ меток, так как таблица локальных меток формируется отдельно и существует в памяти только до появления очередной обычной метки.

Если все-таки таблица меток затирает текст (его на всякий случай перед трансляцией нужно всегда записывать на МЛ), то такую программу оттранслировать невозможно, ее необходимо либо разделить на части, либо перейти в режим «РП». Любая версия ассемблера «МИКРО.10К» рассчитана на работу в этом режиме (кроме редактирования текста), а в последних версиях — «МИКРО.10К-РП» и «МИКРО.10-01К» в режиме «РП» работает и редактор. В этом режиме можно транслировать также тексты программ, составленные из нескольких частей, «сливая» их друг с другом путем догрузки директивой «LF». Естественно, что при этом оператор END должен стоять только в конце последнего блока, а общая длина текста может достигать 50000. Во многих случаях такое слияние текстов программ значительно удобнее компоновки через объектные модули, о котором речь пойдет дальше. Чтобы ассемблер нормально работал в режиме «РП», его необходимо либо запустить в этом режиме с адреса 1000, либо, перейдя в режим «РП» в мониторе, дать команду «RS» (версия

«МИКРО.10К-РП» в этом не нуждается — достаточно перейти в режим «РП» в мониторе ассемблер-системы). Текст программы в памяти при этом, конечно, будет уничтожен. В режиме «РП» таблица меток формируется с адреса 70000.

В результате трансляции (первого прохода) образуется так называемый ОБЪЕКТНЫЙ МОДУЛЬ. Он представляет собой программу в машинных кодах, в которую занесено все необходимое, кроме некоторых операндов, под которые ячейки памяти только ЗАРЕЗЕРВИРОВАНЫ. Дело в том, что при первом проходе транслятор еще «не знает» реальных физических адресов меток, ведь таблица только формируется, поэтому и не может записать операнды, обращение к которым требует знания этих адресов. Такие операнды, ячейки под которые зарезервированы (туда заносятся нули), транслятор отмечает в выдаваемой на экран по окончании трансляции таблице меток инверсией, с указанием адресов зарезервированных ячеек и имен меток, входящих в состав операнда. Наличие в таблице после трансляции инверсных меток однозначно свидетельствует, что программа неперемещаема и нуждается в КОМПОНОВКЕ — это второй проход транслятора. Объектный модуль, полученный при трансляции, может быть после нее записан на МЛ директивой «SL», при этом вместе с ним записывается и таблица меток. Ассемблер дает возможность загрузить с МЛ и скомпоновать вместе несколько объектных модулей, что позволяет получить (в режиме «РП», разумеется) загрузочный модуль длиной до 16 кб. Но пока рассмотрим случай, когда у нас один модуль.

Итак, объектный модуль в памяти, таблица меток — на экране (если таблица меток занимает более 24 строк в обычном режиме или более четырех строк в режиме «РП», ее выдача на экран производится по частям, с остановками до нажатия любой клавиши). Раз в таблице есть инверсные метки, значит, программа нуждается в компоновке. При компоновке не только производится вычисление и запись в загрузочный модуль недостающих операндов, но и ЗАДАНИЕ АДРЕСА ЗАГРУЗКИ, т. е. того адреса, по которому программа будет загружаться и работать в дальнейшем. Этот адрес может быть задан любым, ассемблер внесет соответствующие поправки во все адреса меток программы, как бы «переместив» ее в памяти (в действительности загрузочный модуль остается на месте, по адресу 13000), поэтому наш ассемблер называется ПЕРЕМЕЩАЮЩИМ. Директива «LL» задает компоновку ПО УМОЛЧАНИЮ — по последнему указанному ранее адресу (после запуска ассемблера это адрес 1000), а директива «LS» запрашивает новый адрес компоновки у пользователя. После окончания компоновки инверсных меток в таблице (она выдается на экран заново) не должно остаться, а если они все же есть, значит, в программе имеются ошибки. Это либо неопределенные метки (т. е. обращение к метке есть, а ее самой нет), либо обращение к локальной метке «через обычную» (в скобках заметим, что такое обращение «вперед» допустимо, но лучше его избегать, это только запутывает программу).

После того как получен загрузочный модуль (в таблице меток нет инверсных имен), его можно записать на МЛ директивой «SA». При этом учтите, что располагается он с адреса 13000 независимо от заданного адреса компоновки, с этим начальным адресом он и будет записан на МЛ. При загрузке такой программы для работы нужно всегда явно задавать адрес загрузки, если он отличается от 13000, а так как удобнее грузить программу «по умолчанию», то лучше сразу переслать ее директивами МСД по нужному адресу, а потом уже записать на МЛ, для чего надо выйти из ассемблера, нажав клавишу «СТОП». Длина загрузочного модуля выдается при трансляции и компоновке вместе с таблицей меток, адрес тоже известен — 13000, так что переслать программу по необходимому адресу и затем записать ее на МЛ не составит труда, только для этого надо выйти в МСД (значит, блок МСТД должен быть заранее подключен, а способ перехода в МСД, если вы загружали «МИКРО» из ПМ, уже приводился).

Можно также сразу запустить загрузочный модуль на исполнение директивой «RU», при этом управление просто передается по адресу 13000. Учтите только, что для этого компоновка должна быть произведена именно по этому адресу, а ассемблер в памяти вполне может быть испорчен («затерт») работающей программой. Такой прием (прямой запуск) часто используется для отладки, иногда специально для этого даже изменяют некоторые параметры программы, чтобы она могла работать по адресу 13000 без затирания ассемблер-системы. Это действительно очень удобно: запустив программу по «RU» и проверив результаты ее работы, можно вернуться в ассемблер-систему по «повторному входу» (адрес запуска 1002), войти в редактор, внести в текст изменения, снова оттранслировать и скомпоновать, вновь запустить и т. д., пока работа программы не будет удовлетворительной. За-

тем можно внести в текст программы окончательные изменения для работы по нужному адресу и скомпоновать ее, задав этот адрес. Такая методика отладки особенно удобна, если длина загрузочного модуля невелика (до 4000) и он не затирает при трансляции начало текста программы, — это не сложнее отладки программ на БЕЙСИКе или ФОКАЛе.

А как быть, если программа длиннее и ее текст не помещается в памяти ни в обычном режиме, ни в «РП»? Тогда ее нужно разделить на несколько частей, по возможности таких, чтобы их можно было отладить отдельно. После проверки каждой части ее транслируют и записывают объектный модуль на МЛ. Каждая часть при этом должна содержать в начале необходимые операторы прямого присваивания (относящиеся к меткам, используемым в ней), а в конце оператор END.

Когда получены и записаны на МЛ все необходимые объектные модули программы, даем директиву «LA», указываем адрес компоновки и загружаем первый модуль. В дальнейшем грузим остальные модули по команде «LI». Каждый вновь загруженный модуль компонуется с уже имеющимися в памяти. После загрузки всех модулей (непременно в том порядке, в котором они должны входить в программу!) загрузочный модуль готов, его можно записать на МЛ или запустить. Перед компоновкой модулей с МЛ нужно перезапустить ассемблер командой «RS».

Ошибки при программировании на ассемблере

Синтаксические ошибки в тексте программы выявляются обычно на этапе трансляции. При этом трансляция прекращается, на экран выдается строка с ошибкой и запрос: «E,C?», т. е. «перейти к месту ошибки или продолжить?». Команда «C» вызывает продолжение трансляции независимо от того, что строка с ошибкой оттранслирована не будет. Это может потребоваться для выявления остальных ошибок или для иных целей. В конце такой трансляции таблица меток не выдается. Команда «E» вызывает переход в редактор, при этом курсор указывает на место ошибки. Если это «ошибка 3» (велика длина перехода), то указывается не метка, при переходе к которой допущено превышение длины перехода в SOB, BR или другом операторе ветвления, а сам оператор, т. е. источник обращения. В некоторых версиях «МИКРО.10К» была введена «ошибка 15» (затирание текста программы при трансляции). Но нужно сказать, что эта ошибка только констатирует факт, но не защищает текст от затирания, так как она выдается ПОСЛЕ трансляции последнего оператора и при этом часть текста неизбежно затирается. Но если это был обычный оператор, затертой окажется небольшая часть текста, которую легко восстановить. Если же последним был, например, оператор .+10000, то будет утерян текст длиной 10000, несмотря на останов по ошибке. Обычная реакция на эту ошибку — команда «C», при этом трансляция идет нормально до конца. В версиях «РП» «ошибка 15» исключена, зато добавлена «ошибка 12» — переопределение локальной метки, которая раньше не индицировалась и доставляла много хлопот.

Не совсем приятная ошибка — обращение к неопределенной локальной метке. При этом в таблице меток возникает путаница — появляются инверсные метки без имени, а только с адресом, или метки с именами, которых нет в тексте. Но если знать эти характерные признаки данной ошибки, найти ее не так сложно.

Сложнее дело обстоит с логическими ошибками, когда программа формально написана правильно, но работает не так, как ожидалось. В этом случае, если обычный просмотр текста и проверка алгоритма не выявляют ошибок, следует разбить программу на функциональные блоки и проверять их по отдельности. Частая ошибка — неправильное ветвление программы. Она может быть вызвана непониманием логики работы программы, неправильным оператором (например, применение оператора «со знаком», когда нужен «без знака») или ошибками при сравнении операндов. На последнем случае стоит остановиться подробнее.

Когда ветвление задается после оператора сравнения, то оно может происходить неправильно по следующим причинам:

неправильный порядок записи операндов (нужно помнить, что при сравнении из ПЕРВОГО операнда вычитается ВТОРОЙ, а при вычитании — наоборот);

- сравнение слов, когда надо сравнивать байты, и наоборот;
- искажение одного из сравниваемых операндов за счет РАСПРОСТРАНЕНИЯ ЗНАКА после выполнения оператора MOVB с одним из регистров;
- использование оператора ветвления с иной, чем нужно, логикой работы, например BLO вместо BLOS, BGT вместо BGE, BEQ вместо BNE и т. п.;
- недооценка возможного диапазона представления операндов, например когда при возрастании операнда происходит переполнение и изменение знака.

Кроме перечисленных возможны, конечно, и другие причины, но эти — самые распространенные. Не менее частая причина ошибок ветвления — передача управления «не туда», т. е. когда соответствующая метка находится не на том месте, где должна бы быть. Например в цикле может все время повторно заноситься исходное значение цикловой переменной, обнуляться регистры или повторно задаваться исходный адрес массива и т. д.

Уточнить, как именно происходит ветвление и где программа «зацикливается», а также отследить прочие «сомнительные точки» можно разными приемами. Нагляднее и проще всего на время включать в текст программы в точках перехода по ветвлению операторы, действие которых сразу заметно: HALT, звуковой сигнал, выдача на экран каких-либо символов и пр. Можно временно исключать из текста отдельные строки, «закрывая» их знаком комментариев (поставив в первой позиции символ «;» или вписав в начале строки оператор NOP), можно заменять операторы условного ветвления на безусловные и т. п.

Можно прибегнуть и к помощи отладчика. При пошаговой работе программы можно наблюдать содержимое регистров и т. д. Особенно интересен «ОТЛАДЧИК.К» С. А. Кумандина, который позволяет, кроме регистров, задать индикацию еще до пяти произвольных ячеек памяти, имеет RADIX-ДАМП и прочие сервисные функции. При работе с любым отладчиком затруднение нередко вызывает то, что командные прерывания EMT и TRAP отрабатываются как одна команда. Это происходит потому, что при прерывании ССП заменяется и Т-разряд сбрасывается. Как сделать эти команды «видимыми» при отладке, мы уже обсуждали ранее: чтобы посмотреть, что делается «внутри» прерывания, нужно предварительно записать во втором слове вектора данного прерывания ССП с установленным Т-разрядом. Но, вообще говоря, возможности любого отладчика довольно ограниченны и эта «техника» (во всяком случае, по мнению автора) более подходит для того, чтобы «копаться» в чужих программах, а не для отладки собственных.

Когда программа в общих чертах отлажена и вроде бы работает так, как хотелось, еще нельзя считать работу оконченной. Программу необходимо протестировать. Если она не очень сложная, нужно задавать такие исходные данные, чтобы (с учетом логики обработки) были пройдены по возможности все ветви программы, и перебрать все возможные комбинации данных. Сделать это тому, кто сам написал программу, не так уж сложно, как кажется. Если трудно представить себе последовательность обработки данных в сложной программе, можно ограничиться заданием трех значений для каждого вида входной информации: двух крайних и одного произвольно выбранного в середине диапазона (речь идет не только о числовых данных, но и, например, о директивах, позициях меню и т. п.). Если диапазон входных данных ограничен, необходимо задать еще как минимум два их значения за пределами диапазона с обеих сторон. Но, несмотря на все ухищрения, исчерпывающая проверка сложной программы невозможна в принципе и зачастую только после длительной эксплуатации выявляются ошибки, не обнаруженные в процессе отладки. Такая ошибка всегда очень неприятна, прежде всего потому, что программист уже успел забыть детали реализации алгоритма, ему трудно снова «вжиться» в программу и сообразить, что именно привело к ошибке. В этих случаях нельзя переоценить значение сохранения листинга с подробными комментариями, особенно если данная программа не единственная разработка программиста. Поэтому лишние пять минут, потраченные на написание комментариев, могут в будущем сберечь вам многие часы и даже дни работы. Вероятность же такой ситуации очень велика, ибо давно известное шуточное правило гласит: «Всякая отлаженная программа содержит как минимум одну ошибку. Исключением является только программа, состоящая из единственного оператора END».

Позиционно-независимое программирование

Обычно программа, написанная на языке ассемблера, предназначена для работы в определенной зоне адресов. Но есть программы, которые просто обязаны работать по любым адресам: это отладчики, дезассемблеры, супервизорные драйверы и прочие «инструменты», нужные не сами по себе, а для работы с другими программами. При этом заранее неизвестно, где будет расположена программа, нуждающаяся в отладке или дезассемблировании, поэтому мы вынуждены размещать инструментальную или системную программу «на свободном месте». Как добиться перемещаемости программ? Для этого следует придерживаться ряда принципов, носящих назва-

ние принципов ПОЗИЦИОННО-НЕЗАВИСИМОГО ПРОГРАММИРОВАНИЯ*.

Все адреса переходов и обращений к тексту программы надо задавать только путем относительной адресации, в виде меток.

Все буферы данных внутри программы нужно выделять тоже путем относительной адресации, по меткам, т. е. делать их перемещаемыми.

Ко всем ячейкам ПЗУ и системной области необходимо обращаться только по абсолютным адресам или по меткам, которые определены оператором прямого присваивания.

Не выделяйте в ОЗУ (кроме системной области) буферы и ячейки с абсолютными адресами, так как при размещении программы в этой зоне она может быть затерта собственным буфером.

Если необходимо определить физический адрес обращения к ОЗУ (или к ячейкам в самой программе), например при задании векторов прерываний, адресов текстовых сообщений и т. п., то вычислять эти адреса нужно пользуясь псевдооператором «.@».

Старайтесь чаще использовать регистры общего назначения, стек и косвенную адресацию через регистры и метки, что обеспечивает независимость от адресов размещения программы.

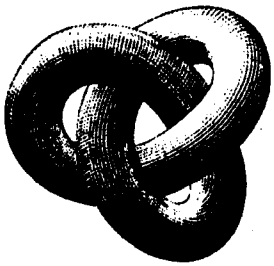
Перечень этих принципов читатель может продолжить самостоятельно.

Признаком ПЕРЕМЕЩАЕМОСТИ является то, что программа не нуждается в компоновке, т. е. таблица меток, выданная транслятором (после первого прохода), не содержит инверсных имен (хотя это и не абсолютная гарантия перемещаемости!). Перемещаемые программы обычно занимают заметно больше места в ОЗУ, чем неперемещаемые, и несколько сложнее по структуре, поэтому стремление делать перемещаемыми все программы вряд ли оправдано.

Особый случай — размещение программы в ПЗУ. Обычной перемещаемости здесь недостаточно, нужно еще позаботиться о том, чтобы внутри программы не было никаких переменных, буферов данных и т. п. Кроме того, если такая программа рассчитана на работу как в ОЗУ, так и в ПЗУ и предназначена для работы в реальном времени, необходимо учитывать, что одна и та же программа при размещении в ПЗУ работает на 10—30% БЫСТРЕЕ, чем в ОЗУ пользователя. Например, для драйвера магнитофона эта разница может оказаться критической, если не предусмотрены специальные меры, обеспечивающие приемлемый эффект при изменении скорости работы. Лучше всего, чтобы после запуска программа сама определяла, где она находится, и в соответствии с этим организовывала свои буферы данных, самозащиту, регулировала скорость работы и т. п.

(Окончание следует)

* Более подробно о перемещаемости программ и методике обеспечения этого свойства можно прочитать в статье А. Г. Прудковского, в №4 за 1994 г., с. 61. — *Прим. ред.*



В статье подробно описывается реализация вызова подпрограмм в машинных кодах (операторы USR и DEFUSR БЕЙСИКа БК-0010.01), а также очень интересные возможности передачи в USR-подпрограмму и возврата из нее нестандартных данных, в том числе массивов любого размера.

Д. Ю. Усенков,

Москва

Передача данных из вильнюсского БЕЙСИКа в подпрограммы в машинных кодах

Ни для кого не секрет, что в вильнюсском БЕЙСИКе БК-0010.01 отсутствуют многие полезные функции (например, SOUND и PLAY, операции со спрайтами и т. п.). Подавляющее большинство из них можно реализовать только с помощью подпрограмм в машинных кодах, вызываемых посредством оператора USR. Однако в руководстве по БЕЙСИКу этому оператору уделено слишком мало внимания, поэтому его использование вызывает у начинающих значительные трудности. Кроме того, как оказалось, USR-функции обладают возможностями, намного превышающими описанные в руководстве. Вот хотя бы такой, казалось бы, простой вопрос, как передача данных из программы на БЕЙСИКе в USR-подпрограмму и обратно.

Немного теории

Прежде чем начать разговор об USR-функциях БК-0010, следует разобраться, какие вообще существуют способы передачи данных в подпрограммы, а также обсудить используемую в дальнейшем терминологию.

В языках программирования высокого уровня известно несколько основных способов передачи данных в подпрограммы (последние могут быть представлены как отдельными вставками в машинных кодах или составными частями подгружаемых библиотек подпрограмм, так и процедурами самой программы на языке высокого уровня), но наиболее удобным и чаще всего используемым вариантом является пересылка данных через стек. Именно так поступает и вильнюсский БЕЙСИК — указанный в скобках аргумент оператора USR переписывается в стек, а для удобства программиста абсолютное значение адреса его расположения в стековом ОЗУ заносится в регистр R5. Тем же путем, через стек, производится возврат результата работы USR-подпрограммы, для чего необходимо записать его вместо аргумента по адресу, указанному в R5.

Данные, которые могут быть переданы в подпрограмму, можно разделить на две группы: передаваемые «по значению» и «по ссылке». В первом случае в стек копируется значение аргумента (константа или содержимое переменной). Очевидно, что, так как подпрограмма работает лишь с копией аргумента, записанной в стеке, никакие производимые ею действия не меняют «истинного» значения аргумента в основной программе. Это позволяет реализовать хорошо известный программистам на «больших» компьютерах механизм формальных и фактических параметров, что, в свою очередь, обеспечивает независимость подпрограмм от основной программы.

Но как быть, если нужно передать в подпрограмму массив, структуру (в СИ или

ПАСКАЛе) или текстовую строку (которая может рассматриваться как одномерный массив байтов — кодов символов). Копировать их в стек, как мы это делали с отдельными переменными, крайне нерационально: для длинных строк и больших массивов могло бы не хватить объема стека (или появилось бы ограничение на максимальные размеры строк и массивов). Очевидный выход из положения — передавать вместо самого аргумента (например, массива) адрес его первого элемента в ОЗУ и количество элементов. Для текстовой строки, соответственно, можно передавать ее начальный адрес и длину (эти две характеристики текстовой константы или содержимого текстовой переменной называются ее дескриптором). Такой способ передачи и носит название «по ссылке», ведь мы передаем в подпрограмму не сам аргумент, а ссылку (указатель) на него.

В последнем случае, как легко заметить, подпрограмма будет работать непосредственно с «оригиналом» аргумента, хранящимся в основном ОЗУ, а не с его копией в стеке. Следовательно, любые его изменения «автоматически» переходят и в основную программу. С одной стороны, это делает ненужным отдельный возврат результатов работы (указатель передается только как ВХОДНОЕ значение), а с другой — требует известной осторожности при программировании, так как подпрограмма может «нечаянно» испортить какие-то нужные данные основной программы. (Кстати говоря, в вильнюсском БЕЙСИКе БК-0010.01 использован вариант именно такого метода, из-за чего подпрограммы GOSUB не требуют отдельной передачи параметров, но зато вынуждают использовать переменные с другими, нежели в основной программе, именами.)

Немного забегая вперед, приведем два примера, наиболее наглядно демонстрирующих свойства механизмов передачи данных по ссылке и по значению (предполагаем, что подпрограмма здесь USR1 обнуляет переданный ей числовой аргумент, а USR2 заменяет все байты текстовой строки на код символа «*»):

PRINT USR1(A%) — на экран выводится возвращенное подпрограммой нулевое значение, но так как числовые данные передаются в USR-подпрограмму по значению (о чем будет рассказано чуть позже), то содержимое переменной A% не изменится (это можно проверить с помощью оператора PRINT A%);

PRINT USR2(A$) — на экран выводится возвращенная подпрограммой строка звездочек, количество которых равно длине исходного содержимого переменной A$. Кроме того, так как текстовые строки передаются в USR-подпрограмму по ссылке, содержимое A$ также будет заменено на звездочки, хотя в данной команде не содержится операции присваивания. В этой замене можно убедиться, введя команду PRINT A$.

Стандартный механизм вызова USR-подпрограмм

«Вооружившись» терминологией и необходимым теоретическим багажом, перейдем к рассмотрению механизма вызова и передачи данных, реализованного разработчиками вильнюсского БЕЙСИКа БК-0010.01 для USR-подпрограмм.

В общем виде выполняемая БЕЙСИК-транслятором последовательность действий следующая:

- занесение в стек данных, передаваемых в подпрограмму;
- запись вспомогательных значений в регистры R3 и R5;
- передача управления подпрограмме в машинных кодах;
- возврат после выхода из подпрограммы на исполнение «шитого кода» БЕЙСИКа и извлечение результата из стека.

Рассмотрим эти этапы более подробно на примере передачи аргумента целого типа. При отработке оператора USR БЕЙСИК прежде всего записывает в вершину стека копию аргумента, после чего значение указателя стека R6 (в данный момент это и есть адрес размещения значения аргумента в ОЗУ стека) копируется в регистр R5, а в R3 записывается условный признак целого типа — число #377 (все адреса и числа здесь и далее восьмеричные). Затем (при передаче управления кодовой подпрограмме) в

вершину стека пересылается адрес возврата, и таким образом значение аргумента оказывается непосредственно «под вершиной», т. е. R5 будет равно R6+2 (напомним, что стек «растет сверху вниз» — от старших адресов к младшим).

Сам вызов USR-подпрограммы реализован не совсем привычным образом по сравнению с ассемблером. Адреса всех десяти USR-подпрограмм хранятся в специально отведенном массиве-списке по адресам 2100—2123. Изначально БЕЙСИК заносит во все ячейки этого списка одно и то же число #124300, являющееся адресом зашитой в ПЗУ команды процессора TRAP5, которая в БЕЙСИК-трансляторе служит для выдачи на экран сообщения об ошибке 5 (*в руководстве по БЕЙСИКу, прилагаемом к БК-0010.01, неправильно говорится об «ошибке 18»*). При использовании оператора DEF USR заданное в нем значение начального адреса заносится в данный список, заменяя собой бывшее там число #124300 (в ячейке, соответствующей указанному номеру USR-функции)*. При вызове USR-подпрограммы БЕЙСИК вначале записывает в вершину стека адрес перехода на специальную, прошитую в ПЗУ программу «возврата на шитый код» (команда MOV #163670,-(R6)), а затем извлекает из списка начальный адрес USR-подпрограммы и передает ей управление командой JMP (а не JSR !).

Теперь кодовая подпрограмма может извлечь из стека переданное ей значение аргумента (в рассматриваемом примере для целого типа — двухбайтное машинное слово, адрес которого указан в регистре R5) и обрабатывать его в соответствии с алгоритмом, заложенным в нее программистом. (Перед началом обработки можно проверить по значению R3 истинный тип переданного аргумента и, если он не соответствует требуемому, перейти сразу к выходу из подпрограммы или, например, передать управление на команду TRAP с нужным номером, чтобы, используя стандартный механизм БЕЙСИКа, вывести на экран сообщение об ошибке.) После этого можно вернуть в БЕЙСИК результат работы подпрограммы, для чего следует записать его значение на место копии аргумента (т. е. в ячейку стека, адрес которой указан в R5).

После выхода из кодовой подпрограммы по команде RTS PC управление будет передано на «возвратную» подпрограмму в ПЗУ БЕЙСИКа, адрес которой хранится в вершине стека (в кодовой подпрограмме нужно внимательно следить, чтобы перед выходом из нее значение R6 было тем же, что и после входа, а значение адреса в вершине стека не оказалось испорчено). При этом в вершине стека оказывается возвращаемый программой результат (или значение переданного на вход аргумента, если оно не изменялось), которое извлекается БЕЙСИКом и переписывается в переменную, стоящую в левой части равенства в операторе USR или выводится на экран при записи этого оператора в виде PRINT USR(<аргумент>) .

На упомянутый факт, а именно то, что БЕЙСИК считывает возвращаемый результат с вершины стека, нужно обратить особое внимание. Из него следует, что передаваемое в кодовую подпрограмму значение регистра R5 носит исключительно вспомогательный характер. Нельзя, например, вернуть в БЕЙСИК содержимое какой-либо произвольной ячейки ОЗУ, просто записав ее адрес перед возвратом из подпрограммы в R5. Напротив, для этого следует переписать содержимое этой ячейки в стек по указанному в R5 адресу. Аналогично, лишь вспомогательное значение имеет регистр R3: тип передаваемого аргумента (а значит, и возвращаемого результата) фиксируется в шитом коде БЕЙСИКа, так что невозможно, скажем, передать в подпрограмму целое число, а вернуть текстовую строку.

* Зная о роли указанных ячеек в вызове USR-подпрограмм, можно легко реализовать их «автоматическое» определение при подгрузке кодового модуля в БЕЙСИК оператором BLOAD "<имя>",R. В начале этого модуля нужно поместить машинные команды MOV, заносящие в требуемые ячейки адреса меток, соответствующих входам в подпрограммы, и завершить их ряд командой HALT. (Запись адреса в ячейку 2100 означает определение USR0, в ячейку 2102 — USR1 и т. д.) После загрузки с параметром R загрузчик запускается, производится «автоопределение» USR-функций, а по HALT — выход в диалоговый режим БЕЙСИКа. Теперь оператор DEF USR уже не требуется.

По тому же способу — копированием в стек — в подпрограмму передаются значения констант и переменных с плавающей запятой одинарной и двойной точности, но в этом случае в стек переписывается соответственно 4 или 8 байт, а в R3 — числа-признаки 1 или 0 (в R5 же, как и прежде, содержится значение R6+2). Следует обратить внимание на то, что двухбайтные машинные слова для многоразрядных чисел записываются в стек в обратном обычному порядке. Поясним сказанное на примерах:

```
USR(1!):  0
          40200       <--- R5=R6+2
          163670    <--- R6

USR(1#):  0
          0
          0
          40200       <--- R5=R6+2
          163670    <--- R6
```

Если же в качестве аргумента передается текстовая строка, то в стек заносится ее дескриптор:

```
<адрес начала строки в ОЗУ>
  <длина строки, байт> <--- R5=R6+2
          163670      <--- R6
```

В регистр-признак R3 при этом записывается число #177400.

Таким образом, мы видим, что в вильнюсском БЕЙСИКе передача в USR-подпрограмму числовых данных любого типа производится по значению, а текстовых строк — по ссылке (передача же массивов вообще не реализована). Поэтому при вызове USR-функций, принимающих в качестве аргументов текстовые константы и изменяющих их, нужно соблюдать известную осторожность, иначе может быть изменен исходный листинг БЕЙСИК-программы. Например, если мы пишем подпрограмму, которая выводит передаваемый ей текст в служебную строку, а затем производит его рулонное «прокручивание» влево, чтобы создать эффект «бегущей строки», то листинг БЕЙСИК-программы будет выглядеть так:

до обращения к USR:

100 A$=USR("Пример текста")

после первого срабатывания:

100 A$=USR("ример текстаП")

после второго срабатывания:

100 A$=USR("имер текстаПр")

и т. д.

Передача в USR-подпрограмму нескольких параметров

Крупным недостатком реализации USR-функций в вильнюсском БЕЙСИКе является то, что передать и вернуть из нее можно только один аргумент. А как быть, если подпрограмма требует нескольких?

Наиболее распространенные варианты решения подобной задачи — запись необходимых аргументов (всех, кроме одного, передаваемого через оператор USR) перед каждым вызовом подпрограммы в заранее отведенные ячейки памяти с помощью операторов POKE (по одному оператору на каждый аргумент) или кодирование нескольких аргументов в виде текстовой строки. Примеры первого способа можно найти практически в любой БЕЙСИК-программе, работающей со спрайтами, а второго — в реализациях оператора PLAY, где массив частот и длительностей записывается в виде строки символов.

Имеется, впрочем, и более совершенный способ, предложенный Ю. В. Котовым и описанный в журнале «Вычислительная техника и ее применение», №3 за 1992 г. При записи строки БЕЙСИК-программы в виде A%=X1%=(X2%=(...=USR(I%))...), где A% — некоторая неиспользуемая для других целей буферная переменная, БЕЙСИК рассматривает подобный оператор как логический и перед вызовом USR-подпрограммы заносит все перечисленные значения в стек, откуда кодовая подпрограмма может их извлечь. Стек при этом имеет вид:

```
<X1%>
<X2%>
.
.
.
<Xn%>
<I%>          <--- R5=R6+2
163670     <--- R6
```

(Значение I% можно использовать для указания количества передаваемых параметров.)

Таким образом, мы можем передать в подпрограмму сразу несколько аргументов, хотя запись при этом становится немного громоздкой. Однако «аппетит приходит во время еды», и весьма хотелось бы большего — передавать в подпрограмму сразу массивы по их именам как данные ссылочного типа.

Новые возможности: свободное оперирование данными

После довольно длительных исследований удалось выяснить, что в вильнюсском БЕЙСИКе имеются неизвестные ранее возможности для реализации в полном объеме передачи данных ссылочного типа, т. е. существует несложный способ, позволяющий не только свободно передавать в качестве аргумента USR-функций массив (как одномерный, так и многомерный) путем указания в качестве аргумента их имен, но и получить ряд других интересных эффектов (например, реализация имеющегося в БЕЙСИКе БК-0011, но отсутствующего в БК-0010.01 оператора VARARG — определение адреса размещения в ОЗУ значения переменной с заданным именем).

Выяснилось, что в «шитом коде», на который после входа в USR-подпрограмму указывает значение регистра R4, содержатся все необходимые сведения. Рассмотрим ряд конкретных примеров.

Аргумент — текстовая константа (USR("ПРИМЕР"))

Ячейка ОЗУ с адресом (R4-12$_8$) содержит адрес начала текста, т. е. фактически указывает на содержащую заданный текст область исходного листинга БЕЙСИК-программы. Если же вызов произведен в непосредственном режиме (во время диалога с пользователем, а не из программы), то данный адрес указывает на ячейки буфера строкового редактора БЕЙСИКа (адреса 2422—3021). Ячейка же с адресом (R4-10$_8$) содержит длину заданного текста в байтах.

Аргумент — текстовая переменная (USR (A$))

Ячейка с адресами (R4-10$_8$) содержит адрес дескриптора текстовой переменной, т. е. если обозначить содержимое этой ячейки как A1, то по адресу (A1+2) в ОЗУ размещается начальный адрес значения текстовой переменной, а по адресу A1 — его длина в байтах.

Аргумент — числовая константа любого типа (USR(1%))

Заданная константа содержится в ОЗУ в ячейках с адресами:

- (R4-10$_8$) — для целых констант (%, &O, &H и &B);
- (R4-10$_8$)—(R4-16$_8$) — для констант двойной точности.

Аргумент — числовая переменная любого типа (USR(A))

Ячейка с адресом (R4-10$_8$) содержит адрес местонахождения в ОЗУ значения переменной, которое может занимать 2, 4 или 8 байт в зависимости от типа. (Последнее можно использовать для реализации оператора VARARG, см. ниже.)

Аргумент — элемент массива любого типа с любым индексом

Ячейка с адресом (R4-12$_8$) содержит размерность массива (1 — для одномерного, 2 — для двумерного и т. д.), а с адресом (R4-14$_8$) — адрес дескриптора массива.

Одномерный массив. Если обозначить содержимое ячейки с адресом (R4-14$_8$) как A1, а содержимое по адресу A1 как A2, то:

- A2 — адрес ячейки, содержащей количество элементов в массиве (максимальная величина значения индекса, заданная в операторе DIM, плюс 1);
- (A2+2) — адрес первого элемента массива (с индексом 0).

Все элементы массива хранятся начиная с данного адреса в порядке следования индексов и занимают 2, 4 или 8 байт, в зависимости от типа. Таким образом, значение (A2+2) является указателем на массив. (Отметим, что оно не зависит от значения индекса элемента, указанного в операторе USR, т. е. смысл в данном случае имеет только само имя массива.)

Дополнительно имеется ячейка с адресом (R4-20$_8$), содержащая значение индекса элемента массива (только для целого значения индекса!), которое было указано в операторе USR (т. е. мы имеем доступ ко всему массиву, зная его начальный адрес, а также можем определить, какой конкретно его элемент был передан в подпрограмму).

Многомерный массив. Для БЕЙСИКа, который правильно работает только с одно- и двумерными массивами, имеет смысл подробно рассмотреть здесь только двумерный массив, однако для трехмерных, четырехмерных и т. д. массивов все будет выглядеть аналогично.

Для двумерного массива (матрицы), если обозначить содержимое ячейки с адресом (R4-14$_8$) как A1, а содержимое по адресу A1 как A2, то:

- A2 — адрес ячейки, содержащей количество столбцов матрицы (второй индекс в операторе DIM + 1);
- (A2+2) — адрес ячейки, содержащей количество строк (первый индекс в DIM + 1);
- (A2+4) — адрес первого элемента массива (с нулевыми индексами).

Значения элементов массива записываются построчно начиная с элемента (0,0): A(0,0), A(0,1),...A(1,0), A(1,1),... и занимают в зависимости от типа по 2, 4 или 8 байт.

Например, для массива, определенного оператором DIM X%(2,4) и содержащего значения:

```
 0  1  2  3  4
10 11 12 13 14
20 21 22 23 24
```

ячейки будут содержать:

- по адресу A2: 5 (4+1);
- по адресу (A2+2): 3 (2+1);
- по адресу (A2+4) и далее: 0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24 (в восьмеричном представлении: 0, 1, 2, 3, 4, 12, 13, 14, 15, 16 и т. д.).

Кроме того, ячейки с адресами (R4-20$_8$) и (R4-24$_8$) для двумерного массива содержат значения правого (второго) и левого (первого) индексов конкретного элемента соответственно.

Следует заметить, что все сказанное выше относится к числовым массивам. Для массивов строковых переменных все обстоит несколько сложнее: в текстовом массиве элементами будут являться не сами строки, а только ссылки на них.

Примеры использования нового метода

Реализация оператора VARARG

Следующая короткая подпрограмма в машинных кодах возвращает адрес размещения в ОЗУ числовой переменной, имя которой задано в качестве аргумента USR.

```
010415 162725 000010
015515 000207
```

Пример организации доступа к массиву

Программа, очищающая одномерный целочисленный массив, обнуляя его элементы.

```
010400 162700 000014
011000 011000 012001
005020 077102 000207
```

Вильнюсский БЕЙСИК, шитые коды и работа с принтером

В процессе исследования шитых кодов для различных операторов вильнюсского БЕЙСИКа БК-0010.01 (методика и вспомогательная программа были приведены в статьях В. В. Авсеева и А. В. Авсеева об использовании прошитых в ПЗУ подпрограмм реализации операторов БЕЙСИКа в журналах «Информатика и образование», №2 за 1990 г., и «Вычислительная техника и ее применение», №12 за 1990 г.) удалось выяснить интересную особенность реализации вывода на экран и на принтер. Оказалось, что отдельных подпрограмм для реализации операторов LPRINT и LLIST в трансляторе просто не существует! Вместо этого используются те же подпрограммы, что и для операторов PRINT и LIST, а «получателя» информации (экран или принтер) определяет содержимое служебной ячейки с адресом 2416 (здесь и далее все адреса и числа восьмеричные). Содержимое этой ячейки устанавливается с помощью отдельных подпрограмм БЕЙСИК-транслятора с адресами 162640 («готовить БК к выводу на принтер») и 162650 («закончить работу с принтером»), которые как бы окантовывают шитые коды оператора PRINT. (С LIST и LLIST все обстоит аналогично, но протестировать их шитые коды по обычной методике невозможно, так как эти операторы допустимы только в режиме непосредственного выполнения.) Конкретно, для печати на принтере в ячейке @#2416 должно быть записано число -1 (&O177777), а для вывода на экран — 0.

Эта особенность дает пользователям БК-0010.01 сразу две новые возможности. Во-первых, в программах на ассемблере можно использовать шитый код оператора PRINT (как это описано в статьях Авсеевых) для вывода не только на экран, но и на принтер. Шитый код при этом будет выглядеть так:

```
162640
<шитый код для оператора PRINT>
162650
```

Во-вторых, так как запись -1 или 0 в ячейку @#2416 можно производить просто оператором POKE, появляется удобная возможность переключения: вывод данных на экран (операторы PRINT) или печать на принтере (как вручную пользователем в непосредственном режиме, так и программно). Это, в свою очередь, позволяет обеспечить работу с принтером в любой программе, использующей операторы PRINT, без их «глобальной» замены на LPRINT (что особенно трудоемко, если вместо слова «PRINT» везде использован вопросительный знак). Достаточно лишь перед запуском программы ввести с клавиатуры команду POKE &O2416,-1 , и весь поток данных «переадресуется» на принтер (как и листинг, выводимый по LIST, причем в последнем случае после срабатывания ячейка @#2416 автоматически обнуляется). Текст подсказок в операторах INPUT также выводится на печать, но попытка считать с принтера значения переменных приводит к останову с выдачей сообщения об ошибке 59 после первого же оператора INPUT. После окончания работы программы для возврата к обычному режиму вывода на экран нужно использовать команду POKE &O2416,0.

И наконец, данная особенность позволяет создавать компактные программы на БЕЙСИКе, позволяющие по желанию пользователя выводить данные на экран или на принтер. «Стандартным» способом это делается, например, так:

```
      . . . . .  . ' предыдущие операторы
100 INPUT  "1 - принтер, 0 - экран:";OT%
110 IF OT%=0 GOTO 220
 120 LPRINT A  ' блок вывода
130 LPRINT B   '
      . . . .  . . ' данных на принтер
200 LPRINT Z   '
210 GOTO 310
```

```
220 ? A       ' блок вывода
230 ? B       '
 ......       ' данных на экран
300 ? Z       '
310 .....     ' дальнейший текст программы
```

Используя описанную выше особенность, можно запрограммировать то же самое без использования ветвления:

```
 ......
 100 INPUT "0 - экран, 1 - принтер:";OT%
110 POKE &O2416,-OT%
120 ? A       ' блок вывода данных
130 ? B       '
 ......       ' на экран или на принтер
200 ? Z       '
210  POKE &O2416,0%
220 .....     ' дальнейший текст программы
```

Выигрыш в объеме листинга очевиден. (Кстати, так как шитый код оператора LPRINT уже содержит в себе обращения к подпрограммам, меняющим содержимое ячейки @#2416, «превратить» аналогичным образом LPRINT в PRINT нельзя.)

В заключение следует сделать еще одно замечание. Подпрограмма реализации оператора PRINT (она же — для LPRINT) не производит операцию перевода строки, т. е. указанный в статье Авсеевых (журнал «Вычислительная техника и ее применение») для оператора PRINT A% шитый код [A%]157070 на самом деле соответствует оператору PRINT A%;. Для выполнения перевода строки как на экране, так и в принтере в шитый код следует добавить адрес 156770. Ниже даны примеры полной записи шитого кода:

```
LPRINT A%
          162640  ; запись -1 в ячейку @#2416 (вывод на принтер)
           156232 ; запись в стек числа, расположенного по
                  ; адресу, указанному в следующем слове
          <адрес> ; адрес ячейки ОЗУ, содержащей значение переменной A%
          157070  ; вывод на экран (или на принтер)
          156770  ; перевод строки
          162650  ; запись 0 в ячейку @#2416

LPRINT   ' отдельный перевод строки на принтере
          162640  ; запись -1 в ячейку @#2416
          156770  ; перевод строки
          162650  ; запись 0 в ячейку @#2416

PRINT A$
          156214  ; пересылка в стек значений адреса и длины
                  ; содержимого A$ из двух машинных слов,
                  ; расположенных по адресам <адрес> и
          <адрес> ; <адрес+2>
          156460  ; вывод текстовой строки
          156770  ; перевод строки

PRINT "<текст>";
          156156   ; запись двух последующих слов в стек
          <адрес>  ; адрес начала строки текста
          <длина>  ; и ее длина в байтах
          156460   ; вывод текстовой строки
```

В. В. Ермаков,
Магаданская обл.

Многомерные массивы на БЕЙСИКе*

Одним из недостатков вильнюсской версии БЕЙСИКа БК-0010.01 является невозможность использования в программах многомерных массивов (трехмерных, четырехмерных и т. д.). Причина этого — ошибка разработчиков транслятора, суть которой состоит в следующем.

Известно, что адрес расположения элемента массива в ОЗУ можно найти по формуле:

$$A=A0+i_1+i_2 \cdot N_1+i_3 \cdot N_1 \cdot N_2+\ldots+i_n \cdot N_1 \cdot N_2 \cdot N_3 \cdot \ldots \cdot N_{n-1},$$

где **A0** — адрес начала массива в ОЗУ, i_1—i_n — значения индексов искомого элемента, а N_1—N_n — максимальные значения индексов, указанные в операторе DIM. В существующей же версии реализована другая формула:

$$A=A0+i_1+i_2 \cdot N_1+i_3 \cdot N_2+\ldots+i_n \cdot N_{n-1},$$

которая, как легко видеть, правильно работает только с одно- и двухмерными массивами.

Исправить ошибку можно с помощью небольшой подпрограммы в машинных кодах, размещаемой в стеке с адреса &O400 по &O460. Ее коды записаны в операторе DATA приведенной ниже «установочной» БЕЙСИК-программы.

```
10 DATA &O13400,&O12403,&O12601,&O5720,&O5303,&O3421, &O10305,&O166405,
&O177776,&O5405,&O12602,&O10046,&O10146, &O14001,&O10546,&O4737,&O160442,
&O10102,&O12605,&O77507, &O62601,&O12600,&O754, &O137,&O160172
20 FOR I%=&O400 TO &O460 ST 2%
30 READ J%
40 S=S+J%
50 POKE I%,J%
60 NEXT
70 IF S<>111935 THEN ?STRING$(20,7);"Ошибка,проверьте данные в строке 10"
80 ? "Загрузка произведена"
90 END
```

После запуска на выполнение производится загрузка кодов в область стека с одновременным контролем правильности их ввода. Если не выдано сообщения об ошибке (строка 70), можно после останова записать кодовую подпрограмму командой **BSAVE"<имя>",&O400,&O460** и позже подгружать к программам, работающим с многомерными массивами, командой **BLOAD**. (Можно также, если объем свободной памяти достаточен, включать вышеприведенный «загрузочный» фрагмент непосредственно в свои БЕЙСИК-программы.)

В программу же на БЕЙСИКЕ, использующую многомерные массивы, нужно записать (до первого обращения к такому массиву) небольшой фрагмент из трех строк:

```
FOR I%=PEEK(&O2002) TO PEEK(&O2004) ST 2%
IF PEEK(I%)=-8146 THEN POKE I%,&O400
NEXT
```

Данный цикл организует просмотр сгенерированного при компиляции программы шитого кода (начало и конец массива шитых кодов берется из служебных ячеек @#2002 и @#2004), обнаружение в нем адреса вызова стандартной подпрограммы обработки массивов, прошитой в ПЗУ БЕЙСИКа (шитый код 160056 или, в десятичном представлении, −8146) и замену его на адрес начала внешней подпрограммы, размещенной в стеке (&O400). Теперь при обращении к элементам массивов управление будет передаваться не на ошибочную подпрограмму в ПЗУ, а на ее исправленный вариант, что дает возможность корректно работать с массивами любой размерности так, как это описывается в прилагаемом к БК руководстве по БЕЙСИКу.

Примечание. В целях экономии занимаемого места в ОЗУ в исправленной версии подпрограммы обработки массивов не реализована диагностика ошибок. Поэтому рекомендуется вначале отладить программу (невзирая на неверное содержимое элементов многомерных массивов), а уже потом «подключить» к ней исправленную подпрограмму.

---

* Публикуется по материалам журнала «Вычислительная техника и ее применение», №8 за 1990 г.

Справочный листок

Вильнюсский БЕЙСИК БК-0010.01 (версия 1980.07.24). Шитые коды

Операторы

KEY A%,B$	(A%, адрес B$, длина B$) 124570
PRINT A%	(A%) 157070, 156770
PRINT A!	(A!) 157014, 156770
PRINT A#	(A#) 157022, 156770
PRINT A$	(адрес A$, длина A$) 156460, 156770
(Примечание. 156770 — шитый код подпрограммы перехода на следующую строку.)	
INPUT X1,...,Xn	160634, 157150, адрес следующей команды, список адресов X1,...,Xn
OPEN A$ FOR OUTPUT	(адрес A$, длина A$) 162336
OPEN A$ FOR INPUT	(адрес A$, длина A$) 163360
CLOSE	162656
POKE A%,B%	(A%, B%) 160536
OUT A%,B%,C%	(A%, B%, C%) 160542
CLS	156776
COLOR A%,B%	(A%, B%) 125172, 125242
LOCATE A%,B%,C%	(A%, B%) 124512, (C%) 124324
PSET (A%,B%),C%	(A%, B%, C%) 125242, 125410, 125220
LINE (A%,B%)-(C%,D%),E%	(A%, B%, C%, D%, E%) 125242, 125426, 125220
LINE (A%,B%)-(C%,D%),E%,B	(A%, B%, C%, D%, E%) 125242, 125454, 125220
CIRCLE (A%,B%),C%,D%	(A%, B%, C%, D%) 125334, 126724, 126742, 127122, 127334
PAINT (A%,B%),C%,D%	(A%, B%, C%) 125334, (D%) 125334, 125642
DRAW A$	(адрес A$, длина A$) 131412
BEEP	156762
TRON	154626
TROFF	154674
ON A% GOTO m1,...,mn	(A%) 155564, n, адреса m1,...,mn
STOP	154754
END	154722
GOTO N	155560, адрес N
GOSUB N	155526, адрес N
RETURN	155722
FOR I%=A% TO B% STEP C%	(A%) 156350, адрес I%, (B%, C%) 156426, адрес следующего оператора, адрес I%
FOR I#=A# TO B# STEP C#	(A#) 156334, адрес I#, (B#, C#) 156404, адрес следующего оператора, адрес I#
NEXT I%	156466, адрес I%, 156572
NEXT I#	156466, адрес I#, 156636, 155176, 155466
IF...THEN...ELSE	сравнение, 155404, адрес ветви ELSE, ветвь THEN, 155560, адрес следующего за ELSE оператора, ветвь ELSE

Отношения и логические операции

A%<B%	(A%, B%) 155172, 155452
A#<B#	(A#, B#) 177034, 155452
A%>B%	(A%, B%) 155172, 155456
A#>B#	(A#, B#) 177034, 155456
A%>=B%	(A%, B%) 155172, 155446
A#>=B#	(A#, B#) 177034, 155446
A%<=B%	(A%, B%) 155172, 155442
A#<=B#	(A#, B#) 177034, 155442
A%<>B%	(A%, B%) 155172, 155436
A#<>B#	(A#, B#) 177034, 155436
A%=B%	(A%, B%) 155172, 155430
A#=B#	(A#, B#) 177034, 155430
NOT A%	(A%) 162306
A% AND B%	(A%, B%) 162316
A% OR B%	(A%, B%) 162312
A% XOR B%	(A%, B%) 162324
A% EQV B%	(A%, B%) 162340
A% IMP B%	(A%, B%) 162344

Команды БЕЙСИКА

CLOAD	136650
BLOAD	136420
FIND	136306
CSAVE	137124
BSAVE	136562
LOAD	136104
SAVE	137256
NEW	134600
AUTO	140376
LIST	134634
MONIT	124302
LLIST	134626
RENUM	141034
DELETE	140514
CONT	136066
RUN	135242

Функции

SQR(X#)	(X#) 171350
SIN(X#)	(X#) 173614
COS(X#)	(X#) 173566
TAN(X#)	(X#) 174306
ATN(X#)	(X#) 174434
EXP(X#)	(X#) 171762
PI	167102
LOG(X#)	(X#) 173052
ABS(X#)	(X#) 166566
ABS(X%)	(X%) 166556
FIX(X#)	(X#) 176212
INT(X#)	(X#) 176340
SGN(X%)	(X%) 166572
SGN(X!)	(X!) 166624
SGN(X#)	(X#) 166614
RND(X#)	(X#) 175176
FRE(X%)	(X%) 160600
FRE(A$)	(адрес A$, длина A$) 160620
CINT(A#)	(A#) 160760
CSNG(A#)	(A#) 162020
CDBL(A!) (A!)	166742
CDBL(A%)	(A%) 166646
PEEK(A%)	(A%) 160556
INP(X%,Y%)	(X%, Y%) 160564
ASC(A$)	(адрес A$, длина A$) 161534
CHR$(A%)	(A%) 160510
LEN(A$)	(адрес A$, длина A$) 161530
MID$(A$,B%,C%)	(адрес A$, длина A$, B%, C%) 161340
STRING$(X%,A$)	(X%) 160634, (адрес A$, длина A$), 61404
VAL(A$)	(адрес A$, длина A$) 161550
INKEY$	124132
STR$(A%)	(A%) 161610 (JSR PC,@#140536)
STR$(A#)	(A#)161602 (JSR PC,@#164710)
STR$(A!)	(A!)161574 (JSR PC,@#164664)
OCT$(A%)	(A%) 161624
HEX$(A%)	(A%) 161734
BIN$(A%)	(A%) 161672
CSRLIN(A%)	(A%) 124560
POS(A%)	(A%) 124544
LPOS(A%)	(A%) 157054
EOF	163040
AT(A%,B%)	(A%, B%) 124512
TAB(A%)	(A%) 124420
POINT(A%,B%)	(A%, B%) 125602

Арифметические операции

A%+B%	(A%, B%) 162102	A#/B#	(A#, B#) 170776
A#+B#	(A#, B#) 167144	A%\B%	(A%, B%) 162052
A%-B%	(A%, B%) 162112	A%*B%	(A%, B%) 162116
A#-B#	(A#, B#) 167124	A#*B#	(A#, B#) 170210

Вспомогательные программы

156160, A%	— запись слова A% в стек
156156, AI	— запись двух слов в стек
156152, A#	— запись четырех слов в стек
156232, A%	— пересылка слова по адресу A% в стек
156214, A%	— пересылка двух слов по адресу A% в стек
156170, A%	— пересылка четырех слов по адресу A% в стек
156350, A%	— запись слова из стека по адресу A%
156330, A%	— запись двух слов из стека по адресу A%
156334, A%	— запись четырех слов из стека по адресу A%
124104	— вызов пользовательского ПЗУ (CALL)
156770	— перевод строки на экране или в принтере
124324	— включение курсора
166562	— изменение знака целого числа в стеке
156750	— изменение знака вещественного числа в стеке
125242	— установка цвета (значение цвета находится в стеке)
125334	— формирование кода цвета по его номеру

Некоторые системные ячейки БЕЙСИКа

1000—2000	— стек БЕЙСИКа
2000—3052	— служебная область БЕЙСИКа
2002	— содержит адрес последнего байта исходного листинга (со следующего четного адреса начинается шитый код)
2004	— содержит адрес конца шитого кода
2006	— количество зарезервированных байтов (CLEAR)
2050	— флаг трассировки (1 — включена, 0 — выключена)
2422	— буфер строкового редактора для ввода с клавиатуры (256 байт)
3022	— буфер преобразования в строку и вывода чисел на экран
37400	— буфер ввода-вывода с магнитофона (или локальной сети)
120300	— адрес перезапуска БЕЙСИКа из монитора с сохранением программы
121052	— вызов строкового редактора текстов БЕЙСИКа (команда JSR PC,@#121052). Текст вводится в буфер с начальным адресом @#2422, длина буфера — 256 байт
146404	— адрес подпрограммы обработки прерывания TRAP
120234	— адрес подпрограммы обработки прерывания по клавише «СТОП»
120540	— адрес подпрограммы обработки прерывания по вектору 10

Примечания

1. Буквами A, B, C и др. обозначены числовые и текстовые константы и адреса. Их типы соответствуют стандарту БЕЙСИК-MSX: # — вещественные двойной точности (четыре машинных слова или восемь байт), ! — вещественные одинарной точности (два слова), % — целые (одно слово).

2. Запись параметров в скобках означает, что перед вызовом последующего шитого кода их необходимо переписать в стек в порядке перечисления. Для этого следует использовать соответствующие вспомогательные подпрограммы.

3. Для вызова последовательности шитых кодов необходимо записать начальный адрес их списка в регистр R4 и передать на них управление командой JMP @(R4)+ .

4. При использовании операторов ввода-вывода нужно обязательно обнулить ячейку @#2416, а перед вызовом арифметических операций — также ячейку @#2314.

5. В БЕЙСИКе нет арифметических операций над вещественными числами одинарной точности, для этого нужно выполнить преобразование CDBL(A!), а затем — арифметическую операцию для полученного числа двойной точности. Аналогичным образом можно поступить, если в списке отсутствует операция над числами нужного типа (например, для выполнения операции POS(A#) нужно использовать A%=INT(A#) и POS(A%)).

6. Для самостоятельного исследования шитых кодов можно использовать следующую программу на БЕЙСИКе. Исследуемые операторы располагаются в строках 50—100, а запуск программы на исполнение производится командой RUN 100. Программа выводит шитые коды вплоть до расположенного в строке 100 оператора CLS (соответственно, среди исследуемых операторов в строках 50—100 CLS быть не должно). Вывод кодов производится постранично, продолжение — после нажатия любой клавиши.

```
10   ' Программа запускается
20   ' командой RUN 100.
30   ' Ниже должны находиться
40   ' исследуемые операторы:
50            . . .
100  CLS
110  N%=PEEK(&O2002)
120  IF N% MOD 2% = 1% THEN
     N%=N%+1%
130  CLS
140  ? " Адрес Шитый код"
150  ? "------------------"
160  S%=0
170  IF PEEK(N%)=&O156776 GOTO 230
180  ? " ";OCT$(N%);" -->";
     OCT$(PEEK(N%))
190  N%=N%+2%
200  S%=S%+1%
210  IF S%<20% GOTO 170
220  IF INKEY$="" THEN 220 ELSE 130
230  ?
240  ? "Все шитые коды выведены"
250  END
```

7. Шитый код, соответствующий реальной программе на БЕЙСИКе, включает в себя комбинацию указанных выше адресов обрабатывающих и сервисных подпрограмм, а также данных (констант и адресов переменных). Пример использования шитого кода в программе на ассемблере стандарта МИКРО-10:

Программа-прототип на БЕЙСИКе

```
10   CLS
20   LINE (0%,0%)-(&O62,&O62),,B
30   PAINT (&O24,&O24)
40   GOTO 10
```

Реализация на ассемблере

```
        MOV #146404,@#34
STRT:   MOV #PRG,R4
        JMP @(R4)+
PRG:    .#156776.#156160.#0.#156160.#0
        .#156160.#62.#156160.#62
        .#125454.#156160.#24.#156160
        .#24.#125376.#125404.#125642
        .#155560.#STRT
        END
```

Публикуется по материалам:

Авсеев В. В., Авсеев А. В. Использование возможностей БЕЙСИКа в программах на языке ассемблера. //Вычислительная техника и ее применение. 1990. №12. С.24.

Авсеев В., Авсеев А. Особенности транслятора с языка БЕЙСИК для БК-0010.01. //Информатика и образование. 1990. №2. С.42.

*** Внимание! Опечатка ***

По вине автора в статье С. К. Румянцева «Принтер–художник» (№1 за 1995 г., стр.70) в строке 1214 листинга программы допущена ошибка.

Следует читать:

```
001214 000204      ; части средней строки
```

* * *

В листингах игр, опубликованных в 1994 г. под рубрикой «Потехе час», допущены следующие опечатки.

• игра «STEP»

в строке 890 пропущена запятая. Следует читать:

```
890 DATA &O4000,&O6000,&O10000,&O12000,&O14000,&O16000, &O20000,&O22000
```

• игра «Потехе час»

— в строке 1350 пропущены знаки «<» и «>». Следует читать:

```
1350 IF B$=" "TH1380ELIF B$<>""TH1040
```

— в строках 1930 и 2030 вместо знаков "«" и "»" должны быть обычные кавычки.

• игра «Сапер»

— в строках 1010 и 1020 пропущен знак «>». Следует читать:

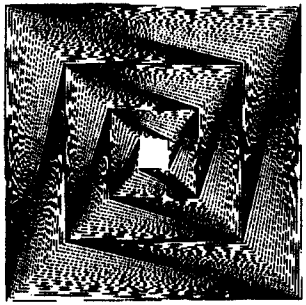
```
1010 IF SUM%>2 TH C%=-21846%
1020 IF SUM%>4 TH C%=-1%
```

— в строке 1450 пропущен пробел:

```
1450 ?AT(I%+6%,J%+1%);CHR$(156%);" ";CHR$(156%);
```

Приносим читателям свои извинения.

Следует также учесть, что все публикуемые в журнале БЕЙСИК-программы рассчитаны на работу в обычном («магнитофонном») варианте вильнюсского БЕЙСИКа БК-0010.01 (если иное не оговорено особо). Их работоспособность в «дисковом» БЕЙСИКе или в других версиях не гарантируется.



Пользователи БК-0010(.01) чаще всего жалуются на низкое быстродействие и малый объем оперативной памяти. При программировании на БЕЙСИКе второй недостаток особенно неприятен (благо большая скорость вычислений в БЕЙСИК-программах требуется не всегда). Но расширение ОЗУ — вещь достаточно сложная и связанная с необходимостью аппаратной доработки. Есть, однако, и другой вариант: с помощью разного рода программных ухищрений постараться втиснуть в имеющийся объем ОЗУ как можно больше полезной информации, размещая в памяти программы и листинги наиболее рациональным образом.

В этой статье описывается один из возможных вариантов, основанный на использовании генерируемого БЕЙСИК-транслятором объектного («шитого») кода без исходного листинга.

Ю. В. Котов,

Москва

Работа на ПЭВМ БК-0010 с программами увеличенного объема

Основным недостатком малых ПЭВМ типа БК-0010 является небольшой объем оперативной памяти, что дополнительно усугубляется неэкономностью транслятора БЕЙСИКа, требующего хранения в ОЗУ, помимо исходного текста программы, еще и объемистого промежуточного (объектного) кода, который для программы средней сложности занимает примерно столько же места, сколько исходный текст вместе с таблицей адресов операторов. В прошлом, когда ограниченный объем памяти имели все ЭВМ, даже профессиональные, выходом из подобной ситуации была поэтапная обработка программы с разделением на трансляцию, редактирование связей (компоновку) и исполнение сгенерированной машинной программы, причем на последнем этапе ни исходный текст, ни транслятор с компоновщиком уже не требуются, т. е. ресурсы ЭВМ используются максимально эффективно. Современные БЕЙСИК-системы для IBM-совместимых ПЭВМ (Turbo-Basic, Quick-Basic и др.) также предусматривают, кроме обычного режима выполнения программы в среде турбо-оболочки, возможность генерации EXE-программ, которые затем могут многократно запускаться без загрузки в память БЕЙСИК-системы и без повторных трансляций. Разработчики же вильнюсского БЕЙСИКа для БК-0010 создали систему, фактически предназначенную для отладки и немногократного выполнения программ ограниченной сложности, «забыв» о не менее необходимом режиме многократного «пользовательского» выполнения готовой, уже отлаженной программы. Это тем более досадно, что такой режим можно было бы обеспечить, практически не усложняя существующую систему.

Анализ прошивки ПЗУ БК и эксперименты показали, что с учетом некоторых условий и ограничений объектный код может работать без наличия в памяти исходного текста программы и таблицы адресов. Далее мы рассмотрим этот способ, а также комплексный (сочетание выполняемой в обычном «штатном» режиме БЕЙСИК-программы с подпрограммой (подпрограммами) в виде заранее полученного объектного кода). В целом объем (а значит, и сложность) обрабатываемой при этом в ПЭВМ программы может возрасти раза в полтора.

Другая возможность работы с программами увеличенного объема — частичное использование экранной памяти. К сожалению, стандартный режим РП оставляет слишком мало места для изображения на экране, что неприемлемо для многих программ, особенно использующих графику. Ниже будет показано, как можно передать программе около

четверти или трети экранной памяти, так что на экране останется еще достаточно места для вывода текста и графики.

Автономная работа объектного кода

Сразу заметим, что объектный код в БК-0010 неперемещаем и содержит в себе абсолютные адреса точек перехода и переменных. Это затрудняет его автономное использование, так как приходится загружать его в то же место памяти, где он был сформирован.

Генерируемый объектный код размещается после исходного текста программы, адрес его начала содержится в ячейке 2002 (все адреса здесь и далее восьмеричные), адрес окончания — в ячейке 2004. Затем идет область текстовых переменных, размер которой может быть задан (уменьшен) оператором CLEAR. Далее следует область (буфер) для организации циклов и размещения содержимого массивов, потом — свободное место, величину которого можно определить с помощью оператора FREE (после отработки программы, когда все рабочие области уже заполнены). Область переменных и описателей массивов начинается с адреса, указанного в ячейке 2022, и доходит до «вершины» таблицы адресов* (это значение указано в ячейке 2024). «Низ» этой таблицы соседствует с началом области ввода-вывода (адрес в ячейке 2026), длина которой 1022 байта, а конец этой области соответствует максимальному адресу используемой оперативной памяти (MAXAD). (Эта область используется не всегда, тем более в режиме автономной работы объектного кода. Но надо иметь в виду, что ячейка с адресом **MAXAD-422** должна быть обнулена.) Значение MAXAD содержится в ячейке 2030, по умолчанию это начало экранной памяти (40000), изменить его можно с помощью оператора CLEAR (второй параметр), соответственно смещается таблица адресов и область переменных.

Автономный объектный код программы (нескольких программ или программы с подпрограммами) получается в ходе трансляции исходного текста оператором RUN. После этого, не изменяя текст программы (чтобы не «сбросились» рабочие переменные), нужно записать объектный код на магнитную ленту командой **BSAVE "<имя>",<а.н.>,<а.к.>**, где **<а.н.>** — адрес начала объектного кода из ячейки 2002, **<а.к.>** — адрес его конца из ячейки 2004. (Содержимое рабочих ячеек удобно просматривать, запрограммировав один из ключей командой **KEY 1,"?OCT$(PEEK(&O"** .)

Для запуска автономного объектного кода нужно выполнить следующие действия:

- установить верхний предел памяти, используемой БЕЙСИКом, с помощью оператора **CLEAR <t>,<адр>**, где **<t>** — величина области для текстовых переменных (для работы «ведущей» программы на БЕЙСИКе), **<адр>** — число, не превышающее адреса начала автономного объектного кода. (Если область ввода-вывода не будет использоваться, этот адрес может быть увеличен на 400—420 байт, но ячейка с адресом **<адр>-422** должна быть обнулена.);
- считать автономный объектный код командой **BLOAD "<имя>"** ;
- ввести (либо считать с магнитной ленты или диска) «ведущую» программу на БЕЙСИКе, из которой будет запускаться автономный код. (Для этого она должна содержать операторы вида:

```
DEFUSR=&O<adr>
I% = USR(0) ,
```

где **<adr>** — адрес начала «стыковочной» подпрограммы, организующей переход к запуску объектного кода);

* Таблица адресов, как и стек БК-0010, «растет сверху вниз», т. е. ее «низ» соответствует большему адресу по сравнению с вершиной. — *Прим. ред.*

- ввести с помощью операторов POKE или считать с магнитной ленты «стыковочную» подпрограмму в машинных кодах. Она может располагаться в неиспользуемой части области ввода-вывода или за пределами области памяти, используемой БЕЙСИКом (ее можно также добавить к автономному объектному коду и считывать вместе с ним либо формировать в ведущей программе на БЕЙСИКе). Простейший вариант стыковочной подпрограммы может быть таким:

<adr>: 12704 <ad.cod> 134 ,

где **<ad.cod>** — начало объектного кода;

- запустить ведущую программу на БЕЙСИКе оператором RUN. Исполнение автономного кода заканчивается на операторе END или STOP с возвратом в среду БЕЙСИКа.

К сказанному выше следует добавить ряд замечаний:

- некоторые операторы (псевдокоманды) объектного кода, например INPUT, требуют наличия имен в области хранения переменных (для автономного кода это старая область). В таком случае вместе с объектным кодом на ленту следует записывать и область переменных до таблицы адресов, т. е. до адреса, содержащегося в ячейке 2024. Но тогда на ленту будет записана и лишняя информация, в том числе свободная зона памяти. Чтобы избежать этого, можно отдельно записать объектный код и область переменных;
- при первом после чтения с ленты запуске автономного кода переменные не инициализируются. Если область переменных была считана с ленты вместе с объектным кодом, переменные имеют значения, сформированные при запуске программы в момент генерации объектного кода. При последующих же его запусках сохраняются прежние значения переменных (полученные при предыдущей его работе);
- по указанной в предыдущем замечании причине объектный код с оператором DIM нельзя отрабатывать несколько раз. Если же такой оператор принципиально необходим (массивы имеют длину более 11 элементов), надо предусмотреть возможность обхода этого оператора при повторных запусках (с помощью оператора INPUT и условного перехода, путем входа в объектный код с другой точки и т. п.).

Другая возможность обхода повторного срабатывания операторов DIM состоит в следующем. Эти операторы размещаются с дополнительной точки входа (где-то со старшими номерами строк), с которой при генерации объектного кода и запускается программа. Затем объектный код записывается на ленту вместе с областью переменных, а значит, и с инициализированными описателями массивов. В автономном же режиме код будет запускаться с основного входа без вызова операторов DIM;

- операторы INPUT в объектном коде могут неправильно читать данные в массивы, тогда можно загружать данные в простые переменные и затем передавать в массивы;
- перед чтением по READ нужно обязательно производить инициализацию с помощью RESTORE;
- в автономном коде нельзя использовать текстовые константы, в том числе записываемые в кавычках слова, символы и пробелы для операторов PRINT (так как они хранятся только в исходном БЕЙСИК-листинге, а объектный код содержит лишь адреса обращения к ним)*;

* Одна из программных «хитростей», позволяющих использовать в исходной БЕЙСИК-программе любые текстовые константы без ограничений, описывается в статье И. В. Канивца, опубликованной в этом выпуске журнала. — *Прим. ред.*

- текстовые переменные, вообще говоря, допускаются, но их значения можно вводить только операторами INPUT и READ (в операторах DATA текстовые константы допустимы);
- при запуске автономного кода из ведущей программы на БЕЙСИКе в памяти существуют старый (автономный) и новый объектные коды, старая и новая области переменных — как простых, так и текстовых, а также элементов массивов. При инициализации описателей массивов перед записью автономного кода на ленту сами эти массивы будут располагаться в старой области хранения. Текстовые же переменные обычно записываются динамически в текущую, т. е. при работе автономизированного кода — в новую область текстовых переменных. (Это надо иметь в виду при использовании оператора CLEAR);
- чтобы в памяти хватило места для ведущей БЕЙСИК-программы, ее объектного кода, переменных и области ввода-вывода, автономный объектный код должен начинаться не очень рано. (Для увеличения адреса начала автономного кода при его генерации в исходную программу можно включить строки комментариев. При этом немного удлиняется таблица адресов, что также необходимо учитывать.) Если старая область ввода-вывода использоваться не будет, целесообразно сделать так, чтобы область переменных заканчивалась как раз перед началом экранной памяти и свободной зоны памяти почти не оставалось. Для этого, помимо удлинения исходного текста за счет комментариев, надо сдвинуть максимальный адрес (MAXAD) в экранную область. (Это можно сделать, например, включив режим расширенной памяти);
- ведущая программа на БЕЙСИКе может решать часть вычислительной задачи перед передачей управления объектному коду. Ее переменные размещаются в новой области и не связаны с переменными автономного кода. Наиболее простой способ передачи данных из ведущей программы объектному коду — использование операторов POKE и PEEK, но он годится только для целых значений. (Чтобы не разыскивать места размещения переменных в объектном коде, при его генерации также можно применить встречные операторы POKE и PEEK и воспользоваться для передачи данных буферными ячейками с фиксированными адресами где-то в свободной зоне памяти.)

Запуск нескольких программ в автономном коде

Как известно, в память ПЭВМ одновременно можно ввести несколько программ на БЕЙСИКе, имеющих разные номера строк, причем при запуске любой из них транслируются все имеющиеся программы. В автономном объектном коде также может содержаться несколько программ. Для управления ими предлагается два варианта:

- объединение программ в одну с развилкой, т. е. с оператором условного перехода по нужным адресам. Переменная — указатель перехода может запрашиваться оператором INPUT или читаться из фиксированной ячейки оператором PEEK;
- передача номера выполняемой программы через параметр оператора USR в ведущей программе. Стыковочная подпрограмма тогда будет иметь вид:

```
<start>: 11501 6301 62701 <adrtab> 11104 134
<adrtab>: <a1> <a2> <a3> ...
```

С адреса **<adrtab>** размещается список начальных адресов для программ с номерами 0, 1, 2 и т. д. Эти адреса можно узнать на этапе генерации автономного кода по таблице адресов: номер первой по счету строки программы будет записан в ячейке **FCB-2**, где **FCB** — адрес начала области ввода-вывода, содержащийся в ячейке 2026. Адрес же соответствующей точки объектного кода находится в ячейке, которая имеет адрес еще

на 4 меньше. Соответственно, данные для n-й по порядку строки размещены по адресу на 6 •(n-1) меньше, чем для первой строки.

Рационализация использования памяти

Чтобы получить экономию в использовании памяти, нужно определенным образом регулировать размещение автономного кода и данных. Если к объектному коду не предполагается добавлять чисто машинные части программы, этот код и данные целесообразно максимально сдвинуть к старшим адресам ОЗУ. Область данных должна заканчиваться чуть выше начала экранной памяти, свободный участок ОЗУ должен быть минимальным, а объектный код, динамически распределяемая область и область данных — примыкать друг к другу. Как уже упоминалось, для смещения объектного кода к старшим адресам исходную программу искусственно удлиняют строками комментариев, оператором CLEAR устанавливают малый размер области текстовых переменных (предполагается, что старая область практически не будет использоваться), а предельный адрес MAXAD устанавливают так, чтобы содержимое ячейки 2024 было несколько меньше 40000 (MAXAD при этом равен 40000+1022+6 •n-k, где **n** — число строк программы, **k** — небольшой запас плюс резерв для размещения машинных подпрограмм). Предварительно следует переключить экран в режим расширенной памяти. Производя пробные запуски программы (она не обязательно должна при этом отрабатываться до конца), с помощью команды FREE следите за размером неиспользуемой памяти, добавляя комментарии так, чтобы оставалось 20—30 байтов для запаса. (Помните, что при добавлении комментариев изменяется длина таблицы адресов и, возможно, надо будет подкорректировать MAXAD.) После достижения приемлемого распределения памяти запоминаем адрес начала объектного кода из ячейки 2002 (если нужно, ищем адреса требуемых строк в таблице адресов) и записываем объектный код и область данных на ленту.

Для автономной работы с объектным кодом устанавливаем MAXAD несколько меньшим, чем адрес его начала. Длину области текстовых переменных (новой) задаем по потребности. Для операций с магнитной лентой теперь может использоваться область ввода-вывода, если же она не нужна, MAXAD можно дополнительно увеличить на 400—420 байт и использовать свободное место от MAXAD-1022 до MAXAD-430, например, для машинных подпрограмм.

Поскольку для получения большого по объему объектного кода нужен исходный текст значительной длины, мы не можем получить код, занимающий почти всю неэкранную оперативную память: как правило, половина ее или несколько меньше остается свободной. В ней можно разместить ведущую программу на БЕЙСИКе, которая будет решать первую часть задачи почти в обычном режиме, но с новым распределением памяти, а затем передавать управление второй части в виде автономного кода.

Подпрограммы в виде автономного кода

Автономные объектные подпрограммы отличаются от программ тем, что возвращают управление ведущей БЕЙСИК-программе (точнее, ее объектному коду). Вторая их особенность — более развитой аппарат передачи параметров из ведущей программы в автономный код и обратно.

Подготовка автономного кода для подпрограмм осуществляется так же, как было сказано выше, за тем исключением, что подпрограмма (или подпрограммы) завершается оператором RETURN. Для перехода к различным подпрограммам можно построить развилку, как было показано ранее. Можно также, найдя начальные адреса подпрограмм, использовать в вызывающей программе несколько функций типа USR. Особенность отладки кода для подпрограмм состоит в том, что пробные пуски порождающей программы будут давать ошибку из-за нештатного использования оператора RETURN (RETURN

без GOSUB). Для пробных пусков можно добавить (не обязательно с младших номеров строк) балластную главную программу, которая вызывала бы создаваемые подпрограммы, но в автономном режиме не использовалась.

Стыковочная машинная подпрограмма в подобном случае может быть такой:

```
<start>: 11537 <ad.ind> 10546 13727 2134
<start+12>: <авт> 12704 <аааа> 134
<аааа>: 155526 <bbbb> <cccc>
<cccc>: 13737 <cccc+12> 2134 12605 207
```

Здесь **<ad.ind>** — адрес переменной, управляющей вычисленным переходом в объектном коде, **<bbbb>** — адрес начала самого объектного кода, точнее оператора вычисленного перехода ON. Стыковочная программа запоминает в стеке содержимое регистра R5, а в ячейке **<start+12>** — рабочую переменную системы БЕЙСИКа (из ячейки 2134). С адреса **<аааа>** начинается фрагмент объектного кода, в котором и вызывается подпрограмма. В завершающей части стыковочной программы (адрес **<cccc>**) восстанавливается R5 и содержимое рабочей ячейки, после чего производится возврат в вызывающую программу.

Другой вариант стыковочной программы предусматривает вместо развилки в объектном коде выбор из таблицы и подстановку в ячейку **<bbbb>** адреса нужной подпрограммы.

(Примечание: если младшая строка программы — оператор ON для развилки, адрес **<ad.ind>** можно выбрать из ячейки **<bbbb>+2**.)

Для всех описанных вариантов вызов автономной подпрограммы осуществляется функцией USR(n%), где n% — номер подпрограммы в соответствии с оператором ON или таблицей адресов подпрограмм.

Параметры целого типа могут передаваться в обе стороны, как это описывалось раньше, — через операторы POKE И PEEK, для передачи действительных чисел и текстовых переменных можно использовать в ведущей программе дополнительные функции USR1, USR2,... и соответствующие стыковочные блоки в машинных кодах.

Вот одна из таких методик. Пусть для всех подпрограмм исходные параметры действительного типа будут одни и те же: P1, P2, P3... (их количество может различаться). Анализируя полученный объектный код, надо определить адреса этих переменных. (Для облегчения поиска можно включить в программу балластную строку типа P4=P1+P2+P3, в которой эти адреса легко распознаются.) Теперь стыковочная функция USR1(P) будет передавать значение параметра P в переменную автономного кода P1, а функция USR2(P) при последовательных ее вызовах после USR1 — загружать параметры P2, P3, P4 и т. д.

Передачу нескольких параметров и вызов автономной подпрограммы можно оформить на БЕЙСИКе одной строкой, вводя, например, фиктивное сложение:

```
I% = USR1(A) + USR2(B) + USR2(C) + USR(n%)
```

Стыковочные машинные подпрограммы в этом случае будут такими:

```
<start1>: 12737 <adtab> <rab>
<start2>: 13701 <rab> 11101 10500
          12021 12021 12021 12021
          62737   2   <rab> 207
<adtab>: <adP1> <adP2> <adP3> ...
<rab>: <avt>
```

Функция USR1 начинается с адреса **<start1>**, функция USR2 — с адреса **<start2>**, таблица с адресами переменных — с **<adtab>**, а **<rab>** — ячейка для хранения текущего (наращиваемого) адреса из таблицы **<adtab>**.

При необходимости нетрудно аналогичным образом построить обратные функции, передающие значения переменных из автономного кода в вызывающую программу.

Массивы и текстовые переменные можно передавать, не пересылая их непосредственно, а лишь заменяя адреса в описателях*. Если вначале (после чтения с ленты) автономная программа работает с массивом, размещенным в старой динамически распределяемой области, то после замены из ведущей программы адреса в описателе она начинает работать с массивом, размещенным в новой области. Вызов стыковочной подпрограммы при этом должен выглядеть как **I% = USR3(M(0))**, где **M** — имя массива. Текст же подпрограммы будет таким:

<start>: 10401 162701 14 11101 11137 <adop> 207

Здесь **<adop>** — адрес описателя массива в автономном коде, который надо разыскать. Аналогично для замены адреса текстовой переменной в ее описателе стыковочная подпрограмма будет следующей:

<start>: 10401 162701 10 12137 <adop> 11137 <adop+2> 207

Здесь **<adop>** — тоже адрес описателя, но в данном случае нужно передавать два слова: длину текстовой переменной и адрес ее начала, так как и содержимое, и местоположение переменной при работе автономного кода могут измениться.

Чтобы не разыскивать в объектном коде адреса описателей, можно при его подготовке применить встречные приемные функции типа USR. Передающая подпрограмма перед вызовом автономного кода должна запомнить описатель (или переменную) в фиксированных ячейках, приемная — взять оттуда данные и переслать по назначению. Передача завершится, когда в автономном коде сработает приемная программа, причем она может быть вызвана несколько раз.

Текст передающей подпрограммы для текстовой переменной будет тот же, что и указанный выше, но в качестве **<adop>** нужно использовать адрес фиксированной свободной ячейки (4 байта). Приемная подпрограмма вызывается оператором вида: E$=USR(C$), где E$ — переменная, отличная от C$, значение которой загружается (иначе результат загрузки будет затерт). Текст приемной подпрограммы:

<start>: 10401 162701 10 11101 13721 <adop> 13711 <adop+2> 207

Здесь **<adop>** (4 байта) — фиксированные ячейки, где сохраняется загружаемый определитель текстовой переменной.

Более сложная методика предусматривает для вызова автономной подпрограммы (с передачей ей параметров) использование стыковочной машинной подпрограммы совместно с функцией типа FN. (Известно, что в данной, не вполне совершенной версии БЕЙСИКа только в функциях этого типа предусмотрена передача фактических параметров списком, при записи их в скобках.) Однако возможности построения самой функции очень ограниченны — это одна строка выражения, значение которого присваивается результату. Опишем функцию так:

DEF FNA (I%, P1, P2, P3) = USR4(I%)

* Аналог передачи через указатель. — *Прим. ред.*

(Здесь мы предположили передачу одного управляющего индекса и трех вещественных параметров.)

Приведем также текст машинной подпрограммы USR4:

```
<start>:  13727 2134 <авт> 11502 6302 62702 <adtab1>
          1203 62702 <dadr> 11205 12302 10601 12100 12100
          1300 12120 12120 12120 12120 62703 2 77210
          10537 <eeee> 12704 <bbbb> 134
<bbbb>: 155526
<eeee>: <авт> <ffff>
<ffff>: 13737 <start+4> 2134 207
<adtab1>: <adpar0> <adpar1> <adpar2> ...
<adtab2>: <adpp0> <adpp1> <adpp2> ...
        . . . . . .
<adpar0>: <npar> <adpn> ... <adp2> <adp1>
<adpar1>: <n1par> <ad1pn>... <ad1p2> <ad1p1>
        . . . . .
```

Здесь **<adtab1>** — адрес начала первой таблицы, в которой содержатся адреса вторичных таблиц для определения параметров подпрограмм, **<adtab2>** — адрес начала второй таблицы с адресами точек входа в соответствующие подпрограммы, **<dadr>** — разность адресов **<adtab2>** и **<adtab1>**. Фрагмент объектного кода начинается с адреса **<bbbb>** и содержит псевдокоманду 155526, после которой в ячейке **<eeee>** следует пересланный из таблицы **<adtab2>** адрес начала подпрограммы. Данный вариант стыковочной подпрограммы временно сохраняет содержимое рабочей ячейки 2134, но не сохраняет R5.

Во вторичных таблицах **<adpar0>**, **<adpar1>**,... размещаются: количество передаваемых вещественных параметров для данной подпрограммы (**<npar>**, **<npar1>**, **<npar2>**...), а затем адреса соответствующих параметров в объектном коде в обратном порядке (от последнего в списке при вызове функции FN до второго, так как первым параметром в этом списке является номер вызываемой подпрограммы I%). Поскольку различным подпрограммам требуется разное число параметров, а система БЕЙСИКа проверяет их количество при вызове функции FN, то приходится либо задавать лишние параметры при вызове функции (второй, третий и т. д.), либо подготовить несколько функций, отличающихся числом параметров.

Из различных вариантов последняя форма обращения к автономным подпрограммам оказалась наиболее удобной и приближенной к современным нормам (сходной с CALL-вызовами в языках типа ФОРТРАН). Вместо численных значений I% можно заготовить несколько мнемонических констант (например, для подпрограмм вычерчивания прямой линии и окружности — LN%=1%, OK%=3%). Другой способ: для каждой из подпрограмм заготовить свою функцию типа FN с подходящим именем. В этом случае параметр I% включать в эти функции не требуется, а аргументы функций USR будут константами. При этом немного изменится стыковочная подпрограмма. Также она изменится, если для всех подпрограмм параметры будут передаваться в одни и те же ячейки (вместо нескольких будет только одна таблица адресов параметров). Программисту же для составления стыковочной программы надо разыскать в объектном коде адреса подпрограмм и параметров, для чего можно воспользоваться описанными выше приемами. Соответствующие вспомогательные части объектного кода (вычисленные переходы с адресами подпрограмм и суммирование параметров) можно затем затереть чем-то полезным, например той же стыковочной подпрограммой.

По приведенному выше образцу можно подготовить варианты стыковочных подпрограмм, возвращающих в случае надобности значения параметров в ведущую программу.

Смещение автономного кода и переменных

При должных навыках сгенерированный объектный код можно изменять и дополнять. При загрузке в другое место (смещении в памяти) надо изменить в нем адреса для операторов перехода, вызова подпрограмм, циклов и др., аналогично изменяются адреса переменных, описателей массивов и т. д. Текстовые константы можно перенести из исходного текста программы поближе к объектному коду или к области переменных и соответственно изменить в объектном коде ссылки на них. На этом этапе работы можно также скомпоновать объектный код из нескольких отдельно оттранслированных частей и при необходимости занять под него и переменные почти всю неэкранную память. Кроме того, в отдельных публикациях отмечалось, что к операторам (псевдокомандам) объектного кода можно добавлять дополнительные, дающие эффект экономии памяти или решающие специальные задачи. (Как известно, операторы объектного кода — это начальные адреса обрабатывающих подпрограмм, завершающихся кодом возврата 134.)

Разработаны специальные программы, перенастраивающие адреса в объектном коде (но пока они рассчитаны на частные случаи). Автоматизировав также подготовку стыковочных подпрограмм, можно прийти к системе, напоминающей загрузчик или редактор связей в более мощных языках.

Использование части экранной памяти

Предположим, что верхняя треть или четверть экрана будет «пожертвована» для размещения программы и данных в расчете, что оставшейся части будет достаточно для вывода текстовой и даже графической информации. В данном случае требуется предохранить программу и данные от затирания (например, при гашении экрана или его прокрутке).

Максимальный используемый адрес (MAXAD) в рассматриваемом варианте задается оператором CLEAR, большим 40000, например равным 50000 или 50420. Подбирая нестандартные значения рабочих ячеек БЕЙСИКа и мониторной системы, добиваемся, что верхняя часть экрана блокируется от вывода текстовой и графической информации, от гашения оператором CLS и от прокрутки. На ней может возникать случайный узор (как правило, он соответствует таблице адресов БЕЙСИК-программы и данным). Если это раздражает, на экран можно установить непрозрачную шторку. Подобранные автором значения рабочих ячеек дают эффект, когда прокрутка осуществляется только для четырех нижних строк. Перемещать курсор на верхние строки экрана (те, что остаются доступными) можно оператором CLS и клавишей СБР, а также оператором LOCATE. (Клавиша СБР опасна: при случайном ее нажатии одновременно с АР2 может переключиться режим работы экрана с его очисткой, а значит, и с затиранием части программы или данных. Вместо использования клавиши СБР можно занести команду CLS в какой-либо ключ, желательно с кодом 12_8, запускающим ее.)

Для включения описанного режима частично расширенной памяти подготовлены варианты программ на БЕЙСИКе и в машинных кодах. Приводим текст первого из них (без простановки номеров строк и условно располагая по нескольку операторов в строке):

```
POKE &O164, &O2100: POKE &O42, &O377: POKE &O202, &O50000
POKE &O204, &O2000: POKE &O206, &O27040: POKE &O210, &O25040
POKE &O177664, &O1330: CLS
FOR I% = &O40000 TO &O47776 STEP 2: POKE I%, 0: NEXT
POKE &O2160, &O2040: POKE &O2162, &O46103: POKE &O2164, &O5123
CLEAR 8, &O50420: END
```

Оператор CLEAR здесь задает небольшую область текстовых переменных (значение можно изменить) и адрес MAXAD, больший адреса начала рабочей области экрана. Подразумевается, что область ввода-вывода использована не будет. (Более стандартный вид этого оператора: CLEAR 256,&O50000.)

Сначала в ПЭВМ нужно ввести и запустить данную программу, при этом режим экрана изменяется. Затем она стирается командой NEW, а пользовательская программа и ее машинные части (если они есть) читаются командами CLOAD и BLOAD (команда LOAD недопустима, так как она занимает рабочую область ввода-вывода).

Другой вариант программы ликвидирует на экране служебную строку, за счет чего увеличивается рабочая область. Однако при переключении регистров РУС/ЛАТ и в некоторых других случаях происходит аварийный останов. (Отличие от предыдущего варианта в том, что в ячейку 164 заносится значение 2200, в ячейку 202 — адрес 50300, в 204 — нуль, а в 206 и 210 — код 27440.) В этом режиме для вывода графической информации используются 188 строк развертки экрана.

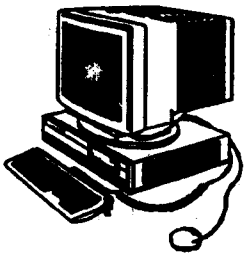
Читатели могут разработать и другие, более подходящие для них варианты частичного расширения неэкранной памяти.

Возможность секционной (оверлейной) обработки программы на БЕЙСИКе

Предположим, что большая исходная программа разделена на ряд секций, работающих последовательно или повторно-последовательно, и что каким-то образом (например, через фиксированные ячейки) налажена передача данных между этими секциями. Идея состоит в том, что исходный текст (возможно, с таблицей адресов и в каком-то сжатом виде) хранится в старших зонах памяти (после MAXAD). Машинная программа-монитор выбирает секции в нужном порядке, переносит их в штатные места (исходный текст — с адреса 3052, номера и адреса строк — в таблицу адресов TABADR) и после этого передает управление системе БЕЙСИКа (в точку, соответствующую обработке оператора RUN). Заканчивается работа такой секции через функцию USR возвратом в монитор, который запускает следующую секцию. Трансляция и генерация объектного кода, таким образом, производятся по частям и сравнительно небольшими порциями. Эффективность метода зависит от соотношения количества секций и их размеров, от длины монитора, степени сжатия текста и т. д. Определенное снижение эффективности будет вызывать повторная трансляция отдельных секций, если они загружаются несколько раз.

Секционированная обработка автономного объектного кода сочетает высказанную идею с ранее описанной. Предполагаем, что каждая секция программы оттранслирована в ограниченном объеме памяти (возможно меньшее значение MAXAD) и все объектные коды, временно записываемые на ленту, затем объединяются и совместно помещаются в ОЗУ, после предельного адреса MAXAD. Если нужно, туда же помещается описание переменных для разных секций. Специальный монитор в требуемом порядке пересылает секции кода и переменные на их расчетные места и запускает их. Потерь на трансляцию здесь не будет, пересылка также выполняется быстро. Остается один существенный вопрос — передача данных между секциями. Для запускающей программы на БЕЙСИКе память перераспределяется повторно (с еще меньшим MAXAD).

И наконец, если выше при обсуждении работы автономного кода предполагалось, что его длина ограничена примерно половиной используемой памяти, в данном варианте суммарная длина секций кода может быть больше.



Идея использования шитого (или объектного, как точнее было бы его называть) кода отдельно, без сохранения в памяти исходного листинга БЕЙСИК-программы, выдвигалась различными авторами неоднократно. Но вот беда: шитый код содержит ссылки на текстовые константы, содержащиеся в листинге, а значит, далеко не всякая программа будет правильно работать в «объектном» варианте...
И все же, если очень хочется, то можно! Описываемая в данной статье разработка позволяет существенно снизить объем памяти, занимаемой программой на БЕЙСИКе, оставляя в ОЗУ только те строки исходного листинга, которые содержат пресловутые константы (и, разумеется, запускаемый шитый код).

И. В. Канивец,

С.-Петербург

Укорочение БЕЙСИК-программ при трансляции

Многим пользователям БК-0010.01, пишущим программы на вильнюсском БЕЙСИКЕ, нередко приходится сталкиваться с проблемой нехватки памяти. Казалось бы, предоставляемые БЕЙСИКом 14 кб — достаточно большой объем, однако фактически он исчерпывается программой из 200—250 строк. Втиснуть же мало-мальски добротную и конкурентоспособную программу (по сравнению с ассемблерными) в этот объем практически нереально. Владельцам БК хорошо известно, что это связано с особенностями транслятора БЕЙСИКа: во время исполнения программы в памяти одновременно хранится исходный текст и так называемый шитый код. Методы же программирования, предложенные ранее в литературе (см., например, [1]), дают крайне незначительную экономию памяти при резком ухудшении читаемости листинга.

Предлагаю читателям способ, позволяющий значительно увеличить объем памяти, доступной программе во время выполнения, — на 3—4 кб для программ в 200—250 строк. Основная идея состоит в том, чтобы максимально избавиться от исходного листинга в ОЗУ, увеличив тем самым объем памяти для шитого кода и переменных.

Теоретически, когда БЕЙСИК-программа уже оттранслирована, в памяти достаточно иметь только ее шитый код. Но последний неперемещаем, поэтому его нельзя просто «сдвинуть» к младшим адресам ОЗУ, «наложив» на текст программы. Кроме того, шитый код содержит ссылки на строковые константы в листинге и на информацию в операторах DATA.

Во время трансляции шитый код записывается в ОЗУ сразу же за листингом, адрес окончания последнего БЕЙСИК-система хранит в ячейке &O2002. (Сведения об используемых специфических адресах БЕЙСИК-системы можно найти в литературе, например [2] и [3].) Чтобы сделать адрес начала шитого кода как можно меньшим, увеличив тем самым объем свободной памяти для работы программы, можно поступить следующим образом. Перед трансляцией весь листинг разделяется на две группы строк. В первую включаются строки, содержащие текстовые константы или операторы DATA (на которые будут осуществляться ссылки из шитого кода). Эти строки размещаются в памяти с адреса начала текста программы в БЕЙСИК-системе — &O3052 и в ячейку &O2002 заносится адрес окончания именно этой группы строк.

Вторую группу составляют все остальные строки, не нужные для нормальной работы шитого кода БЕЙСИК-системы. Они помещаются в экранную память начиная с адреса &O40000 и тем самым не занимают место в ОЗУ пользователя.

Во время перемещения строк соответственно изменяются значения в списке их номеров и адресов, используемом БЕЙСИК-системой.

После этого производится трансляция листинга обычным образом с записью шитого кода в ОЗУ пользователя сразу же после первой группы строк. Транслятор находит строку по ее адресу в таблице номеров и адресов, не накладывая никаких ограничений на место ее расположения в памяти. Когда шитый код сформирован, вторая группа строк уже не нужна и экранную память можно очистить. Таким образом, в ОЗУ после трансляции программы остается только то, что необходимо для ее выполнения: шитый код, текстовые константы и данные из операторов DATA.

Разумеется, после таких действий исходный текст БЕЙСИК-программы теряется, и, чтобы запустить ее снова, нужно опять загрузить листинг, возможно, внести в него требуемые изменения и вновь повторить приведенную выше последовательность шагов. Таким образом, дополнительная свободная память оплачивается постоянным обращением к тексту программы на внешнем носителе. Поэтому время отладки программы сильно возрастает и использовать данный прием следует лишь на заключительном этапе. (Например, можно отлаживать программу с уменьшенными размерами используемых массивов, не допуская появления ошибки переполнения памяти, а когда отладка закончена, оттранслировать программу по предложенному методу уже с реальными размерами массивов.)

Чтобы пользоваться сгенерированным кодом программы многократно, нужно в качестве ее первого оператора поставить STOP. Тогда после останова можно записать на магнитофон участок памяти с шитым кодом, начиная с адреса &O2000 до &O40000. Впоследствии, после его загрузки командой BLOAD программа запускается командой CONT (но не RUN !).

Описанный алгоритм реализован в программе UKOR, ассемблерный и кодовый листинги которой приведены ниже. Порядок работы с ней следующий (считаем, что листинг БЕЙСИК-программы уже находится в памяти).

- Первым оператором в программе ставится **STOP**.
- Загружается (лучше всего с адреса &O400, но можно и с любого другого, так как программа перемещаема) и запускается предварительно оттранслированная программа UKOR: **BLOAD "UKOR",R,&O400** .
- БЕЙСИК-программа запускается командой **RUN**, при этом можно наблюдать, как в экранной памяти формируется вторая группа строк.
- Оттранслированная БЕЙСИК-программа, начав выполняться, сразу же останавливается по STOP.
- Шитый код записывается на магнитофон: **BSAVE "<имя>",&O2000,&O40000.**
- Когда впоследствии необходимо запустить готовую БЕЙСИК-программу, она загружается командой **BLOAD "<имя>"** и запускается командой **CONT**.
- Чтобы произвести обработку другого текста БЕЙСИК-программы (или внести изменения в тот же самый), необходимо повторить весь процесс заново.

Программа UKOR активизируется сразу после подачи команды RUN вместо «штатной» процедуры БЕЙСИК-системы, расположенной начиная с адреса &O140716 и предназначенной для «сборки мусора» (т. е. удаления незаполненных участков в памяти, образующихся в результате редактирования листинга между строками БЕЙСИК-программы).

Маленькое замечание для тех, кто включает ассемблерные процедуры или данные в состав операторов комментариев. Все такие строки должны находиться в начале программы и начинаться оператором «'» (апостроф), но не REM. Только тогда эта строка будет включена в первую группу и останется в обычной памяти.

Никаких других ограничений на листинг БЕЙСИК-программы не накладывается.

Литература

1. Экономия памяти БК-0011.01 // Информатика и образование, 1990 №3. С. 44—49.
2. Совместное использование БЕЙСИКа и машинных кодов. //Вычислительная техника и ее применение. 1992, №3. С. 30—33.
3. Особенности транслятора с языка БЕЙСИК для БК-0010.01. //Информатика и образование. 1990, №2. С. 42—46.

Ассемблерный листинг программы UKOR

```
; ******************************
; Начальный загрузчик
        MOV PC,R0
        .#62700 .@LEMT+2
        MOV R0,@#30
        JMP @#120234
; ******************************
; Отлов прерывания EMT
LEMT:   CMP 4(SP),#135312
        BEQ OTR
; Нормальная отработка EMT
        MOV R5,-(SP)
        MOV 2(SP),R5
        MOV -(R5),R5
        JSR PC,@174000(R5)
; Интересный прием, позволяющий
;  обойтись без оператора BIC
        MOV (SP)+,R5
        RTI
; ******************************
; Отловлено нужное прерывание
; Подготовка системных адресов
OTR:    MOV #2036,@#2040
        CLR @#2052
        CLR @#2036
M10:    INC @#2032
        BEQ M10
        INC @#2000
        MOV R4,@#2046
        MOV @#2024,@#2016
        MOV @#2024,@#2020
        MOV @#2024,@#2022
        CLR @#2004

        MOV R0,-(SP)
        CLR R4
        MOV #3052,R1
; Адрес приемника в ОЗУ:
;  сюда будут копироваться
;  строки первой группы
        MOV #40000,R0
; Адрес приемника в
;  видеопамяти для
;  строк второй группы
;
; Ищем очередную строку
;  текста программы
CYCLE:  MOV #177777,R3
        MOV @#2026,R2
        CMP -(R2),-(R2)
M0:     CMP (R2),#40000
        BHIS M1
        CMP (R2),R4
        BLOS M1
        MOV (R2),R3
        MOV R2,R5
M1:     CMP -(R2),-(R2)
        CMP R2,@#2024
        BCS MF
        CMP -(R2),R3
        BCS M0
        BR M1

; Найден адрес очередной
;  строки в памяти
MF:     CMP R3,#177777
        BEQ END

; R3 - адрес начала строки
;  в исходном тексте
; R5 - адрес ячейки, содержащей
;  адрес данной строки
        MOV R3,R2
; Продублировать адрес строки
        CMPB (R3),#100
; Оператор DATA ?
        BEQ L0
        CMPB (R3),#2
; Оператор ' (апостроф) ?
        BEQ L0
        CMPB (R3),#50
; Оператор REM ?
        BEQ HI
; Ищем вхождение знака кавычек
        TSTB (R3)+
; Первый байт пропускаем
M2:     CMPB (R3),#42
        BEQ L0
; Знак кавычек найден
        CMPB (R3)+,#12
```

```
; Конец строки?
        BNE M2

; Перемещение в видеопамять
;  (первая группа)
HI:     MOV R0,(R5)
        MOV R2,R4
        MOVB (R2)+,(R0)+
M3:     MOVB (R2)+,(R0)
        CMPB (R0)+,#12
        BNE M3
        BR CYCLE
; Перемещение в ОЗУ пользователя
;  (вторая группа)
LO:     MOV R1,(R5)
        MOV R2,R4
        MOVB (R2)+,(R1)+
M4:     MOVB (R2)+,(R1)
        CMPB (R1)+,#12
        BNE M4
        BR CYCLE
; Окончание обработки
;  листинга БЕЙСИК-программы
END:    MOV (SP)+,R0
        MOV R1,@#2002
        INC R1
        BIC #1,R1
        MOV R1,@#2014
        MOV #100112,@#30
; Восстановить вектор
;  прерывания по команде EMT
        MTPS #0
; Разрешить прерывания
        JMP @#135400
; Перейти к стандартной
;  трансляции
    END
```

Кодовое представление программы UKOR

```
010700 062700 010037 000030 000137 120234 026627 000004
135312 001410 010546 016605 000002 014505 004775 174000
012605 000002 012737 002036 002040 005037 002052 005037
002036 005237 002032 001775 005237 002000 010437 002046
013737 002024 002016 013737 002024 002020 013737 002024
002022 005037 002004 010046 005004 012701 003052 012700
040000 012703 177777 013702 002026 024242 021227 040000
103004 021204 101402 011203 010205 024242 020237 002024
103403 024203 103763 000771 020327 177777 001441 010302
121327 000100 001425 121327 000002 001422 121327 000050
001407 105723 121327 000042 001413 122327 000012 001372
010015 010204 112220 112210 122027 000012 001374 000721
010115 010204 112221 112211 122127 000012 001374 000711
012600 010137 002002 005201 042701 000001 010137 002014
012737 100112 000030 106427 000000 000137 135400 000000
```

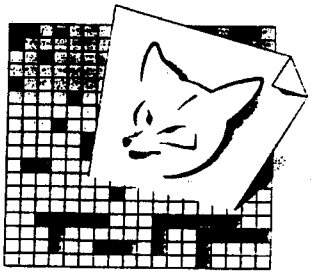
Длина: 000360. Контрольная сумма: 142311.

Примечание редактора

Для уменьшения объема хранящегося на магнитной ленте файла содержимого ОЗУ по адресам &O2000—&O40000 и времени его загрузки оператором BLOAD можно воспользоваться архиватором для программ в машинных кодах BKPACK. О том, как это сделать, вы можете прочитать в статье «Компактное хранение БЕЙСИК-кодовых программ» («Персональный компьютер БК-0010 — БК-0011М». 1994. №2. с. 60.)

* * *

Автором опубликованных в пятом выпуске за 1994 г. полезных советов по БЕЙСИКу БК-0010.01 и БК-0011М (стр. 69 вверху) является Д. Б. Лаврик. Приносим автору и читателям свои извинения.



Когда возможности вильнюсского БЕЙСИКа оказываются недостаточными, возникает заманчивая мысль дополнить язык собственными операторами, расширить функции существующих и т. д. Но для этого необходимо «хирургическое вмешательство» в «святое святых» БК-0010.01 — в работу зашитого в ПЗУ БЕЙСИК-транслятора. В данной статье приводится общая идея осуществления подобного вмешательства и ряд полезных для этого сведений, в частности описание служебных TRAP-функций БЕЙСИКа.

Подключение кодовых подпрограмм к БЕЙСИК-программам на этапе трансляции

Не секрет, что для написания хорошей программы на БЕЙСИКе его возможности нередко приходится дополнять подпрограммами в машинных кодах (особенно в играх). Но стандартный механизм их вызова (с помощью оператора USR) при всех достоинствах обладает и рядом недостатков. Во-первых, таких функций всего десять, и, если требуется большее их количество, приходится идти на всевозможные ухищрения с объединением нескольких подпрограмм в один «USR-блок», с переопределением USR-функций в процессе работы программы с помощью DEF USR и т. д. Во-вторых, стандартно в USR-подпрограмму можно передать только одно значение аргумента и, соответственно, вернуть из нее только один результат, причем лишь того же самого типа, что и аргумент.

Сам БЕЙСИК (а точнее, его разработчики) «заботится о себе» гораздо больше, чем о пользователях: вызов подпрограмм, прошитых в ПЗУ транслятора и реализующих те или иные операторы, производится более рациональным путем, позволяющим передавать и возвращать любое количество параметров любого типа. Это механизм шитого кода (данное название уже «прижилось» среди пользователей БК и стало почти стандартным, хотя на самом деле правильнее было бы называть генерируемую БЕЙСИК-транслятором последовательность адресов и данных объектным кодом). В нем каждая подпрограмма принимает и передает параметры через стек (для переписывания данных из шитого кода в стек также служат специальные подпрограммы транслятора). Передача управления от одной подпрограммы к следующей выполняется с помощью завершающей команды **JMP @(R4)+** (вместо обычной для подпрограмм **RTS R<номер>**), а чтение данных, как правило записываемых в шитом коде после адреса использующей их подпрограммы, производится в ней командой типа **MOV (R4)+,...** . (Подробнее о шитых кодах и обо всем, связанном с ними, можно ознакомиться в статьях В. В. Авсеева и А. В. Авсеева в журналах «Информатика и образование», №2 за 1990 г. , и «Вычислительная техника и ее применение», № 12 за 1990 г.)

Чтобы воспользоваться всеми «благами» такого метода для вызова своей подпрограммы, нам нужно «всего-навсего» разместить в шитом коде ее адрес и необходимые данные. Простейший способ (правда, он пригоден далеко не всегда), который удобен для замены стандартной реализации какого-либо оператора БЕЙСИКа на свою, состоит в следующем. В начале БЕЙСИК-программы в цикле (т. е. уже ПОСЛЕ трансляции) производится поиск и замена нужного шитого кода своим. Примером может являться программа В. В. Ермакова, реализующая правильную работу с многомерными массивами. Но данный способ имеет и свои неудобства — необходимость вставки в БЕЙСИК-программу специальной подпрограммы для поиска нужного шитого кода и его замены, просмотр шитого кода всей программы (что требует некоторого времени), возможность ошибочной замены вместо адреса вызова подпрограммы равного ему восьмеричного числа из списка данных и т. п.

Другой способ, о котором далее и пойдет речь, состоит в записи нужных адресов и данных в список шитых кодов ВО ВРЕМЯ трансляции программы. Для этого, в свою очередь, нужно вначале понять хотя бы в общих чертах алгоритм работы транслятора, а также познакомиться с функциями TRAP-прерываний, играющих в вильнюсском БЕЙСИКе важную роль.

Алгоритм компиляции БЕЙСИК-программы на БК-0010.01 в простейшем виде выглядит следующим образом.

- В очередной программной строке исходного листинга имя оператора (стоящее в строке первым) по очереди сравнивается со стандартными именами, прошитыми в ПЗУ в виде таблицы. Как только нужное имя найдено, в соответствии с его порядковым номером из другой таблицы, также прошитой в ПЗУ, извлекается и помещается в список шитых кодов нужный адрес подпрограммы, а адрес ячейки ОЗУ, в которую он попадает, записывается в таблицу связи номеров строк с адресами начала соответствующих им шитых кодов (эта таблица формируется в старших адресах ОЗУ). Условно действия по поиску имени оператора и записи его шитого кода можно представить в виде исполняемой в цикле команды: **IF X$=A$(N) THEN K=C(N)** , где **X$** — имя оператора в исходном листинге, **A$** — массив стандартных имен в ПЗУ, **K** — найденный шитый код, а **C** — таблица шитых кодов в ПЗУ.

Если после просмотра всей таблицы имен в ПЗУ сравнение не оказалось истинным (т. е. имя оператора не соответствует стандарту БЕЙСИКа), выдается сообщение об ошибке 2 (синтаксис) и трансляция прерывается.

Реально, конечно же, дело обстоит намного сложнее. Во-первых, имена сравниваются не целиком, а по двум-трем первым буквам, с распознаванием присваивания без LET. Во-вторых, сравнение имен БЕЙСИК производит уже при вводе программной строки (с клавиатуры или с магнитофона оператором LOAD) и заменяет стандартные имена их «внутренними кодами», по которым при трансляции и разыскиваются шитые коды. В-третьих, одному оператору БЕЙСИКа может соответствовать несколько шитых кодов (или, наоборот, один и тот же шитый код — нескольким операторам, например **PRINT** и **?**), да и заносятся они в список, как правило, после данных и адресов «сервисных» подпрограмм, переписывающих эти данные в стек. Однако для первого знакомства приведенной выше упрощенной картины вполне достаточно.

- Производится анализ оставшейся части программной строки до кода ее окончания 12_8 (код клавиши «ВВОД»). При этом незначащие пробелы пропускаются, а в исходном листинге выделяются скобки, запятые, кавычки, знаки арифметических и логических операций, имена переменных, текстовые константы, числа (их символьная запись перекодируется в цифровую) и т. д. Конкретный алгоритм анализа для каждого оператора свой. После этого извлеченные параметры (числа, ссылки на массивы, переменные и текстовые строки) в нужном порядке записываются вместе с адресами «сервисных» подпрограмм в список шитых кодов. При несоответствии заданных в строке параметров или порядка их записи стандарту БЕЙСИКа для данного оператора выдается сообщение об ошибке и трансляция прекращается.
- Производится переход к следующей по порядку номеров строке исходного листинга, а если строк больше нет — к исполнению сгенерированной последовательности шитых кодов. Для этого в регистр R4 заносится адрес начала списка шитых кодов, а затем дается «пусковая» команда JMP @(R4)+ .

Теперь пришло время поговорить о TRAP-прерываниях. Для БЕЙСИКа их роль не менее важна, чем EMT. Команды TRAP используются в вильнюсском БЕЙСИКе для двух целей: TRAP с номерами от 0 до 77_8 применяются для генерации сообщений об ошибках (правда, в настоящей версии БЕЙСИКа не все эти номера задействованы). При этом числовой аргумент команды TRAP выводится на экран в качестве номера ошибки (с учетом того, что в командах TRAP номера восьмеричные, а в сообщениях об ошибках они выводятся в десятичном виде). Возврат из TRAP-прерывания по ошибке производится не на следующую машинную команду программы транслятора в ПЗУ (с помощью RTI), а на режим непосредственного диалога с пользователем. Так что в машинных подпрограммах, входящих в состав БЕЙСИК-транслятора, TRAP-прерывания с номерами от 0 до 77_8 алгоритмически играют ту же роль, что и команды ассемблера BR или JMP: конец соответствующей логической ветви алгоритма (это нужно учитывать при анализе прошивки ПЗУ БЕЙСИКа).

Таким образом, можно использовать «отлов» TRAP-прерываний по ошибкам, чтобы поместить свой адрес и данные в список шитых кодов, а потом передать управление на продолжение трансляции. Общий вид возможного алгоритма такой подмены может быть таким.

- **Перехват TRAP-прерывания. Если его номер больше 100_8 или не равен требуемому номеру ошибки, управление нужно передать в стандартный драйвер обработки TRAP-прерываний БЕЙСИКа по адресу 146424_8. (TRAP с номерами 100_8 и более используются в трансляторе для**

служебных целей, чуть позже мы с ними ознакомимся.)

- Определение номера строки, в которой произошла ошибка. В системной ячейке @#2056 содержится номер последней оттранслированной программной строки, а так как при ошибке трансляция обычно прерывается аварийно, эта ячейка как раз и содержит нужный нам номер (БЕЙСИК использует эту ячейку при выводе сообщения «ОШИБКА... В СТРОКЕ...»). Затем следует проверить, соответствует ли текст, записанный в данной строке листинга, требуемому. Если это действительно ошибка (а не введенный нами «новый» оператор), нужно вернуть управление стандартному драйверу TRAP-прерываний для вывода сообщения об ошибке.
- Извлечение из стека всех сохраненных в процессе обработки TRAP-прерывания значений регистров процессора.
- Запись требуемых чисел (адресов и констант) в список шитых кодов с помощью ряда команд **MOV #<число>,(R1)+** (регистр R1 во время трансляции содержит адрес очередной, пока еще пустой ячейки списка).
- Нормализация стека. При вызове TRAP для сообщения об ошибке в стек стандартным путем заносится адрес следующей после TRAP машинной команды транслятора, содержимое ССП и другая нужная БЕЙСИКу информация. Теперь же необходимо «сдвинуть» указатель стека на 6 байт «вверх», чтобы привести стек в такое состояние, как будто ошибки не было.
- Последний и самый сложный этап. Нужно передать управление на нормальное продолжение трансляции следующей строки. Вообще-то этот адрес определяется в каждом конкретном случае после анализа всего алгоритма работы транслятора. По ассемблерному листингу содержимого ПЗУ (дезассемблируя прошивку ПЗУ, например, с помощью отладчика), следует проследить, как БЕЙСИК просматривает исходный листинг, где в данном случае передается управление на соответствующий оператор TRAP и в случае, если трансляция этой строки выполнена успешно. Вот этот-то «успешный» адрес нам и нужен! Конечно, такой анализ БЕЙСИКа под силу только хорошо подготовленному пользователю. Однако экспериментальным путем удалось выяснить, что если сразу после нормализации стека (команда **ADD #6,R6**) подать команду **RTI**, то в большинстве случаев трансляция продолжается. (Кстати говоря, адрес, на который нужно передать управление для продолжения трансляции, далеко не обязательно соответствует машинной команде, следующей после вызова TRAP-прерывания.)

Для иллюстрации всех этих рассуждений приведем несложный пример.

```
; драйвер TRAP-прерываний  ;  коды  ; комментарии
30000:  MOV R0,-(R6)       ;  10046 ; R0 и R2 занести
        MOV R2,-(R6)       ;  10246 ;  в стек (копия
        MOV 4(R6),R0       ;  16600 ;  стандартного
                           ;     4 ;  обработчика TRAP из ПЗУ)
        MOV 177776(R0),R2  ;  16002 ; номер TRAP -
                           ; 177776 ;  в регистре R2
        BIC #177400,R2     ;  42702 ;
                           ; 177400 ;
        CMP R2,#100        ;  20227 ; номер TRAP больше 100 -
                           ;    100 ;  перейти на стандартный
        BLO A$             ; 103402 ;  обработчик TRAP
        JMP @#146424       ;    134 ;
                           ; 146424 ;
A$:     MOV (R6)+,R2       ;  12602 ; извлечь R2 и R0
        MOV (R6)+,R0       ;  12600 ;  из стека
        MOV #32000,(R1)+   ;  12721 ; записать число 32000
                           ;  32000 ;  в качестве шитого кода
        ADD #6,R6          ;  62706 ; нормализовать стек
                           ;      6 ;
        RTI                ;      2 ; продолжить трансляцию
; реализация "оператора"
32000:  MOV R0, -(R6)      ;  10046 ; сохранить R0 в стеке
```

```
MOV #235,R0      ;  12700 ; код "инверсия экрана"
                 ;    235 ;
EMT 16           ; 104016 ;
EMT 16           ; 104016 ;
MOV (R6)+,R0     ;  12600 ; извлечь R0 из стека
JMP @(R4)+       ;    134 ; стандартная команда
                 ;        ;  для завершения подпрограмм
                 ;        ;  шитого кода
```

Здесь производится запись адреса #32000 в качестве шитого кода для строки с первой же встреченной при трансляции ошибкой. (Естественно, перед началом работы нужно выполнить оператор **CLEAR ,&O30000.**) После запуска БЕЙСИК-программы:

```
10 CLS
20 POKE &O34,&O30000
30 CLS
40 END
```

будет произведена очистка экрана, а в ячейку &O34 (вектор TRAP-прерывания) — занесен адрес нашего TRAP-драйвера. Намеренно сделаем в строке 30 синтаксическую ошибку: заменим в ней «CLS» на «CLR» и снова запустим программу на выполнение. И теперь, вместо обычного в подобных случаях сообщения об ошибке 2 и выхода в режим диалога с пользователем, программа начнет работать, очистит экран (строка 10) и дважды его проинвертирует (экран «мигнет»), а затем выдаст нормальное «Ok». После этого можно убедиться, что адрес нашей «мигающей» подпрограммы — число #32000 — действительно занесен в список шитых кодов, а адрес размещения его в ОЗУ — в таблицу соответствий номеров строк. Адрес начала шитых кодов можно определить по содержимому ячейки @#2002. Если оно нечетное, нужно увеличить его на 1. Просмотреть же содержимое ОЗУ поможет простейшая команда **?OCT$(PEEK(&O<адрес>))** .

Конечно, приведенный выше пример — не верх совершенства. К тому же «возврат из ошибки» здесь выполняется не слишком корректно, так что после выполнения БЕЙСИК-программы может оказаться блокированным действие клавиши «ВС», а при наличии в программе более одной строки с ошибкой БК и вовсе может перезапуститься или «вылететь» в монитор — БЕЙСИК все-таки вещь деликатная... Однако не забудем, что это лишь простейшая иллюстрация. Можно проверять содержимое исходного листинга, чтобы отсечь «просто ошибки» от «запланированных заранее». Можно предусмотреть анализ оставшейся части строки для выявления числовых параметров и занесения их в список шитых кодов. Можно, наконец, найти другой, более надежный адрес возврата из прерывания. Но думаем, что читателям будет интереснее сделать это самим. (В качестве исходного образца можно принять драйвер многоцветной закраски в вильнюсском БЕЙСИКе, опубликованный в журнале «Информатика и образование», №4 за 1993 г.)

Но продолжим нашу беседу о TRAP-прерываниях, а также поговорим о некоторых других полезных подпрограммах и ячейках БЕЙСИКа. Все это может очень пригодиться при разработке своих версий TRAP-перехватчиков, да и просто при написании программ в кодах.

TRAP с номерами 100_8 и больше, как уже было сказано ранее, служат в БЕЙСИКе для вызова часто используемых служебных подпрограмм, в большей части применяемых при трансляции. Почти все они осуществляют возврат из прерывания стандартным образом (в отличие от генерации сообщений об ошибках), т. е. передают управление на машинную команду, следующую после TRAP. Если же подпрограмма использует одно-два следующих машинных слова в качестве параметров, адрес возврата соответственно увеличивается. Всего таким способом вызывается 25 подпрограмм. Их адреса вызова и соответствие этих адресов номерам TRAP-прерываний приведены в таблице.

Номер TRAP	Адрес	Номер TRAP	Адрес
100	147324	115	153420
101	151646	116	154300
102	151704	117	154346
103	151760	120	154404
104	152176	121	154460
105	152422	122	154544
106	152744	123	154570
107	152774	124	154574
110	153072	125	154600
111	153154	126	154610
112	153210	127	151046
113	153222	130	151036
114	153402		

Рассмотрим некоторые из этих подпрограмм более подробно. (Сразу отметим, что регистры R1 и R3 почти во всех из них зарезервированы для служебных целей: R1 при трансляции содержит адрес очередной свободной ячейки списка шитых кодов, а R3 является указателем на текущий символ (байт) исходного листинга на БЕЙСИКе.)

- **TRAP 102** — используется при анализе строк листинга. В частности, позволяет при написании программы свободно пользоваться как заглавными, так и строчными латинскими буквами. *Входные данные:* R3 — указатель на символ текста программы, R0 — признак (исходно R0=0). *Результаты работы:* латинский строчный символ, на который указывает R3, преобразуется в латинский заглавный. В R0 заносится «флаг класса символа»: R0=4 — цифра, R0=2 — латинская буква, R0=0 — прочие коды.
- **TRAP 106** — используется при анализе строк листинга для проверки на конец строки и «отсечения» ремарок (символ «'»), а также для пропуска незначащих пробелов . *Входные данные:* R3 — указатель на текст листинга, R0 — флаг. R3 устанавливается на первый не равный пробелу символ текста. Если флаг R0 равен нулю, производится выход. Иначе проверяется код, на который указывает R3. Если это не «ВВОД» и не символ ремарки («'»), то генерируется сообщение об ошибке 2 (TRAP 2).
- **TRAP 107** — реализует последовательный вызов TRAP 112, TRAP 102 и TRAP 2.
- **TRAP 110** — очищает R5, «подгоняет» R3 к первому не равному пробелу символу исходного листинга и «тестирует» его (TRAP 102).
- **TRAP 112** — устанавливает R3 на первый не равный пробелу символ исходного листинга, смещаясь в сторону увеличения адресов. После выхода R3 содержит адрес этого символа. Данная функция как раз и позволяет сделать незначащие пробелы «невидимыми» для БЕЙСИКа.
- **TRAP 114** — очень полезная подпрограмма, позволяющая вывести на экран десятичное представление целого числа, содержащегося в R5 (в R2 при этом нужно записать копию R5). Регистр R3 здесь является указателем на первый байт используемой в качестве буфера свободной области ОЗУ.

Фактически это прерывание вызывает подряд две подпрограммы. Первая, с адресом @#140600 формирует в буфере текстовую строку из символов цифр, а также устанавливает содержимое регистров R1 и R2 таким образом, что последующая команда EMT 20 выводит десятичное представление числа на экран с текущей позиции курсора. Вторая подпрограмма (@#122560) представляет собой, по сути, вызов функции EMT 20, по каким-то соображениям создателей БЕЙСИК-транслятора вынесенный в виде подпрограммы (весь ее текст состоит из двух команд: EMT 20 и RTS PC).

В пользовательских USR-подпрограммах можно обращаться как к прерыванию TRAP 114, так и непосредственно к подпрограмме @#140600 (не забудьте, что в этом случае для вывода на экран потребуется еще и EMT 20). Дополнительно отметим, что кодом-ограничителем сгенерированной текстовой записи числа является пробел.

- **TRAP 115** — устанавливает R3 на первый не равный пробелу символ исходного листинга, после чего вызывает TRAP 102 и TRAP 120.
- **TRAP 116** — используется при анализе строк, содержащих разделяемые запятыми числовые аргументы. Если R0 перед вызовом равно 0, то подпрограмма возвращает R0=0, когда первый не равный пробелу символ — не запятая, либо запятая, за которой следует еще одна запятая (т. е. следующий аргумент пропущен), либо конец строки. Если же найден очередной числовой аргумент, R0=2 и производится вызов TRAP 100.
- **TRAP 117** — отсчет количества десятичных точек «вправо» от текущей позиции указателя R3. После возврата R2 содержит количество найденных точек, а R3 устанавливается на первый не равный точке символ. Если R2=0 или больше 1, генерируется ошибка 2.
- **TRAP 120** — анализ текста, заключенного в кавычки. *Входные данные:* R3 — указатель на исходный листинг, R1 — указатель на буфер, R0=0. *Результат работы:* если код по адресу R3 не соответствует символу кавычек, то R0=0 и выход. Иначе обрабатывается весь последующий текст до закрывающих кавычек или до кода конца строки. В буфере @R1 создается дескриптор текста в виде: **[число #156156, адрес начала текста, длина строки в байтах]**. Значение R0 устанавливается рав-

ным 2 и в служебных целях в ячейку @#2034 заносится число #177400.

Вспомним, что при трансляции регистр R1 является указателем на очередную ячейку списка шитых кодов, так что данное TRAP-прерывание формирует в этом списке блок данных, соответствующий текстовой строке. Число #156156 — это адрес (шитый код) «сервисной» подпрограммы, переписывающей два следующих за ним машинных слова в стек.

- **TRAP 122** — делает то же, что и TRAP 115, но на время отработки этой функции R3 сохраняется в стеке, т. е. после возврата R3 не будет смещено на первый не равный пробелу символ.
- **TRAP 126** — проверка на запятые. Если код, на который указывает R3, равен коду символа «запятая» («,»), то R3 устанавливается на первый не равный пробелу символ после нее. Иначе выводится сообщение об ошибке 2.
- **TRAP 127** — делает то же, что и TRAP 126, но проверка ведется для кода «открывающаяся скобка».
- **TRAP 130** — то же, но для кода «закрывающаяся скобка».

Ряд других полезных подпрограмм может быть вызван с помощью команды JSR PC,@#<адрес>.

- **@#140600** — в буфере десятичного представления числа, записанного в R5 (см. выше).
- **@#122500** — возвращает в R0 количество знакомест в экранной строке для различных режимов вывода:

— 64 символа в строке — #100,
— 32 символа в строке — #40.

Она может оказаться полезной при написании редакторов текста и т. д.

- **@#122766** — печать на принтере символа, код которого записан в R0. Перед вызовом в R5 необходимо занести число #177714 (адрес порта ввода-вывода БК). Побочный эффект — обнуление ячейки @#2136, если содержавшееся в ней число было больше 100_8. (Этот эффект можно использовать для реализации автопереноса длинных строк при печати, если рассматривать ячейку @#2136 как счетчик, увеличивать ее содержимое на 1 после печати очередного символа, а возвращенное в ней нулевое значение считать флагом для пересылки на принтер кода перевода строки.)
- **@#124076 и @#124120** — соответственно, гасит и включает курсор независимо от текущего режима (в отличие от посылки через EMT 16 кода $\#232_8$, осуществляющего только переключение текущего режима в ему противоположный).
- **@#124136** — реализация оператора INKEY$. Считанный код (или 0) заносится в ячейку @#2420. В вершине стека формируется флаг: 0 — не была нажата ни одна клавиша, 1 — код клавиши находится в ячейке @#2420. (Фактически значение этого флага указывает длину принятой текстовой строки: 1 — один символ, 0 — пустая строка.) Выход из подпрограммы оформлен в виде JMP @(R4)+ .
- **@#124304** — монитор. (Кстати, команда **JMP @#100300** позволяет осуществить выход в монитор без перезаписи (инициализации) векторов прерываний.)
- **@#125226** — по номеру цвета в R0 (от 1 до 4) формирует в R0 код переключения цвета: 221_8—224_8. При R0>4 выдается сообщение об ошибке 5.
- **@#125242** — установка цвета текста и графики по его номеру (запись соответствующего значения в ячейку @#214). Номер цвета задается в R0. При R0=0 устанавливается «прозрачный» цвет, т. е. содержимое @#214 копируется из ячейки @#212. При R0>4 выдается сообщение об ошибке 5. Выход — по JMP @(R4)+ .
- **@#125456** — чертит текущим цветом прямоугольник с диагональю (X1,Y1)—(X2,Y2) или стирает его. *Входные данные:* R0 — флаг (1 — черчение, 0 — стирание), в стек нужно занести (в указанном здесь порядке) значения координат X2, Y2, X1, Y1. Выход по JMP @(R4)+ , стек очищается от заданных значений координат автоматически.
- **@#126274** — возвращает горизонтальную линию (X0,Y)—(X1,Y), не включая крайние точки. *Входные данные:* R0=X1, R1=X0, R2=Y.
- **@#126402** — возвращает в R0 значение цвета точки, координаты которой указаны в регистрах R1 и R2 (R1=X, R2=Y, режим 32 символа в строке). В R0 при этом возвращается не номер цвета, а код его включеиия, как это указано в таблице.

Цвет	Номер цвета	Содержимое R0
Красный	1	221
Зеленый	2	222
Синий	3	223
Черный	4	224

Номер цвета можно определить по содержимому R0 с помощью оператора SUB:

```
JSR PC,@#126402
SUB #220,R0
```

- **@#137710, @#137642, @#137670** и **@#137750** — записывают в ОЗУ начиная с адреса, содержащегося в регистре R1, тексты: «.ASC␣#000↑», «.BIN␣␣␣␣␣␣», «.COD␣␣␣<0><0><0><0><0><0>» и «␣␣<0><0><0><0>» соответственно (<0> обозначает нулевой байт). Эти подпрограммы могут быть использованы для формирования имен файлов в стандарте БЕЙСИКа.
- **@#120606** — восстанавливает программируемые ключи, определяя их стандартным для БЕЙСИКа образом (как после включения питания), и выводит в служебной строке их первые символы.
- **@#120760** — вывод первых символов текущих значений программируемых ключей в служебную строку в БЕЙСИКе (то же можно сделать вручную, включив и тут же выключив режим «ГРАФ», для чего следует дважды нажать «АР2»+«НР»+«5»).
- **@#140064** — чтение файла с магнитофона (с выводом имен встреченных файлов и проверкой контрольной суммы). Перед вызовом нужно подготовить блок параметров в буфере @#320 и занести в ячейку @#2414 адрес входа в используемый драйвер обмена с магнитофоном (по умолчанию БЕЙСИК использует стандартный адрес входа для EMT 36). В R1 нужно также записать 320_8.
- **@#125154** — вывод байта из R0 на линию ИРПС с проверкой готовности линии к передаче.
- **@#125140** — прием байта с линии ИРПС в R0 с ожиданием его поступления.
- **@#137430** — вывод на экран с текущей позиции курсора содержащегося в блоке параметров для EMT 36 имени последнего встреченного файла.

Назовем также некоторые служебные ячейки БЕЙСИКа и расположение его служебных текстов.

- **@#2056** — номер компилируемой строки (после компиляции — номер последней откомпилированной строки). Во время трансляции используется как номер строки с ошибкой.
- **@#2000** — флаг-признак режима работы БЕЙСИКа: #177400 — непосредственный режим (диалог с пользователем), #177777 — режим исполнения программы, #377 — режим трансляции.
- **#146706** — текст «ОШИБКА␣»+CHR$(7) (10 байт).
- **#155160** — текст «В СТРОКЕ␣» (12 байт).
- **#120226** — текст «↵СТОП↵» (6 байт), где «↵» — код перевода строки.
- **#132666** — текст — заголовок вильнюсской версии (34 байта).
- **#132730** — текст «Ok» (4 байта).
- **#147000, #147036, #132734** и **#143262** — таблицы зарезервированных имен БЕЙСИКа. С адреса 133400 также расположена таблица-«связка»: ее ячейки с адресами **133400+<внутренний код оператора>** содержат начальные адреса текстов этих операторов в таблице имен (код-ограничитель каждого из них — пробел).

И напоследок дадим еще один полезный совет. Если данные для подпрограммы передаются через стек, то желательно предусмотреть в ней очистку стека перед возвратом в основную программу. Для этого в ассемблере предусмотрена команда MARK, но как ее применять — известно не всем. Проще (для понимания) использовать последовательность команд **MOV (R6)+,(R6)+** , каждая из которых смещает вершину стека (в которой хранится нужный нам адрес возврата!) на два байта вверх.

Литература

1. *Авсеев В. В., Авсеев А. В.* Особенности транслятора с языка БЕЙСИК для БК-0010.01. //Информатика и образование. 1990. №2. с. 42.

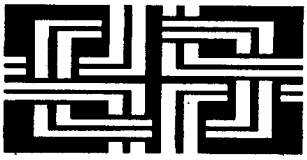
2. *Ермаков В. В.* Многомерные массивы на БЕЙСИКе. //Вычислительная техника и ее применение. 1990. №8. с. 9.

3. *Авсеев В. В., Авсеев А. В.* Использование возможностей БЕЙСИКа в программах на языке ассемблера. //Вычислительная техника и ее применение. 1990. №12. с. 24.

4. *Котов Ю. В.* Совместное использование БЕЙСИКа и машинных кодов. //Вычислительная техника и ее применение. 1992. №3. с. 25.

5. *Усенков Д. Ю.* Реализация многоцветной закраски на БЕЙСИКе. //Информатика и образование. 1993. №4. с. 115. (См. также №2 за 1989, с. 101; №5 за 1993, с. 122.)

6. *Якошвили Д. В.* Приручение «хищников». //Информатика и образование. 1994. №2.с.115



В статье подробно рассказывается о механизме реализации операторов вильнюсского БЕЙСИКа RESTORE, READ, DATA и о том, как реализовать обращение к нужной строке DATA путем указания ее номера в качестве содержимого переменной.

В море данных

Одним из наиболее удобных способов присвоения значений числовых и символьных переменных в вильнюсской версии БЕЙСИКа для БК-0010.01 является механизм RESTORE—READ—DATA, особенно при задании исходных значений ячеек большого массива. (Пример — упорядочивание слов по алфавиту: коды русских букв, как известно, располагаются в знаковой таблице БК не в алфавитном порядке.) На ФОКАЛе для этой цели приходится писать длинные цепочки операторов прямого присваивания, тогда как на БЕЙСИКе достаточно нескольких строк — цикл изменения индекса массива и ряд операторов DATA.

Но есть в этом механизме и недостатки (правда, не очень большие и проявляющиеся, как правило, лишь в отдельных программах). Один из них — невозможность указывать в операторе RESTORE номер требуемой строки DATA с помощью переменной, а не в виде константы, как это единственно допускается в вильнюсском БЕЙСИКе. Какие преимущества могло бы дать использование в данном случае переменной, можно видеть на следующем несложном примере.

Пусть мы составляем программу, опрашивающую пользователя по какой-либо учебной теме, выставляющую ему оценку (скажем, от 1 до 10 баллов) и сопровождающую ее при выводе на экран шутливым текстом: для единицы — «Двоечник!», для пятерки — «Средненько!», для 10 баллов — «Отлично!» и т. п. (По подобной схеме построены практически все обучающие и контролирующие программы.) Реализация блока вывода текста, сопровождающего оценку, обычно делается так:

```
. . . .                  ' опрос и выставление оценки
FOR I=1 TO 10            ' загрузка в массив
READ A$(I)               '  текстов для всех оценок
NEXT
DATA "Двоечник!"         ' ряд строк DATA с комментариями
DATA . . .               '  к оценкам
DATA "Отлично!"
PRINT A$(I)              ' вывод текста для оценки, содержащейся
                         '  в переменной I
```

В этом случае очевидны дополнительные затраты времени и свободной памяти при переписывании данных из списков DATA в массив. Другой способ позволяет обойтись без массива (а значит, и без лишних затрат памяти), но при выводе нескольких текстов подряд требует намного большего времени работы программы:

```
    . . . .
    RESTORE <n>     ' реинициализация списка DATA
    FOR I=1 TO N%   ' N% - оценка
    READ A$         ' A$ - просто символьная переменная,
    NEXT            '  а не элемент массива
<n> DATA. . .       ' список операторов DATA
    PRINT A$
```

Здесь производится «холостое» считывание нескольких первых элементов DATA, пока в переменную A$ не будет загружено требуемое значение. А вот как можно было бы легко и просто реализовать то же самое, если бы в операторе RESTORE допускалось использование переменной, содержащей номер строки (условно считаем, что строки всей программы пронумерованы с шагом 1):

```
      ....
      RESTORE A%+I%   ' A% - номер первой строки с оператором DATA,
                      '  I% - оценка
      READ A$
      PRINT A$
<A%>  DATA . . .      ' список DATA (строка с номером A%)
```

Экономия во времени счета, занимаемой памяти и длине исходного листинга БЕЙСИК-программы очевидна. Попытаемся же реализовать эту идею, научиться самостоятельно «плавать в море данных» DATA.

Стандартная реализация механизма RESTORE—READ—DATA

Опустим подробности использования вышеназванных операторов с точки зрения пользователя БЕЙСИКа (это и так достаточно ясно описано в прилагаемых к БК руководствах) и поговорим о том, как работа этих операторов реализована в трансляторе БЕЙСИКа.

При компиляции исходного листинга БЕЙСИК-транслятор формирует в оперативной памяти, помимо шитых кодов и таблицы соответствия номеров строк и адресов исходного текста и шитого кода, единый список данных из всех имеющихся в программе операторов DATA (именно это позволяет реализовать «автоматический» переход с закончившейся строки DATA на следующую, даже если она находится на несколько десятков строк листинга позже). Правила записи этих данных те же, что и в DATA: символьное представление чисел, текстовые константы (включая кавычки, если они имелись в DATA в исходном листинге), разделяющие отдельные элементы данных запятые и завершающий весь этот список код конца строки &O12. (Здесь нельзя еще раз не пожурить создателей БЕЙСИКа за уже ставшую притчей во языцех неэкономичность транслятора — по сути, нередко громоздкие списки данных DATA хранятся в памяти в двух экземплярах, так что с учетом их дублирования в массив получается уже не двойной, а тройной (!) перерасход памяти.) Дополнительно отметим, что, поскольку операторы DATA в БЕЙСИКе считаются невыполняемыми, шитые коды для них не генерируются.

Адрес очередного элемента в вышеописанном едином списке данных хранится в специально отведенной для этого системной ячейке — указателе &O2040. По завершении трансляции и перед запуском программы на выполнение в ячейку &O2040 заносится адрес начала единого списка, подобное делает и оператор RESTORE. Его шитый код: **157114 <адрес>** , где **<адрес>** — вычисляемый в ходе трансляции адрес в едином списке элемента, который стоял первым в строке DATA с заданным номером. Этот адрес и заносится в ячейку &O2040.

Последний из рассматриваемых операторов, READ, производит чтение данных из единого списка в указанные переменные. При этом, если в ячейке &O2040 записан нуль (т. е. в программе вообще отсутствуют операторы DATA) или по указанному в ней адресу находится код &O12 (список DATA исчерпан), выдается сообщение об ошибке 4. Шитый код оператора READ (при чтении значений целочисленных переменных) выглядит следующим образом:

160634
157122
<адрес шитого кода следующего оператора>
<адрес первой переменной>
<адрес второй переменной>
. . . .
<адрес последней переменной>
<шитый код следующего оператора>

Любопытно, что формат записи и первый из шитых кодов идентичны таковым для оператора INPUT, также служащего для чтения значений переменных, но символьная запись этих значений в случае INPUT содержится в буфере ввода с клавиатуры.

★ ★ ★

Ознакомившись с тем, как работают «стандартные» операторы READ—DATA, можно попытаться реализовать и высказанную ранее идею использования переменного номера строки. Как это можно сделать?

Посмотрим еще раз на описанный выше формат хранения данных в генерируемом БЕЙСИКом едином массиве. Очевидно, что мы могли бы отказаться от использования последнего и с помощью все того же стандартного механизма чтения значений обращаться непосредственно к строкам данных, входящим в состав листинга (в исходных операторах DATA). Для этого необходимо разыскать в памяти начало требуемой строки данных (т. е. DATA в программной строке с нужным номером), записать с помощью POKE найденный адрес в ячейку &O2040 (вместо RESTORE) и вызвать оператор READ. Последний работает как обычно, за исключением лишь одного момента: «автоматического» перехода с исчерпанной строки DATA на следующую производиться не будет, а сообщение об ошибке 4 будет означать окончание текущей строки DATA. Переход же на следующую строку данных должен указываться самим программистом путем записи нужного адреса в ячейку &O2040 (впрочем, если единый список данных из памяти не стерт, можно с помощью **RESTORE <номер строки>** вновь вернуться к стандартному механизму READ).

Теперь осталось обсудить последнюю важную деталь: как определить адрес исходного текста строки с требуемым номером (это может потребоваться и в других случаях, например при реализации программной обработки ошибок — команды ON ERROR GOTO...).

Таблица связи адресов и номеров строк

Как уже упоминалось, при компиляции исходного листинга БЕЙСИК-транслятор формирует в старших адресах ОЗУ таблицу, которая содержит сведения о номерах строк, соответствующих их началу адресах исходного текста и шитых кодов. Подобно стеку, эта таблица «растет сверху вниз» — от старших адресов к младшим. При этом адрес начала таблицы (ее ячейки с наименьшим адресом) содержится в служебной ячейке &O2024 (Будьте внимательны: эту ячейку правильнее было бы называть указателем не на НАЧАЛО таблицы, а на ее ВЕРШИНУ. Роль этой ячейки по смыслу аналогична использованию R6 для стека.) Нам же удобнее будет пользоваться в расчетах именно адресом начала («основания») таблицы. Зная, что вслед за ней всегда идет буфер для обмена данными с магнитофоном, мы можем определить адрес начала таблицы (еще раз

внимание: это указатель на ячейку таблицы с наибольшим адресом) по содержимому ячейки &O2026, где находится адрес начала буфера магнитофона. Очевидно, что тогда адрес начала таблицы равен **PEEK(&O2026)-2** .

Далее, в упомянутой таблице для каждой строки отводится по три двухбайтных машинных слова, первое из которых (отсчет ведется в сторону уменьшения адресов!) содержит номер строки, второе — адрес первого байта этой строки в исходном листинге, а третье — адрес ее первого шитого кода.

* * *

Теперь легко догадаться, как определить адрес начала списка DATA по номеру строки. Для этого можно использовать два способа. Первый заключается в просмотре в цикле всей таблицы соответствия адресов и номеров строк и в сравнении извлекаемых из нее значений с требуемым номером (этот метод использовался при реализации программной обработки ошибок в статье: *Сапегин И. А.* О приручении клавиши «СТОП»... //Информатика и образование. 1993. №4. с.124). Его очевидный недостаток — довольно громоздкий цикл и затраты времени на просмотр всей таблицы. Второй способ — прямое вычисление нужного адреса ячейки таблицы по номеру строки. Сразу оговоримся: это возможно только тогда, когда строки программы на БЕЙСИКе пронумерованы с одинаковым шагом (или если все операторы DATA, к которым предполагается обращение по новому методу, собраны вместе в начале программы и пронумерованы с одинаковым шагом; последующие же строки можно нумеровать как угодно). Договоримся, что шаг нумерации будет всегда равен 10 (при вставке строк с промежуточными номерами нужно выполнить оператор REN без параметров). Тогда легко сообразить, что адрес A% ячейки таблицы связи, в которой содержится адрес начала исходного текста строки с номером N%, можно вычислить по формуле:

A%=(PEEK(&O2026)-4)-(N%/10-1)*6 .

(Аналогично адрес ячейки таблицы с адресом первого шитого кода для данной строки можно вычислить как **(PEEK(&O2026)-6)-(N%/10-1)*6**, это может пригодиться при реализации упомянутой выше программной обработки ошибок.)

Теперь осталась последняя тонкость — найти в выделенном исходном тексте строки DATA с заданным номером начало списка самих данных. Не занимаясь этим вопросом подробно, отметим, что адрес начала списка данных на 1 байт больше, чем найденный по таблице адрес начала программной строки (этот первый байт — внутренний код оператора DATA, равный &O100).

И вот наконец, после долгих трудов по взращиванию «древа знаний», пришло время попробовать его плоды. Запишем всего два (!) оператора БЕЙСИКа, реализующих вожделенный аналог RESTORE с переменным номером строки:

```
A%=PEEK((PEEK(&O2026)-4)-(N%/10-1)*6)+1
POKE &O2040,A%
```

В первой строке производится вычисление адреса начала списка данных в операторе DATA в программной строке с номером N% (обращение «с двойной косвенностью» и с учетом байта, хранящего внутренний код DATA; заметим также, что константа 10, на которую делится N%, является шагом нумерации строк). Вторая строка — запись вычисленного адреса в ячейку-указатель &O2040. Следующим должен быть оператор READ...

Возможно, кому-то из читателей покажется мало этих двух строк после стольких теоретических рассуждений. Но, как говаривал небезызвестному Шерлоку Холмсу его простоватый друг Ватсон, «все это просто и понятно — после того, как Вы все мне объяснили». А напоследок приведем несложный пример — программу, запрашивающую у пользователя цифру от 0 до 9 и выводящую на экран ее русское и английское названия. (Внимание! Строка 10 должна обязательно присутствовать в листинге.)

```
10 ' Названия цифр
20 DATA "Нуль","Zero"
30 DATA "Один","One"
40 DATA "Два","Two"
50 DATA "Три","Three"
60 DATA "Четыре","Four"
70 DATA "Пять","Five"
80 DATA "Шесть","Six"
90 DATA "Семь","Seven"
100 DATA "Восемь","Eight"
110 DATA "Девять","Nine"
120 CLS
130 INPUT "Введите цифру:";N0%
140 N%=20%+N0%*10% ' вычисляем номер строки
150 A%=PEEK((PEEK(&O2026)-4)-(N%/10-1)*6)+1 ' вычисляем адрес
160 POKE &O2040,A% ' аналог RESTORE N%
170 READ RUS$,ENG$
180 ? N0%;": ";RUS$;", ";ENG$
190 ?
200 ' ожидание нажатия клавиши, пробел - конец работы программы, любая
    другая - повтор
210 K$=INKEY$
220 IF K$="" GOTO 210
230 IF K$=" " GOTO 250
240 GOTO 130
250 RESTORE 290 ' возврат к стандартному
    механизму READ-DATA
260 READ X%,Y%,R%,C%
270 CIRCLE (X%,Y%),R%,C%
280 END
290 DATA 100,100
300 DATA 20,2
```

В строке 250 показан возврат к стандартному механизму реализации READ—DATA (с «автоматическим» переходом от строки 290 к строке 300). Если же подобный возврат вообще не предполагается, можно использовать область единого списка (он начинается сразу после шитых кодов и следует до байта &O12) для других целей, например для размещения USR-подпрограмм.

HARD & SOFT

Вниманию читателей предлагается описание новых аппаратных разработок для БК_0011М: жесткого (винчестерского) диска и мультирежимных контроллеров дисководов с дополнительным ОЗУ от 16 до 128 кб. На сегодня имеется уже несколько вариантов подключения винчестеров к БК, и выбор из них наиболее подходящего редакция предоставляет самим БКманам.

В. Е. Новак,
Москва

Новинки аппаратного обеспечения для БК

Прежде чем углубиться в традиционное перечисление новых разработок и их преимуществ, хотелось бы отметить ряд важных моментов, которые оказывают существенное влияние на судьбу компьютерных разработок и, поэтому, имеют большое значение для пользователей БК при выборе тех или иных аппаратных расширений своего компьютера.

В первую очередь пользователь заинтересован в обилии и низкой стоимости программного обеспечения, поддерживающего приобретаемое им устройство. Это пожелание легче исполняется в тех случаях, когда изделие претендует на массовое распространение (иначе кто же будет писать программы?), поэтому, предпочтительнее те разработки, которые функционируют и на БК-0011М, и на БК-0010(.01), ведь в большинстве районов нашей страны БК0011М практически не представлена, да и по стоимости она никогда не будет столь доступной, как БК0010(-01).

Кроме того, при оценке перспективности разработки необходимо принимать во внимание следующие факторы:

- изделие должно быть совместимо со всеми остальными широко распространенными устройствами и со всеми существующими программами;
- на наш взгляд, должна быть предусмотрена возможность подключения дополнительных устройств через внешние разъемы компьютера. Ориентация на установку внутрь БК сдерживает распространение изделия за пределами одного города и создает некоторые неудобства в связи с нарушением принципа модульности.

Отметим, что далеко не все разработки последних лет удовлетворяют приведенным требованиям. Скорее даже наоборот: складывается впечатление, что авторы ориентировались не на запросы пользователей, а на то, «как проще и быстрее сделать для себя». Судите сами: самарский контроллер винчестера устанавливается только внутрь компьютера и работает только на БК-0011М, то же и с музыкальным сопроцессором включающим в себя несколько микросхем, припаиваемых проводками друг к другу и к разным точкам платы компьютера.

Рассмотрим разработки 1994—1995 гг., выполненные с учетом перечисленных в этой статье требований.

Контроллер винчестера для БК-0010(.01) и БК-0011М

Подключение винчестера к БК, к чему многие ранее относились скептически, сегодня стало актуальным благодаря постоянному снижению валютных цен на комплектующие для IBM-совместимой техники (полная стоимость установки винчестера в фирме «Альт-Про» составляет 95—140 долларов в зависимости от модели).

Для подключения к БК используются современные IDE-винчестеры (которые установлены на большинстве IBM), отличающиеся низким уровнем шума, малым энергопотреблением, удобным интерфейсом и небольшими габаритами. Их размеры таковы, что для некоторых моделей возможна установка внутрь стандартного корпуса КНГМД вместе с контроллерами дисковода и винчестера.

Что же дает подключение винчестера?

- В первую очередь, удобство работы с архивами. На одном накопителе может храниться информация с 50 и более дискет.
- Хранение информации на винчестере абсолютно надежно — вы забудете о проблемах с испорченными дискетами и файлами.
- Обмен информацией производится молниеносно и соответствует скорости передачи данных между ячейками оперативной памяти, так как внутри винчестера имеется собственный буфер ОЗУ емкостью 8—32 Кб.

С точки зрения пользователя БК винчестер представляет собой как бы комплект дополнительных дисководов, количество (до 120) и емкость которых (до 32 Мб и более) задается пользователем. Обращение к ним производится так же, как к обычным дисководам, путем указания номера привода — C:, D:, E:, F:, G: и т. д., поэтому никаких проблем с совместимостью программ не возникает.

Контроллер винчестера «АльтПро» выпускается в двух конструктивных исполнениях: либо в отдельном корпусе, как и блок КНГМД (в этом случае необходимо использовать разветвитель МПИ и отключить ПЗУ в контроллере дисковода), либо устанавливается «вторым этажом» в контроллер дисковода. На мой взгляд последиее решение проще и логичнее, ибо эти два устройства тесно взаимосвязаны, да и на других компьютерах (например IBM) практикуется изготовление контроллеров дисковода и винчестера единым блоком.

Контроллер винчестера «АльтПро» может сосуществовать с заводским КНГМД для БК-0011М и с любыми контроллерами дисковода для БК-0010(.01) фирмы «АльтПро». Во всех вариантах для работы с винчестером выделяется 3.5 Кб дополиительного ОЗУ в нигде до сих пор не задействованной области адресного пространства 170000—177000. Наличие дополнительного ОЗУ в контроллере винчестера желательно по следующим причинам.

- Для обращения к дискам винчестера (C:, D:, E:, F:,...) необходимо хранить их координаты в таблице, которая, как и на IBM, называется таблицей разделов. Если дополнительного ОЗУ нет (как в самарском контроллере), то чтением таблицы разделов с нулевой дорожки винчестера предваряется выполнение любой операции ввода-вывода. В контроллере же «АльтПро» таблица разделов и ее контрольная сумма всегда находятся в памяти, что ускоряет работу.
- Последние версии дисковых операционных систем полностью занимают всю отпущенную им память 120000—160000, наличие же ДОЗУ 170000—177000 может значительно облегчить задачу расширения дисковых ОС дополнительными функциями для работы с винчестером;
- И наконец, в прошивке ПЗУ фирмы «АльтПро» (с адреса 160000) предусмотрены гибкие возможности частичной или полной замены драйверов ПЗУ (дискового, винчестерного и арифметического) на пользовательские, размещаемые в ДОЗУ-170000. Данная функция в ряде случаев оказывается незаменимой.

Мультирежимные контроллеры дисковода с ДОЗУ 16, 64 и 128 Кб

Контроллеры дисковода для БК-0010(.01) с дополнительной памятью 16, 64 и 128 Кб, выпускаемые в настоящее время фирмой «АльтПро», монтируются на одинаковых печатных платах, поэтому память в них может легко наращиваться по запросу пользователя. Опознавательные признаки контроллеров «АльтПро» — слово «мультирежимный» в названии, номера модели, начинающегося с буквы «А» (А16, А64, А128), и этикетки фирмы-изготовителя.

Отдельно расшифруем смысл названия «мультирежимный». У всех контроллеров «АльтПро» имеется не менее восьми режимов работы, отличающихся распределением памяти и разрешением доступа к ней по записи или чтению. В частности, возможно подключение ОЗУ вместо монитора БК с адреса 100000 (при этом некоторые экземпляры БК требуют доработки) как в режиме, обычном для ОЗУ, так и с запретом записи (в режиме «квази-ПЗУ»).

Впрочем, подключение ОЗУ вместо ПЗУ монитора — далеко не самое главное и для контроллеров А16 большой ценности не представляет, так как при этом у операционной системы «отбираются» 8 Кб по адресам 140000—160000.

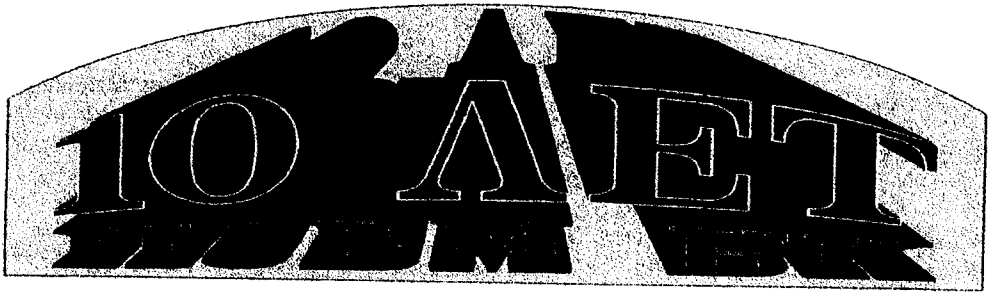
Отдельно следует обсудить следующий принципиальный вопрос: оправдана ли установка памяти свыше 16 Кб, ведь к моменту начала разработки КНГМД моделей А64 и А128 программ, использующих ДОЗУ такого объема, еще не существовало? Оправдана, и вот почему. Благодаря постоянному снижению цен на импортные детали стоимость статического ОЗУ стала более чем доступной. А о значимости преимуществ, которые дает наращивание памяти по схеме фирмы «АльтПро», судите сами.

- В одном из режимов дополнительная память подключается по адресам 100000—177777, так что становится возможным размещение БЕЙСИКа целиком в ОЗУ.
- Наличие 64 или 128 Кб памяти делает возможной адаптацию к БК-0010(.01) части программ, которые ранее работали только на БК-0011М.
- В настоящее время наиболее популярные дисковые ОС исчерпали тот запас памяти, который имелся в ДОЗУ 16 Кб. Дальнейшее развитие этих операционных систем идет по пути создания оверлейной структуры, которая хорошо работает либо с винчестером, либо с виртуальным диском (дополнительной памятью) для оверлеев. Контроллеры А64 и А128 здесь незаменимы. Кроме того, наличие нескольких страниц памяти позволяет загружать существующие программы в ДОЗУ по адресам 120000—160000, предварительно переместив операционную систему в другую страницу — такая процедура давно и успешно практикуется на БК-0011М.
- И наконец, главное: контроллеры А64 и А128 впервые позволили по-настоящему раскрыть на БК возможности создания резидентных программ. Ранее их и размещать было негде (ANDOS, например, использует все 16 Кб дополнительной памяти), и не было на БК прерываний, которые сплошь и рядом не блокировались бы прикладными программами. Теперь же оригинальное решение, не сводящееся к простому добавлению памяти, позволило получить доступ к НЕМАСКИРУЕМОМУ ПРЕРЫВАНИЮ ПО НЕРАЗРУШАЕМОМУ ВЕКТОРУ, вызываемому нажатием кнопки «СТОП» (при этом ее работа «по-старому» тоже доступна и конфликтов не возникает).

Как это используется на практике? Точка входа резидентной программы располагается в ОЗУ между адресами 165000 и 170000, а основная ее часть — в другой странице памяти. При простом нажатии кнопки «СТОП» программно эмулируется ее обычная работа (т.е. управление передается по вектору 4), если же одновременно с ней была нажата какая-нибудь алфавитно-цифровая клавиша, то, либо управление передается на одну из подпрограмм обработки, либо на экран выводится меню выбора функций (с предварительным сохранением фрагмента экрана, используемого под меню, в другой странице памяти). При этом доступны следующие функции.

- Печать или запись в файл копии экрана или его фрагмента. Причем, во-первых, это прерывание не может быть заблокировано прикладной или игровой программой, во-вторых, всегда возможен возврат в прерванную программу и продолжение ее работы и, в-третьих, не нужно устанавливать кнопку «Print Screen».
- Полное сохранение на дискете текущего состояния программы, включая содержимое регистров, в виде файла с блоком запуска. Такая функция интересна возможностью продолжения работы любой программы (игры) с записанного момента.
- В любом графическом редакторе можно использовать функцию «UNDO» (отмена сделанного и возврат к предыдущему шагу). Для этого, конечно, нужно предварительно сохранить предыдущее состояние.

Редакция журнала заключили договор с фирмой «АЛЬТПро», поэтому по вопросам приобретения описанных в данной статье и многих других аппаратных и программных разработок, а также для получения консультаций по подключению периферии, работе с программами и ремонту компьютера вы можете обращаться по телефону (095) 151-19-40 или письменно по адресу 125315, Москва, а/я 17. Для заказа каталога имеющегося программного и аппаратного обеспечения в письмо нужно вложить пустой конверт с марками и вашим обратным адресом.



Поздравляем вас с ДЕСЯТИЛЕТИЕМ БК!

М. Королев, Д. Бутырский,
Москва

Винчестер: подарок к десятилетию БК

Как-то незаметно подкралась эта юбилейная дата, не отмеченная в календарях, не обласканная вниманием власть имущих и многочисленных представителей прессы. Десять лет назад, в 1985 году начался выпуск самой первой модели БК-0010 (для сравнения: первая IBM PC поступила в продажу в отечественную торговую сеть только в 1987 году). Первая БКшка была оснащена 16-разрядным процессором и «пленочной» клавиатурой, работала с магнитофоном в качестве накопителя информации, имела ОЗУ 32 кб (из них 16 кб — экранная память) и язык ФОКАЛ, «прошитый» в ПЗУ. IBM PC же имела только 8-разрядный процессор, память 64 кб, два 40-дорожечных дисковода и операционную систему, лишь отдаленно напоминающую MS-DOS. Как развивались события дальше, знают все. Но задумывались ли вы, почему столь мощная (по тем временам) машина не получила такого распространения как PC? В этом сыграло свою роль и отсутствие рекламы заводов-изготовителей, и их нежелание заниматься разработкой и организацией распространения программных и аппаратных средств для БК, и недостаток информации об этих средствах и о самом компьютере у пользователей (в том числе потенциальных). Программы писали энтузиасты, часто не имеющие даже необходимой документации, не знающие о уже существующем программном обеспечении других авторов и вынужденные из-за этого повторно «изобретать велосипед». А о пользователях PC побеспокоились такие гиганты как MicroSoft, Borland и др. Да и 8-разрядный «Спектрум» так популярен в России только потому, что написанием программного обеспечения для него занимаются профессионалы за границей, а наши «спектрумисты» получают его в основном пиратским способом.

Задумайтесь: за целых десять лет существования БК известно лишь несколько профессионально написанных и распространяемых цивилизованно (а не путем «свободного обмена») программных продуктов, необходимых каждому пользователю. И к сожалению, их количество (и качество) не увеличивается с должной быстротой. Причина этого всем ясна: НИКТО НЕ ОБЪЯСНИЛ ПОЛЬЗОВАТЕЛЯМ, ЧТО НАПИСАНИЕ ДЕЙСТВИТЕЛЬНО ХОРОШЕЙ И НУЖНОЙ ИМ ПРОГРАММЫ ОТНИМАЕТ У ПРОГРАММИСТА МНОГО СИЛ, ЭНЕРГИИ, ВРЕМЕНИ И ЗДОРОВЬЯ. ЭТОТ АДСКИЙ ТРУД НЕОБХОДИМО ОПЛАЧИВАТЬ, ПРИОБРЕТАЯ ПРОГРАММЫ ТОЛЬКО НЕПОСРЕДСТВЕННО У АВТОРА ИЛИ

У ОРГАНИЗАЦИЙ, ИМЕЮЩИХ ДОГОВОР С АВТОРОМ И ВЫПЛАЧИВАЮЩИХ ЕМУ СООТВЕТСТВУЮЩЕЕ ВОЗНАГРАЖДЕНИЕ (увы, такие организации можно пересчитать на пальцах одной руки).

Авторы, в свою очередь, стараются защитить свои детища каким-либо образом от пиратского распространения (ставят защиты на игры, снабжают дискеты фирменными наклейками и т. д.). Но это довольно бесполезное занятие, ибо обязательно найдется «доброжелатель», который снимет защиту, скопирует авторские дискеты и предложит пользователям по цене в 5—6 раз ниже авторской (зачем писать свои программы, когда можно украсть чужие?). Однако и это еще не все. Пиратство приобретает угрожающие масштабы, когда украденная программа попадает в «клубы БК», где она копируется всем их членам. Все это отнюдь не стимулирует авторов к написанию новых программ. Но теперь появился выход из сложившейся ситуации!

Что такое винчестер

Накопитель на жестких магнитных дисках (винчестер) представляет собой маленькую коробочку размером с пачку сигарет (имеются в виду двухдюймовые IDE-винчестеры, которые устанавливаются в портативных компьютерах NoteBook), при этом на него можно записать 20, 40, 80 и более (в зависимости от типа винчестера) мегабайт информации. (Для сравнения: емкость обычной дискеты БК составляет 0.8 Мб.) Скорость обмена с таким накопителем превосходит быстроту работы с «электронным диском» на БК-0011(М). При работе с винчестером никогда не возникает «дисковых ошибок». Кроме этого, двухдюймовые винчестеры потребляют ток не более 0.5 А, почти бесшумны и практически не боятся механических перегрузок.

Контроллер IDE-винчестера впервые был разработан и опробован на БК в 1993 году Алексеем Луговым (г. Самара). К настоящему времени разработана компактная схема контроллера, плату которого удалось уменьшить до размеров платы БЕЙСИКа-ПЗУ для БК-0011(М).

Все это позволяет разместить в стандартном корпусе БК-0011М винчестер, контроллер винчестера, контроллер дисковода и музыкальный сопроцессор! Получается мощный, компактный и надежный компьютер, по удобству работы превосходящий любые другие ПЭВМ, даже более высокого, по общепринятым оценкам, класса.

Драйвер винчестера (программа, позволяющая читать и записывать информацию на винчестере) располагается в ПЗУ контроллера дисководов (326-я прошивка) вместе с драйвером дисковода и программой обработки команд плавающей арифметики. При продаже винчестер разбивается на логические диски (размером от 10 кб до 32 Мб каждый, по желанию пользователя) с помощью специальной программы. Доступ к логическим дискам винчестера осуществляется так же, как и к дисководам: путем указания номера привода в блоке параметров дисковода, т. е. при работе в любой операционной системе устройствами A: и B: будут дисководы, а C:, D:, E:, F:, G: и далее — логические диски.

Кроме удобств для пользователя, БК с винчестером открывает новые возможности для программистов. Жалобы на недостаток памяти БК уже не актуальны, ибо можно писать программы, состоящие из большого количества модулей, подгружаемых по мере необходимости. Но самое главное — появляется возможность ЗАЩИЩАТЬ программные продукты от несанкционированного копирования без каких-либо неудобств для честного пользователя, так как каждый винчестер имеет свой оригинальный серийный номер, выдаваемый только при записи в регистры контроллера определенной команды. Пользователь, имеющий БК с винчестером, сам заинтересован в приобретении только авторских программ. В противном случае он рискует, например, из-за некачественной программы потерять базу данных, которую составлял в течение года...

Операционная система

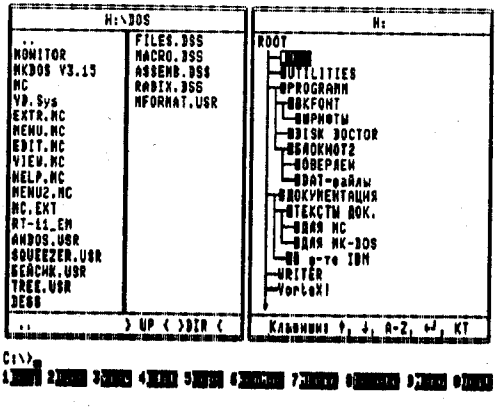
Итак, вы приобрели БК с винчестером. Как использовать все его дисковое пространство? Какую операционную систему установить?

Система ANDOS способна обслуживать устройства емкостью до 1600 блоков (800 кб), так как размер каталога и объем FAT в ней фиксированы (размер таблицы FAT в ANDOS позволяет хранить информацию о 400 кластерах, но реально еще меньше, так как длина каталога не позволяет записывать на диск больше 112 файлов, а файл длиной два байта на самом деле будет занимать минимум один кластер, или 2 кб). Получается, что 20-мегабайтный винчестер придется разбить на 25 «дискет» с именами приводов от C: до [: (квадратная скобка). А если взять винчестер емкостью 40, 80, 120 Мб и более? Букв алфавита явно не хватит (не говоря уже о том, что вы сами скоро забудете, что записали, например, на диск Щ:). Что же делать?

Знакомьтесь: MK-DOS v3.15

У дисковой операционной системы MKDOS каталог с последовательным расположением файлов, что обеспечивает повышенную надежность, оперативность, высокую скорость доступа к информации и позволяет без проблем восстанавливать утраченную информацию (что делает эту систему привлекательнее других, имеющих возможность фрагментации файлов по таблице FAT). При удалении файла в каталоге образуется «дырка», а занимаемая ранее файлом область включается в общий объем свободного места на диске. При записи новых файлов MK-DOS ищет свободную область подходящего размера и записывает туда файл. Если же он достаточно мал, область перед записью файла дробится на две, позволяя использовать оставшееся свободное место в дальнейшем (в отличие от NORD, MicroDOS и других систем).

Система MK-DOS v3.15 и сервисная оболочка MicroCommander v3.00 работают как с минимальной конфигурацией компьютера (БК-0010(.01), контроллер дисковода с 8 или 16 кб дополнительного ОЗУ и один дисковод), так и с максимальной (БК-0011М с винчестером емкостью до двух гигабайт) благодаря модульному построению системы. При загрузке MK-DOS сама определяет, на какой машине ей предстоит работать, и устанавливает максимально возможную и удобную для пользователя комбинацию модулей. На БК-0010(.01), оснащенной контроллером с 16 кб дополнительного ОЗУ, туда загружается резидент MK-DOS и сервисная оболочка. Если есть только 8 кб дополнительного ОЗУ, то оболочка всегда будет подгружаться с дискеты в основную память (с адреса 1000). На БК-0011(М) без винчестера дополнительно подгружаются МОНИТОР БК-0010 и драйвер «электронного диска», а сервисная оболочка «прячется» в седьмую страницу ОЗУ. На БК-0011М с винчестером выводится марка винчестера и его параметры, а «электронный диск» не грузится (впрочем, он и не нужен). Отличительной особенностью MK-DOS является возможность «возрождаться»: если какая-либо программа испортит тот или иной модуль системы, производится его автоматическое подчитывание с системного диска.



В версии 3.15 по аналогии с RT-11 введены такие понятия как «системный диск» (привод, с которого производится загрузка) и «привод для оверлеев» (на БК-0011М им может быть, в частности, «электронный диск»). К ним допускается доступ не только из оболочки, но и из

любой программы, имеющей функции чтения-записи файлов, по специальным именам: @: (системное устройство) и !: (привод для оверлеев).

Новая версия операционной системы MK-DOS принципиально отличается от предыдущих возможностью работы с устройствами объемом до 32 Мб и с логическими дисками до 16 Мб, создаваемыми на этих устройствах (не следует путать логические диски винчестера с логическими дисками MK-DOS и RT-11!), что позволяет полностью использовать дисковое пространство винчестера. При работе с дисководом такая организация файловой системы тоже удобна: логические диски можно копировать, удалять и т. п. как обычные файлы большого размера. На логических дисках MK-DOS запускаются и работают все корректно написанные программы, обращающиеся к операционной системе через EMT 36, и все утилиты.

Сервисная оболочка MK-DOS

С точки зрения пользователя самым главным инструментом при работе с любой операционной системой является сервисная оболочка. Оболочка для MK-DOS — MicroCommander v3.00 — отвечает самым высоким требованиям пользователей и программистов. В отличие от версии 2.10 (которая уже успешно разошлась по стране) она не содержит ошибок, более надежна и удобна. В пакете программ MicroCommander v3.00 реализованы следующие дополнительные функции:

- информация о файлах на панелях может выводится в двух режимах — «Full» (в одну колонку) и «Brief» (в две колонки);
- реализована обработка файлов по расширению (как в Norton Commander на IBM) с передачей параметров из командной строки (как в RT-11 или ОС БК v4.1) для одиночной и групповой обработки файлов;
- по ключу «AP2»+«3» («View») возможен как просмотр (в режимах 64 и 80 символов в строке), так и печать на любом принтере текстовых файлов неограниченной длины в кодировке БК и IBM;
- ключом «AP2»+«4» («Edit») вызывается текстовый редактор, с помощью которого можно редактировать текстовые файлы размером до 4 Кб (например, командные файлы);
- используя меню оболочки («AP2»+«9»), можно установить и сохранить на диске ЛЮБЫЕ параметры операционной системы и оболочки, а также инициализировать дискету или логический диск, создать логический диск любого объема, сохранить или восстановить копию корневого каталога дискеты на самой последней дорожке;
- копирование файлов стало более надежным и быстрым, при этом для наглядности производится динамическая индикация процесса. Исключены ошибки, связанные со сменой диска при копировании на одном дисководе, переименовании и других операциях;
- после нажатия клавиш «AP2»+«2» («User») на текущей панели появляется список файлов только с расширением .USR . Вы можете сами включать программы в этот список, просто задав это расширение в имени часто требуемой программы. В комплекте оболочки имеются подгружаемые функции: «Эмулятор ANDOS» (позволяет копировать файлы с дисков ANDOS и запускать программы, хранящиеся на этих дисках), «Дерево подкаталогов и логических дисков» (для быстрого и наглядного поиска файлов), «Commander Link» (для осуществления связи между двумя БК через параллельный порт, что очень удобно при «перекачивании» информации с одной БК с винчестером на другую без использования дискет).

В комплект MK-DOS включены новые версии самых необходимых утилит, в том числе MFORMAT4 (новая версия программы MFORMAT, полюбившейся многим владельцам отечественных дисководов), где реализованы еще более быстрые алгоритмы форматирования и проверки дискет. Упрощена и операция создания нового системного диска. Впрочем, в небольшой статье сложно перечислить все отличия от старых версий. Лучше один раз поработать с MK-DOS v3.15 .

Технический центр «АЛьТЕК»:

НАША СПЕЦИАЛИЗАЦИЯ - РАЗРАБОТКА АППАРАТНЫХ И ПРОГРАММНЫХ СРЕДСТВ ДЛЯ БК-0010(.01), БК-0011(М)

Представляем вам наше последнее достижение — БК-0011М со встроенными внутрь (!) стандартного корпуса БК:

— двухдюймовым IDE-винчестером емкостью от 20 до 210 Мб (по вашим потребностям и возможностям);

— контроллером винчестера, дисковода и музыкальным сопроцессором.

Питание машины +5V, 2А (например, от БП МС 9016).

Ваша информация и программы всегда будут «под рукой», не говоря уже о 100% надежности их хранения (винчестер «не боится» неожиданного отключения питания, «тряски» и проч.). Скорость чтения-записи превосходит скорость работы с «электронным диском» на БК-0011М.

Без проблем подключаются к нашим машинам МОДЕМ (для работы в компьютерных сетях) и КОНТРОЛЛЕР ЛОКАЛЬНОЙ СЕТИ (для работы в КУВТ-86М).

БК-0011М с винчестером является идеальным «сервером» в компьютерном классе, ибо загрузка 12 ученических машин разными программами занимает не более 5 минут. При подключении других расширений дополнительный коннектор не нужен, так как разъем «МПИ» остается свободным.

На винчестере работают любые программы и операционные системы (RT-11, ANDOS, MK-DOS, CSI-DOS и т. д.). Для полного использования возможностей винчестера специально разработана новая версия MK-DOS (v3.15) с комплексом сервисных утилит и программ.

Винчестер — это именно то, чего вам не хватало для плодотворной и комфортной работы на Компьютере БК!

Кроме того, предлагаем приобрести мультирежимные контроллеры дисковода для БК-0010(.01), дисководы, мониторы, принтеры, модемы, фильтры, мягкую клавиатуру, мыши, джойстики и другие периферийные устройства.

Технический центр «АЛьТЕК» сотрудничает с ведущими авторами программного обеспечения. Новейшие программы — издательская система VorteX v4.0, работающая в ANDOS с файлами неограниченной длины, MK-DOS v3.15, Writer (для печати шрифтом «пишущей машинки» на матричных принтерах), ТЕЛЕТЕКСТ (система обеспечения работы студий кабельного телевидения: реклама, объявления и пр.), новые игры, поддерживающие музыкальный сопроцессор, и многое другое вы можете приобрести уже сейчас.

В стадии разработки находятся: пакет программ бухгалтерского учета (ни в чем не уступающий аналогам на IBM), среда пользователя — аналог MS-Windows и профессиональные «электронные таблицы» под нее. На эти программные продукты ведется подписка (нас интересует спрос на вышеперечисленные разработки).

МЫ ЖДЕМ ВАС по адресу: г.Москва, Шмитовский проезд, дом 2а.
Проезд до ст. м. «Улица 1905 года», далее пешком по улице 1905 года в сторону набережной до «Детского Центра Культуры», 2 этаж, компьютерный класс.

Время работы: среда, четверг, пятница с 12:00 до 18:00.

АДРЕС ДЛЯ ПЕРЕПИСКИ: 113162, Москва, а/я 24.

ТЕЛЕФОНЫ : (095) 946-41-47 (Сергей),
375-76-96 (Дмитрий) с 19 до 22.

А. В. Милюков,

г. Сергиев Посад

Перенос текстов на БЕЙСИКе с УКНЦ и IBM на БК

В последние несколько лет наблюдается значительный рост популярности БК, вызванный подключением к нему дисковода. Для индивидуальных пользователей, измученных работой с магнитофоном, это настоящая находка. А для тех, кто пользуется дисковой ОС ANDOS, появился еще один повод для оптимизма: диски БК можно обрабатывать на IBM, т. е. без особых проблем вести обмен текстовыми файлами. Но всегда, имея что-то, хочется большего. Следующий этап — программы. Понятно, что машинные коды так просто использовать нельзя — процессоры разные. Но с исходными текстами нет проблем (точнее, почти нет).

Как известно, БЕЙСИК-БК способен хранить программу в трех форматах: ASCII, COD и BIN. Последние два отпадают в силу своей уникальности, а вот формат ASCII вполне пригоден для обмена. Однако БЕЙСИК-БК записывает текст блоками по 256 байт, каждый со своим именем, — никакого каталога не хватит, чтобы хранить их на диске*! БЕЙСИК-УКНЦ сохраняет программы в аналогичном формате, но на диске (ОС RT11) файл все-таки один. А вот ТурбоБЕЙСИК-IBM создает обычный текстовый файл.

Так что же делать, чтобы перенести на БК текст программы с УКНЦ или IBM? Любым доступным способом перенесите файл на диск ANDOS, в режиме расширенной памяти подайте команду **BLOAD "BI",&O40000,R** , введите имя файла с нужным текстом и после окончания ввода — команду **CSAVE "<имя>"**.

Что при этом происходит? С диска читается ваш файл как фрагмент обычного текста, а затем обработчик EMT меняется так, что при обнаружении запроса транслятора БЕЙСИКа EMT 6 (ожидание ввода клавиши) автоматически посимвольно вводится весь файл. Если возникают ошибки синтаксиса, нехватка памяти и пр., ввод все равно выполняется, как если бы пользователь нажимал на клавиши с закрытыми глазами. После этого вектор EMT восстанавливается и дальнейшая работа не отличается от обычной.

Программа «BI» написана в формате ассемблер-транслятора «МАВР». Начальный адрес выбран равным 1000, но программа перемещаема.

```
; Basic Input (C) Милюков А.В.
BEG:    MOV     R0,-(SP)
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     R4,-(SP)
        MOV     R5,-(SP)
      ; сохранить регистры
        MOV     PC,R5
        ADD     #BUF-1016,R5
      ; адрес для чтения
        JSR     PC,@#100536
      ; чтение с указанием имени
        TSTB    @#321
        BNE     1
      ; ошибка чтения?
        MOV     @#322,ADR
        MOV     @#350,LEN
        MOV     PC,R5
        ADD     #IRQ-1052,R5
      ; задать новый вектор EMT
        MOV     @#30,EXI+2
      ; сохранить старый вектор
        MOV     R5,@#30
    ; восстановить регистры
1:      MOV     (SP)+,R5
        MOV     (SP)+,R4
        MOV     (SP)+,R3
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        RTS     PC
      ; возврат в БЕЙСИК
IRQ:    MOV     R0,-(SP)
      ; обработчик: сохранить R0
        MOV     2(SP),R0
      ; счетчик команд
        MOV     -(R0),R0
```

* Имеется в виду так называемый «дисковый БЕЙСИК» — программно-аппаратная доработка КНГМД, позволяющая работать с дисководом в вильнюсском БЕЙСИКе БК-0010.01. — *Прим. ред.*

```
        ; код предыдущей команды
        CMP     R0,#104006
      ; EMT 6?
        BNE     1
        TST     LEN
      ; конец файла?
        BEQ     2
        DEC     LEN
        MOV     ADR,R0
        INC     ADR
        MOVB    @R0,R0
        BNE     3
      ; код 0 заменить на &012
        MOVB    #12,R0
3:      CMPB    R0,#40
      ; непечатаемые коды
      ; заменить пробелами
        BCC     4
        CMPB    R0,#12
        BEQ     4
        MOVB    #40,R0
4:      TST     (SP)+
        RTI
      ; возврат из прерывания

2:      MOV     EXI+2,@#30
1:      MOV     (SP)+,R0
EXI:    JMP     @#0
      ; возврат на старый
      ; обработчик EMT
ADR:      .#0
LEN:      .#0
BUF:      .#0
```

Следует отметить, что описанный способ не исчерпывает всех возможностей применения данной программы. Вы можете вообще отказаться от ввода текста в редакторе БЕЙСИКа и работать, например, с TEDом или EDALT3M, несравненно более широкие возможности редактирования которых заметно облегчат ваш труд.

Говоря о работе на БЕЙСИКе с дисководом, нужно отметить и еще одно обстоятельство. Вначале автора вполне устраивал штатный драйвер ПЗУ БЕЙСИКа BASIC10, имеющийся в комплекте поставки КНГМД с возможностью работы вильнюсского БЕЙСИКа с диском. Роль этого драйвера сводится к подключению к шине процессора на время выполнения EMT 36 вместо ПЗУ БЕЙСИКа ОЗУ с ANDOS и обратному включению ПЗУ после записи на диск. Делается это стандартным способом — перехватом вектора EMT. (Необходимая аппаратная доработка КНГМД приводится в приложении.)

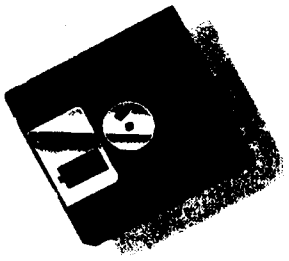
При разработке описанной выше программы «BI» была обнаружена некорректность BASIC10, выражающаяся в том, что всякий возврат из прерывания EMT сопровождался установкой этого вектора на обработчик, имеющийся в составе BASIC10. Иными словами, у этого обработчика «чрезмерная жадность». Квалифицированные программисты знают, что такое цепочка обработчиков прерывания, как надо брать его на себя и корректно отдавать. Если в цепочку встраивается новый обработчик, он обязан придерживаться следующих правил: не «портить» регистры, которые не должны изменяться, и грамотно отдавать управление. В связи со спецификой БК обработчик обычно следит еще и за тем, чтобы у него не отняли управление, для чего после вызова, например, EMT 14, он повторно устанавливает вектор 30. Но при этом он должен указывать не «на себя», как в BASIC10, а НА ПРЕЖНЕЕ МЕСТО. Несоблюдение этих правил ведет к тому, что в системе не могут работать одновременно более двух обработчиков EMT (один в ПЗУ, а второй — «жадный»). Отдавать управление нужно тоже не по принципу «я хозяин вся страна», т. е. в ПЗУ, а ПРЕЖНЕМУ обработчику. Только в этом случае на одном прерывании могут «висеть» несколько обработчиков без влияния друг на друга.

С учетом подобных соображений был разработан драйвер «DBAS», текст которого приведен ниже. После трансляции с адреса 1000 и записи исполняемого модуля нужно изменить, например при помощи UNIC5A, адрес загрузки на 760.

```
; DBAS DiscBasic (C) Милюков А.В.
;  формирование автозапуска
        .#1000
        .#1000
        .#1000
        .#1000
        .#1000
        .#1000
        .#1000
        .#1000
      ;  запомним старый вектор
                MOV   @#30,T+2
      ;  установим новый вектор
                MOV   #IRQ-20,@#30
      ; останов мотора дисковода
                MOV   #14,@#177130
      ;  подключить ПЗУ
                MOV   #40220,@#177716
      ; аналог JMP 120000
                EMT   272
```

```
      IRQ:    MOV   R5,-(R6)
              MOV   2(R6),R5
              MOV   -(R5),R5
; запомним текущий вектор
              MOV   @#30,S+2
; это EMT 36?
              CMP   R5,#104036
              BEQ   IO
; нет, все как обычно
              JSR   R7,@174000(R5)
; вектор вернем на место
        S:    MOV   #0,@#30
              MOV   (R6)+,R5
              RTI

; старый вектор "СТОП"
       IO:    MOV   @#4,SET+2
; и новый
              MOV   #STOP,@#4
; запомним стек
              MOV   R6,STOP+2
; прервано?
              CMPB  (R1),#4
              BCS   8
              DECB  (R1)
; ищем имя SY:
              CMPB  @#330,':'
              BNE   2
; SY
              CMP   @#326,#54523
              BNE   2
; сброс дисковода
              CLR   @#177130
;  выключить ПЗУ
              MOV   #140220,@#177716
; рестарт
              EMT   130
        2:    CMPB  @#327,':'
              BNE   3
              CLR   FLAG
              MOVB  @#326,R5
              BIC   #200,R5
              CMPB  R5,#140
              BCS   11
              SUB   #40,R5
       11:    CMPB  R5,#115
              BNE   9
              COM   FLAG
              BR    5
        9:    CMPB  R5,#104
              BNE   3
        5:    MOVB  #4,1(R1)
              BR    STOP

        3:    TST   FLAG
              BNE   EMU
              MOV   #40000,2(R1)
; DAT
              CMP   @#334,#42056
              BNE   6
              CMP   @#336,#52101
              BNE   6
              MOV   #334,R5
              BR    1

        6:    MOV   #326,R5
              TST   @R5
              BNE   4
              BR    1

        8:    TST   FLAG
              BNE   EMU
; остатки имени
        4:    MOV   #340,R5
; пробелы
        1:    MOV   #20040,(R5)+
              CMP   R5,#346
              BCS   1
              CLR   @#177130
; отключить ПЗУ
              MOV   #140220,@#177716
; старый вектор
        T:    MOV   #100112,@#30
        T:    MOV    #100112,@#30
; старый вектор
              EMT     36
              BR     STOP

      EMU:    JSR     R7,@#116076
;=EMT 36
     STOP:    MOV     #1000,R6
              MOV     #14,@#177130
              MOV
#40220,@#177716  ; подключить ПЗУ
      SET:    MOV     #1000,@#4
              BR      S
     FLAG:    .#0
```

Для не искушенных в программировании читателей подчеркну, что не все константы сохранят свои значения при работе программы, отчасти она самомодифицирующаяся и поэтому некоторые значения могут вызвать удивление. Но все же в BASIC10 странностей было больше, и главное — все, что можно, было написано в косвенной адресации к ПЗУ, т. е. принципиально ни с каким иным ПЗУ драйвер не мог работать корректно.



Несмотря на гибкий и мощный ассемблер и на обилие реализованных на БК-0010.01 загружаемых трансляторов с языков высокого уровня (имеется даже ПАСКАЛЬ), для не очень сложных расчетов часто предпочтительнее использовать БЕЙСИК. Кроме того, благодаря своей простоте этот язык незаменим для обучения начинающих. Но вот вы подключили к своему БК дисковод и... обнаружили, что с зашитым в ПЗУ БЕЙСИКом он не работает. Конечно, существуют «дисковые» версии (загружаемые в ОЗУ), но они обладают значительными недостатками (менее богатый набор операторов, малый объем отводимого пользователю ОЗУ и т.д.).

В настоящее время в продаже появились контроллеры дисководов для БК-0010(.01) нового типа, снабженные не только ДОЗУ на 16 кб, но и возможностью вызова прошитого в ПЗУ вильнюсского БЕЙСИКа, но пока таких новинок немного, особенно на периферии. А как быть тем, кто приобрел КНГМД несколько лет назад, когда дискового БЕЙСИКа не было вообще? Впрочем, если вы не боитесь работать с паяльником и обладаете хотя б минимальным опытом по монтажу радиосхем, не очень сложно самостоятельно доработать контроллер для получения возможности работы на БЕЙСИКе с диском.

С. М. Неробеев, А. В. Сорокин,

Москва

Дисковый БЕЙСИК для БК-0010.01

Очевидно, что при работе любой прикладной программы, в которой не требуется обращение к диску, ни операционная система в дополнительном ОЗУ, ни прошивка драйвера дисковода в ПЗУ КНГМД не используются. То же верно и для программ, работающих с диском время от времени, в промежутках между вызовами файловых операций. Поэтому можно передать управление ПЗУ БЕЙСИКа, предварительно отключив находящиеся по тем же самым адресам 16 кб ДОЗУ с операционной системой и ПЗУ с драйвером дисковода. При этом, хотя ОС и становится недоступной, она не исчезает из ДОЗУ. Когда возникает необходимость чтения-записи файла, специальный диспетчер, резидентно находящийся в памяти, вновь подключает ОС и драйвер дисковода, а после завершения дисковых операций восстанавливает прежнее состояние.

Чтобы программно осуществлять такое переключение, требуется специальный регистр. На больших машинах (например ДВК-4) для этой цели имеется регистр управления памятью. То же самое можно сделать и на БК-0010, однако существует более простой путь. В регистре состояния КНГМД, который находится по адресу 177130, обычно не задействованы второй и третий разряды — выбор накопителя 2 и 3 соответственно (так как к БК-0010 больше двух дисководов не подключают). Этого достаточно не только для организации работы с БЕЙСИКом, но и для формирования еще нескольких страниц ОЗУ.

К сожалению, как и для всех остальных доработок для БК-0010, здесь не существует единого стандарта. Каждый разработчик дискового БЕЙСИКа предлагает свой вариант подключения. В зависимости от того, какие разряды регистра 177130 используются для этой цели, отличаются и конкретные реализации схемы доработки. Например, в схеме А. М. Надежина подключение БЕЙСИКа производится установкой второго и третьего разрядов одновременно, а в схеме П. В. Петрова — только второго. Ниже приведены две схемы: выполненная по стандарту ANCO (она предназначена для работы в операционной системе ANDOS с драйвером DBASIC) и разработанная авторами данной статьи, которая является наиболее универсальной (работает в ОС ANDOS, MKDOS и NORD с соответствующими драйверами).

Для доработки КНГМД нам потребуется микросхема 155ТЛ3 (ее можно заменить на 155ЛА3) и два маломощных кремниевых диода КД522 или 2Д522.

Вначале необходимо произвести доработку самого контроллера дисковода:

- отключить контакты разъема МПИ А29 и А14 от напряжения питания (+5В, подаваемое с контакта А12);
- перерезать дорожки, идущие от контактов В19 и А23.

После этого можно выполнять монтаж согласно принципиальным схемам на рис. 1 и 2. (На рис. 3 показано примерное расположение монтируемых элементов и необходимых соединений.) На схемах в обозначениях типа (xx)Dyy «xx» означает номер вывода микросхемы, а «yy» — позиционное обозначение микросхемы на принципиальной схеме КНГМД. Остается только ввести в БК программу, осуществляющую перехват вектора прерывания @#30 (ассемблер стандарта МИКРО10К), и можно начинать работу с дисковым БЕЙСИКом.

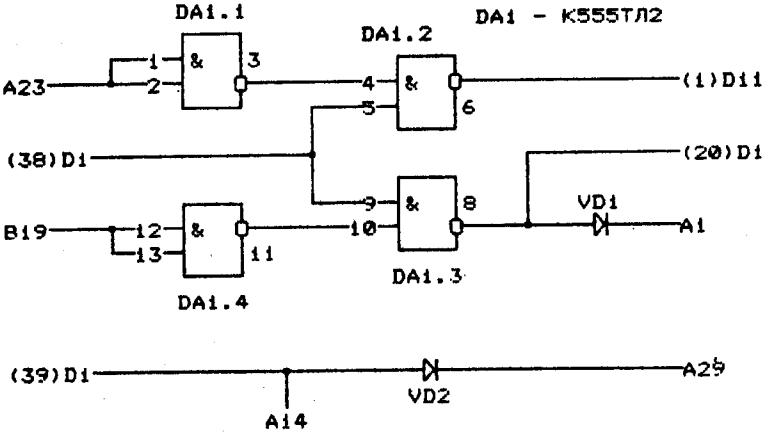


Рис. 1. Вариант схемы по стандарту ANCO

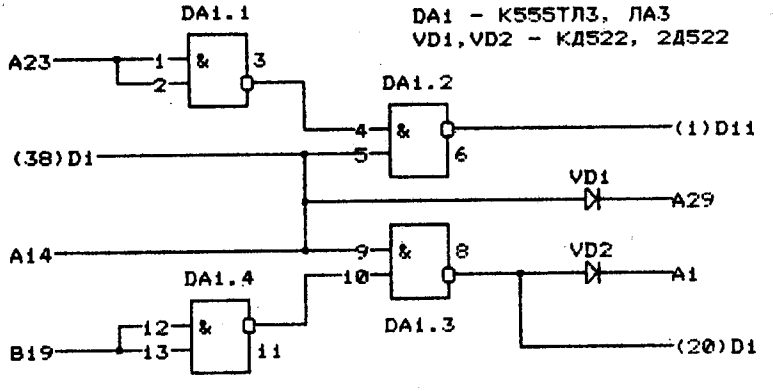


Рис. 2. Вариант схемы, предложенный авторами статьи

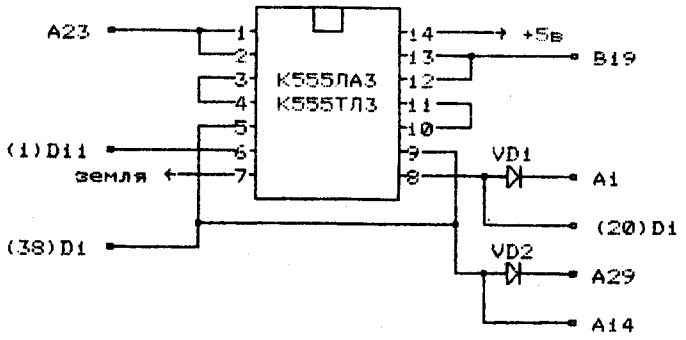
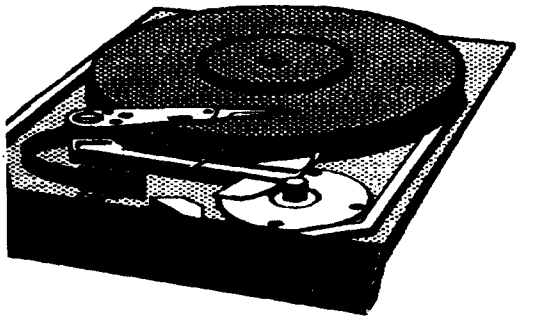


Рис. 3. Размещение элементов и связей

```
;Драйвер для работы с БЕЙСИКом-MSX (Вильнюс-86)
;(версия с корректировкой нулей в имени и обработкой ASC-файлов)
;
;Блок инициализации
        MOV     @#30,S30        ;Сохранение вектора 30. Требуется
                                ;из-за того, что различные
                                ;версии MDOS имеют разные
                                ;адреса перехвата вектора 30
        MOV     #V30,@#30       ;Новый вектор 30
        MOV     #14,@#177130    ;Подключение ПЗУ БЕЙСИКа и
                                ;отключение сигнала DIN на
                                ;контроллере, в результате чего
                                ;блокируется чтение из регистров,
                                ;ПЗУ и ДОЗУ контроллера,
                                ;но сохраняется возможность
                                ;записи в них (с целью подключения
                                ;КНГМД на время записи и
                                ;чтения файлов)
        JMP     @#120000        ;Выход в БЕЙСИК
S30:    .E                      ;Ячейка для сохранения вектора 30

;EMT-диспетчер
V30:    MOV     R5,-(SP)        ;Сохранить R5
        MOV     2(SP),R5        ;Найти адрес команды EMT
        CMP     -(R5),#104036   ;EMT 36?
        BEQ     E36             ;Да
        MOV     (R5),R5         ;Иначе получить команду,
        BIC     #177400,R5      ;выделить ее аргумент,
        ADD     #100000,R5      ;найти адрес обработки по таблице
        CALL    @(R5)+          ;Вызов реализующей п/пр монитора
        MOV     (SP)+,R5        ;Восстановить R5
        MOV     #V30,@#30       ;Восстановить вектор 30
        RTI                     ;Выход из прерывания

;Обработка EMT 36
E36:    CMP     14(R1),'.A'     ;Расширение начинается с "A"?
        BEQ     ASC             ;Да

;Обработка обычных файлов
E63:    JSR     R4,@#110346
        MOV     #20,R2
        ADD     #6,R1
```

```
0B:     MOVB    (R1)+,R3
        BIC     #177400,R3
        CMP     #40,R3
        BMI     0A
        MOVB    #40,-1(R1)
0A:     SOB     R2,0B
        JSR     R4,@#110362
        CLR     @#177130
        MOV     S30,@#30
        EMT     36
        MOV     #14,@#177130
        MOV     #V30,@#30
        MOV     (SP)+,R5
        RTI
```



```
;Обработка ASC-файлов
ASC:    CMP     16(R1),'SC'     ;Расширение "ASC" ?
        BNE     E63             ;Нет
        CMP     @R1,#3          ;Чтение файла?
        BEQ     0RE             ;Да
        CMP     @R1,#2          ;Запись файла?
        BNE     E63             ;Нет

;Запись текстового файла на диск
        TST     ZU
        BEQ     SAV
        CMP     4(R1),#400
        BEQ     SAV
        CLR     ZU
        MOV     (SP)+,R5
        RTI
SAV:    JSR     R4,@#110346     ; Сохранить регистры
        MOV     ADR,R2          ;Перенос
        MOV     @#2026,R1       ;текущего
        MOV     #200,R3         ;блока
0A:     MOV     (R1)+,(R2)+     ;файла
        SOB     R3,0A

;Поиск наличия кода окончания файла
        MOV     ADR,R1          ;Откуда искать
        MOV     #400,R2         ;Сколько байтов просмотреть
0B:     CMPB    (R1)+,#32       ;Найден?
        BEQ     0EX             ;Да
        SOB     R2,0B           ;Иначе продолжать просмотр
        ADD     #400,ADR        ;Читается следующий блок
        BR      IXT             ;Завершение операции

;Запись файла на диск
0EX:    MOV     R1,R5           ;Сохранить адрес последнего символа
        JSR     R4,@#110362     ;Восстановить регистры
        MOV     #40000,2(R1)    ;Адрес всегда 40000
        SUB     #40000,R5       ;Из адреса последнего байта
                                ;получить длину
        DEC     R5              ;Завершающий код 32 удалить
        MOV     R5,4(R1)
        MOV     #40000,ADR      ;Следующий файл - с начала
        INC     ZU
        CLR     20(R1)
        CLR     22(R1)
        CLR     24(R1)
        JMP     E63             ;Запись текущего файла
```

```
ORE:    BR      RIA

;П/пр выхода из промежуточной стадии операции
IXT:    JSR     R4,@#110362
        MOV     @#2026,@#346
        MOV     #400,@#350
        CLRB    1(R1)
        MOV     (SP)+,R5
        RTI

;Чтение файла с диска
RIA:    JSR     R4,@#110346     ;Сохранение регистров
        CMP     ADR,#40000
        BNE     ONE
        MOV     #214,R0
        EMT     16
        MOV     #1330,@#177664
        MOV     '  ',R0         ;Забить лишний текст пробелами
        ADD     #20,R1
        MOV     R0,(R1)+
        MOV     R0,(R1)+
        MOV     R0,(R1)+
        SUB     #26,R1
        MOV     #40000,2(R1)    ;Адрес 40000
        MOV     S30,@#30        ;Считать файл с заданным именем
        CLR     @#177130
        EMT     36
        MOV     #V30,@#30
        MOV     #14,@#177130
        MOV     @#350,R1
OKE:    INC     R1
        BIT     #777,R1
        BNE     OKE
        MOV     R1,DLN          ;Запомнить длину
        MOV     ADR,R2          ;Замена кодов 0 на 12
OKU:    TSTB    (R2)+
        BNE     OKO
        MOVB    #12,-1(R2)
OKO:    SOB     R1,OKU
ONE:    MOV     ADR,R1          ;Перенос
        MOV     @#2026,R2       ;текущего блока
        MOV     #200,R3         ;(400 байт=200 слов)
        MOV     DLN,R0          ;Запись признака окончания файла
        ADD     R1,R0           ;
        MOVB    #32,1(R0)       ;
OB:     MOV     (R1)+,(R2)+
        SOB     R3,OB
        ADD     #400,ADR        ;Увеличить адрес
        SUB     #400,DLN        ;Уменьшить длину
        BPL     IXT             ;Считано еще не все
        MOV     #40000,ADR      ;Иначе подготовиться
                                ;к следующему сеансу обмена
        MOV     #214,R0         ;Выйти из режима РП
        EMT     16
        BR      IXT
ZU:     .E                      ;Признак последующего маркера
ADR:    .#40000                 ;Адрес текущего блока
DLN:    .E                      ;Длина считанного модуля
        END
```

Приводим список статей о компьютерах семейства БК, вышедших в журналае «Информатика и образование» за 1993 и 1994 гг. Перечень публикаций в более ранних выпусках, а также в других источниках см. в журнале «Персональный компьютер» №2 за 1994 г., с. 62. Следует также отметить, что в журнале «Информатика и образование» под рубрикой «Клуб БК» будут публиковаться в основном лишь статьи методического характера.

Материалы о БК, опубликованные в 1993 г.*

№1

Клуб БК

Белозеров О., Добряков В. Ну и что, что мала память! — 80.

Володин Е. БК-стихотворец — 81.

Вормсбехер В., Саяпин А. Опыт рационального решения локальной сети с использованием IBM PC и БК-0011М — 82.

Кузницкий Е., Казарновский К., Кричевский С. Сетевая операционная система для учебных классов КУВТ-86М и УКНЦ-01 — 84.

№2

Разложим звук, взвесим и исчислим

Усенков Д. Музыкальный редактор для Бейсик-Вильнюса БК-0010.01 — 71.

Пименов Г., Пименов Д. Четырехголосный музыкальный редактор на Бейсике для БК-0010.01 (БК-11М) — 74.

Самойлов В. Программирование мелодий на БК-0011М — 79.

Маслов В. Музыка на Форте — 80.

Обмен опытом

Рахманкулов Р. Звуковые эффекты на БК-0010 — 81.

Ивашинников С. О создании музыкального оформления — 83.

Новиков Ф. Подпрограмма в ПЗУ — 86.

Николаев В. Рев БК — 86.

Клуб БК

Усенков Д. От БК-0010 до БК-0011М — 87.

Саяпин А., Вормсбехер В. Знакомьтесь: БК-0011М — 93.

Конюшенко А. Система машинных команд БК-0010 — 96.

Таланов С. БК-0010 и телетайп — 101.

Котов Ю. Простое соединение ПЭВМ типа IBM PC и БК-0010 — 104.

№3

Клуб БК

Булитко В. В. Формульный редактор для БК — 97.

Романов Д. А. «VorteX!» — издательская система для БК — 98.

Надежин А. М. Дисковая операционная система ANDOS — 103.

№4

Клуб БК

Усенков Д. Ю. Реализация многоцветной закраски на Бейсике (по следам одной старой ошибки) — 115.

Сапегин И. А. О приручении клавиши «Стоп»... — 124.

Аскеров Р. Если нет принтера... — 125.

---

* Через тире указаны номера страниц.

№5

Клуб БК

Котов Ю. В. Экономное программирование для БК-0010.01 и БК-0011М — 111.
Андронов И. Л. Ода Форту — 113.

Информация

Принтеры производственного объединения «Радий» — 126.

Материалы о БК, опубликованные в 1994 г.

№1

Языки программирования

Самбиев А. А. Минимизация программ — 86.

№2

Клуб БК

Якошвили Д. В. Приручение «хищников» — 115.

№3

Клуб БК

Саяпин А. А. Сетевая система SPRUT2. Краткое описание — 115.
Разбитной С. А. Геометрическое моделирование на БК — 116.

Нам пишут

БЕЙСИК КУВТ-86: работа с клавиатурой — 126.

В №6 за 1993 г. и №4—6 за 1994 г. статьи о БК не публиковались.

СОДЕРЖАНИЕ

Ю. А. Зальцман **3** МикроЭВМ БК-0010. Архитектура и программирование на языке ассемблера

Д. Ю. Усенков **21** Передача данных из вильнюсского БЕЙСИКа в подпрограммы в машинных кодах

27 Вильнюсский БЕЙСИК, шитые коды и работа с принтером

В. В. Ермаков **29** Многомерные массивы на БЕЙСИКе

30 Справочный листок

Ю. В. Котов **35** Работа на ПЭВМ БК-0010 с программами увеличенного объема

И. В. Канивец **45** Укорочение БЕЙСИК-программ при трансляции

49 Подключение кодовых подпрограмм к БЕЙСИК-программам на этапе трансляции

56 В море данных

HARD & SOFT

В. Е. Новак **61** Новинки аппаратного обеспечения для БК

М. Королев, Д. Бутырский **64** Винчестер: подарок к десятилетию БК

А. В. Милюков **69** Перенос текстов на БЕЙСИКе с УКНЦ и IBM на БК

С. М. Неробеев, А. В. Сорокин **72** Дисковый БЕЙСИК для БК-0010.01

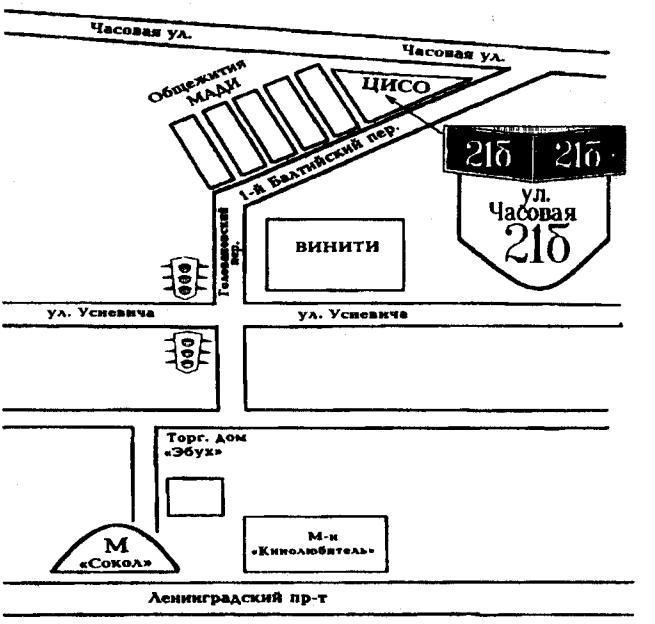
77 Материалы о БК, опубликованные в 1993 г.

78 Материалы о БК, опубликованные в 1994 г.

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР БК-0010 – БК-0011М

Наш адрес: Москва, ул. Часовая, 21Б, помещение Центра Интерактивных Средств Обучения (ЦИСО), комн. 36
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
Лагунова Э. Е.

Подписано в печать с оригинал-макета издательства «Информатика и образование» 14.04.95. Тираж 1500 экз. Формат 70×100 1/16. Бумага офсетная. Усл.печ.л. 6,5. Заказ № 340. . Цена 4000 руб. (по подписке). В розничной продаже цена договорная.
Ордена Трудового Красного Знамени Чеховский полиграфический комбинат Комитета Российской Федерации по печати.
142300, Чехов Московской обл.

ЛИнТех

Хватит мечтать - давайте действовать!
Превратите КУВТ, УКНЦ, "Корвет" и БК в IBM-PC

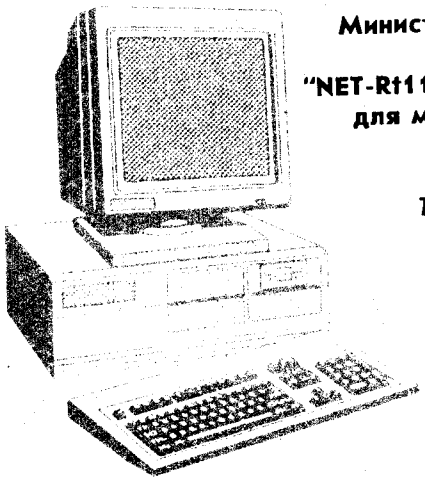
Принципиально новые системы **"NET-R111 & DOS-LINE"** и **"NET-CP/M & DOS-LINE"** позволят Вам превратить КУВТ УКНЦ, "Корвет" и БК в классы IBM-PC. На каждом рабочем месте Вы будете работать, как на IBM-PC, под управлением MS-DOS, использовать Norton Commander, Лексикон, Turbo Basic и другие популярные программы для IBM-PC. При этом полностью сохраняется возможность использования всего существующего программного обеспечения для этих КУВТ.

Для модернизации КУВТ достаточно приобрести нашу систему и установить в КУВТ IBM-совместимый головной компьютер.

Локальные сети **"NET-R111 & DOS-LINE"** и **"NET-CP/M & DOS-LINE"** объединяют с помощью высокоскоростных сетевых адаптеров в единое целое головной компьютер IBM-PC и ученические машины. Скорость работы повышается в 30-100 раз, на каждом компьютере ученика обеспечивается полноценная работа без сбоев и зависаний благодаря отказу от использования стандартного оборудования и дисководов.

Цена систем ниже цены одного IBM-совместимого компьютера. В настоящий момент ими оснащено более 300 компьютерных классов на территории России, Белоруссии, Украины и Казахстана.

Все системы просты в установке и использовании, не требуют перемонтажа существующих линий связи, весь процесс модернизации стандартного класса занимает 2-3 часа. Гарантия - 3 года со дня приобретения.



Министерство образования РФ рекомендует использовать системы "NET-R111 & DOS-LINE" и "NET-CP/M & DOS-LINE" для модернизации КУВТ "Корвет" и УКНЦ.

ЛИнТех
Телефон/факс: (095) 273-50-14
E-mail: shop@lintech.msk.su
119501, Москва, а/я 942