Международная Академия Информатизации

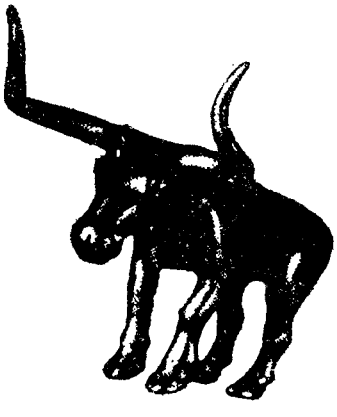
ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР БК-0010 БК-0011м

Компьютика

5'95 (11)

Издается с 1993 г.

В НОМЕРЕ



Издательство Компьютика

Москва 1995

Авторы выпуска

Котов Ю. В.
Надежин А. М.
Неробеев С. М.
Новак В. Е.
Сергеев Б. Н.
Сорокин А. В.
Усенков Д. Ю.
Юров В. В.

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

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

E-Mail: mail@infoobr.msk.su
Телефон: (095) 151-19-40

Продолжаем практическое освоение ассемблера БК-0010(.01), БК-0011(М). Начало статьи и справочные материалы (коды клавиш, перечень команд ассемблера, назначение системных ячеек ОЗУ и т. д.) см. в №4 за 1995 г.

Б. Н. Сергеев,
Москва

Практикум на ассемблере

Команда EMT 6

Напишем следующую программу:

```
EMT 6
EMT 16
.END
```

Запустим ее. На первый взгляд может показаться, что она не работает. Но если теперь нажать на любую символьную клавишу, то на экране напечатается именно этот символ, а ассемблер-транслятор выйдет в диалоговый режим, о чем сообщает приглашение «>».

Вспомним, как работает EMT 16. Для вывода символа на экран она извлекает код этого символа из регистра R0. А команда EMT 6 — своего рода противоположность EMT 16. Она выполняет следующие три основных действия:

- приостанов выполнения нашей программы и ожидание нажатия любой клавиши;
- после нажатия клавиши — запись ее кода в регистр R0;
- продолжение выполнения программы.

Поэтому EMT 6 в нашем примере «заготавливает» код клавиши в R0 для EMT 16. Запустите программу командой RU еще раз и нажмите другую клавишу. Попробуйте напечатать таким образом символы ВСЕХ клавиш. Интересно? Да, но надоедает перед очередным нажатием символьной клавиши каждый раз вводить команду RU. Давайте заставим саму программу выполнять операцию перезапуска с начала!

Команда BR

Итак, мы хотим «научить» программу после вывода символа на экран не заканчивать работу, а возвращаться к команде EMT 6, которая будет ожидать следующего нажатия на клавиши. Для этого можно использовать команду BR МЕТ — «переход к строке с меткой МЕТ». МЕТКА — это короткое имя из латинских заглавных букв и цифр, которое пишется в начале нужной строки и обязательно должно заканчиваться знаком «:» («двоеточие»).

Так как наши программы будут со временем становиться все длиннее и запомнить, что делает та или иная строка, будет все труднее, используем такую возможность ассемблера, как КОММЕНТАРИИ — краткие пояснения к строкам (даже на русском языке). Их пишут прямо в листинге, причем комментарии никак не влияют на работу программы, а нам (и тем, кто потом захочет разобраться в наших творениях) становится легче понять заложенный в нее алгоритм. Комментарий записывается в программной строке ПОСЛЕ мнемоники и всегда* начинается с символа «;» («точка с запятой»).

С учетом сказанного текст нашей программы (ЛИСТИНГ) будет иметь следующий вид. (Для большей наглядности метки расположены по левому краю экрана, а команды — с отступом от левого края на 8 позиций, который получается автоматически при нажатии на клавишу «ТАБ». Между соседними полями должно быть не менее одного пробела.)

* Некоторые трансляторы, например МИКРО, способны распознавать комментарии просто по факту их расположения после ассемблерной мнемоники (т. е. раз команда уже найдена, все дальнейшие символы до перевода строки — комментарий), а начальная «;» требуется, только когда комментарий размещается в отдельной строке, не содержащей ассемблерных команд (что тоже допускается всеми трансляторами). Однако «хороший тон» в программировании, равно как и унификация, требует простановки первого символа «;» во всех случаях. — *Прим. ред.*

```
Поле    Поле     Поле
меток   команд   комментариев

МЕТ:  EMT 6    ; ожидание нажатия на клавишу
      EMT 16   ; вывод символа на экран
      BR МЕТ   ; переход на строку с меткой МЕТ
      .END     ; конец программы

мнемоника
```

Запустите программу и понажимайте на символьные клавиши. Ну как? Теперь работать приятнее?

Пойдем дальше. Добавим еще одну команду EMT 6 так, что теперь наша программа примет вид:

```
МЕТ: EMT 6     ;1
     EMT 16    ;2
     EMT 6     ;3
     BR МЕТ    ;4
     .END      ;5
```

(Строки программы в комментариях пронумерованы по порядку сверху вниз, чтобы удобнее было обсуждать их назначение.)

По сравнению с предыдущим примером нами добавлена строка 3. Здесь EMT 6 используется «в другой ипостаси», на которую ранее вы, возможно, и не обращали внимания. Если в строке 1 команда EMT 6 записывает в регистр R0 код нажатой клавиши и тут же передает его EMT 16 (строка 2), которая печатает на экране символ, то строка 3 нужна для приостанова работы программы и ожидания, пока пользователь нажмет любую клавишу. Нажатие этой клавиши не приводит к печати на экране ее символа, хотя код по-прежнему заносится в R0 (ведь EMT 16 далее отсутствует), а просто продолжает работу со строки 4. Последняя, в свою очередь, передает управление на строку 1, где EMT 6 по сути тоже ЖДЕТ нажатия клавиши, о чем ранее мы как-то не задумывались. Нажав любую клавишу, мы тем самым позволяем команде EMT 6 в строке 1 записать код этой клавиши в регистр R0, причем бывший в нем код клавиши, записанный ранее в строке 3, теряется.

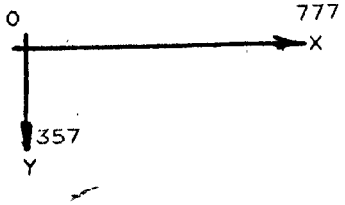
Команда EMT 30

Продолжим освоение ассемблера, перейдя от текста к графике. Напишем программу «Точка», использующую команду EMT 30. (Правила «хорошего тона» в программировании предписывают аккуратное отношение к оформлению своих произведений, одним из элементов которого является название программы. Обычно название помещают в первой строке листинга в виде комментария, написанного с самого начала строки.)

```
; Точка
MOV #1,R0      ; Занесем...
MOV #300,R1    ;  (впрочем, эти три строки
MOV #200,R2    ;  вы можете прокомментировать и сами)
EMT 30         ; нарисовать ТОЧКУ с координатами X=300, Y=200
.END
```

«Нарисовать» — когда в R0 записана единица*, а если вместо нее будет нуль, то это будет означать «стирание».

Координатная сетка на экране имеет следующий вид (в режиме 64 символа в строке X меняется от 0 до 777_8, Y — от 0 до 357_8):



Запустите программу. Нашли ТОЧКУ на экране? Поменяйте координаты (числа в R1 и R2) и попробуйте запустить ее еще раз. Получается?

А теперь убедимся, что запись в регистр R0 единицы или нуля, соответственно, предписывает рисовать или стирать точку на экране. Для этого несколько усовершенствуем нашу программу:

```
        ; "Моргалка"
1:  MOV #1,R0      ; "нарисовать"
    MOV #300,R1
```

* Вообще говоря, вместо единицы в R0 с тем же успехом может стоять любое ненулевое число. Для экономии занимаемой программой памяти вместо требующей четыре байта команды MOV #1,R0 часто используется двухбайтовая MOV PC,R0, так как счетчик команд PC практически всегда содержит ненулевой адрес. — *Прим. ред.*

```
      MOV #200,R2
      EMT 30          ; рисование точки
      CLR R0          ; "стереть"
      MOV #300,R1
      MOV #200,R2
      EMT 30          ; стирание точки
      BR 1            ; возврат в начало
      .END
```

Здесь применена новая для нас команда CLR R0. Ее действие точно такое же, как команды MOV #0,R0, т. е. запись нуля в регистр R0. (Или, как еще говорят, очистка R0.)

Данная программа работает непрерывно, пока не будет выключен компьютер или нажата клавиша «СТОП». Попытайтесь сами доработать эту программу так, чтобы она сама останавливалась через определенное время*.

Команда EMT 32

Добавим в программу «Точка» после EMT 30 фрагмент:

```
MOV #500,R1
MOV #50,R2
EMT 32
```

Что теперь делает наша программа? Усовершенствуйте ее так, чтобы она нарисовала что-нибудь по вашему желанию. Самостоятельно ответьте на вопрос: для чего предназначена команда EMT 32?

Команда SOB

Чтобы рассмотреть процесс рисования «в замедленном показе», добавим в нашу программу «замедлитель»:

```
      ; Линия-2
      MOV #1,R0      ; рисовать
      MOV #777,R1    ; начальная координата X
      MOV #0,R2      ; начальная координата Y
M1:   EMT 30         ; рисование точки
      SUB #2,R1      ; X=X-2 (задание новой координаты X)
      INC R2         ; Y=Y+1 (задание новой координаты Y)
      MOV #7000,R3   ; время задержки = 7000
1:    SOB R3,1       ; задержка вывода на экран
      BR M1
      .END
```

Здесь появилась новая для нас команда SOB. Она работает в паре с предыдущей командой MOV #7000,R3:

```
      MOV #7000,R3 ; время задержки = 7000 "единиц"
1:    SOB R3,1     ; задержка
```

Эти две строки как раз и выполняют роль «замедлителя»**. Алгоритм здесь такой: команда SOB вычитает единицу из числа, записанного (в данном случае) в регистре R3. А в него мы в предыдущей строке поместили число 7000$_8$ (проверьте работу программы при других числах). Вычтя из содержимого регистра единицу, команда SOB проверяет результат. Если еще не нуль, управление передается на метку 1 (в данном случае), т. е. на саму команду SOB. Она опять вычитает единицу из регистра R3, и так до тех пор, пока в регистре не окажется нуль. И только после этого команда SOB передает управление на строку, расположенную ниже. Время, затраченное на цепь вычитаний по единице до нуля, и составляет задержку перехода к следующей строке программы. Соизмерение числа «тактов» задержки с привычной для нас мерой времени (например, с секундой) — дело не простое. Все зависит от нескольких факторов,

* Это задание, возможно, пока по силам не всем читателям, так как команды цикла (SOB) и переходов по условиям (BEQ, BNE и т. д.) будут рассмотрены немного позже. Поэтому данную задачу можно немного облегчить: найдите хотя бы один, а еще лучше — продумайте все возможные способы «заставить» программу останавливаться, скажем, после вывода на экран 100 точек (как бы вы сделали это на БЕЙСИКе?). Пока ограничьтесь только общими идеями, а к их программной реализации можно будет вернуться после изучения соответствующих команд ассемблера. — *Прим. ред.*

** Некоторые временные характеристики исполнения ассемблерных команд приведены в статье Ю. А. Зальцмана в №1 за 1995 г. — *Прим. ред.*

связанных с длительностью выполнения одного цикла «вычитание-сравнение» и плюс к тому — однократного вызова MOV (впрочем, последнее вносит в общую задержку только небольшой вклад). Очень приблизительно можно считать, что время одного цикла задержки составляет не менее одной стотысячной доли секунды. В качестве самостоятельного задания ответьте на такой вопрос: как повлияет установка символа «;» перед меткой 1? Проверьте свой ответ на практике.

Команда CMP

Все бы хорошо, но в нашей программе обнаруживается маленький недостаток — она работает «вечно»! То есть пытается рисовать линию и за пределами экрана, «уводя» ее неведомо куда. Вообще-то это непорядок. «Восстановить справедливость» можно путем ограничения длины рисуемой линии. Зададим условие «рисовать линию не далее координаты Y=300».

```
      ; Линия-3
      MOV #1,R0       ; рисовать
      MOV #777,R1
      MOV #0,R2
M1:   EMT 30          ; рисование точки
      SUB #2,R1       ; X=X-2
      INC R2          ; Y=Y+1
      MOV #7000,R3    ; время задержки
1:    SOB R3,1        ; задержка вывода на экран
      CMP #300,R2     ; координата Y=300 ?
      BEQ END         ; если ДА, то перейти на метку END
      BR M1           ; если НЕТ, то рисовать дальше
END:  .END
```

Теперь программа останавливается, нарисовав линию заданной длины. Это достигается с помощью команды CMP, которая непрерывно «следит» за числом, содержащемся в регистре R2, и сравнивает его с константой 300.

Первоначально в R2 записан нуль (командой MOV #0,R2) — это координата Y первой рисуемой точки (в самой верхней строке экрана). Команда INC R2 с каждым следующим повтором программы прибавляет к числу в R2 по единице, т. е. координата Y с каждым шагом увеличивается. (Напомним, что на экране Y возрастает сверху вниз.)

Команда CMP #300,R2 при каждом повторе программы сравнивает число в регистре R2 с заданной нами константой 300. Это сравнение делается путем вычитания* по формуле «300 минус содержимое R2» и оценки результата: разность равна или не равна нулю (содержимое R2 совпадает с заданным числом 300 или нет). Свою оценку команда CMP записывает в специальную ячейку памяти с названием «Z» (по-английски «Zero» означает «Нуль»).

Команда BEQ

В следующей строке после CMP команда BEQ END отслеживает содержимое ячейки «Z» и как только там окажется единица (т. е. если последняя нарисованная точка имеет координату Y=300), передает управление на метку END, что приводит к окончанию работы программы. Если же в ячейке «Z» нуль, то команда BEQ END игнорируется, а управление передается на расположенную ниже команду BR M1, которая, в свою очередь, вызывает переход на метку M1 для рисования следующей точки.

---

* Точнее было бы сказать, что команда CMP выполняет «псевдовычитание», так как ни первый, ни второй ее аргумент не меняется. Разность же их значений оценивается не только сравнением с нулем, а более многосторонне. Результаты этих оценок заносятся в четыре бита («флага») специального РЕГИСТРА СОСТОЯНИЯ ПРОЦЕССОРА (PSW):
- бит 0 («C») — перенос двоичной «1» из старшего разряда числа (например, при сложении чисел 177777₈ и 1);
- бит 1 («V») — переполнение (устанавливается в «1», если произошла смена знака числа: «0» в старшем разряде изменился на «1» или наоборот);
- бит 2 («Z») — равенство нулю (устанавливается в «1», если число или разность чисел равна нулю);
- бит 3 («N») — знак (устанавливается в «1», если число меньше нуля; фактически бит N является копией старшего (знакового) бита числа).

Подробнее о флагах условий и их использовании читайте в статье Ю. А. Зальцмана (№1, 1993 — №3, 1995). — *Прим. ред.*

Для закрепления полученных знаний предлагаем читателям доработать программу так, чтобы она прекращала рисование линии по другим признакам, например по достижению определенного значения координаты X или по количеству нарисованных точек. Попытайтесь также изменить наклон рисуемой линии. Это можно сделать, меняя приращения координат в строках, отмеченных комментариями «X=X-2» и «Y=Y+1».

А вот еще одна программа в дальнейшее развитие изучаемой темы:

```
M0: MOV #1,R0      ; рисовать
    MOV #777,R1
    MOV #0,R2
M1: EMT 30         ; рисование точки
    SUB #2,R1
    INC R2
    MOV #7000,R3
1:  SOB R3,1       ; замедление
    CMP #300,R2    ; ограничение длины линии
    BEQ M2         ; если Y=300, переход к следующему этапу
    BR M1          ; иначе продолжить рисование
M2: CLR R3         ; стирать
    MOV #777,R1
    MOV #0,R2
2:  EMT 30         ; стирание точки
    SUB #2,R1
    INC R2
    CMP #300,R2
    BEQ M0         ; если стерта вся линия, начать все снова
    BR 2           ; иначе продолжить стирание
    .END
```

Рассматривая этот листинг, мы видим, что некоторые наборы строк повторяются. Возникает мысль: а можно ли избавиться от таких повторов? Ведь, наверное, не зря по свету ходит крылатая фраза: «Любую программу всегда можно сократить хотя бы на одну команду!». Автор предлагает такое решение:

```
R:  MOV #1,R0      ; рисовать
    BR T
S:  CLR R0         ; стирать
T:  MOV #777,R1    ; координаты
    CLR R2         ;   точки
M1: EMT 30         ; рисование или стирание
    SUB #2,R1      ; изменение
    INC R2         ;  координат
    MOV #7000,R3
1:  SOB R3,1       ; замедление
    CMP #300,R2    ; ограничение длины линии
    BEQ M2         ; если Y=300, переход
    BR M1          ; иначе продолжить рисование/стирание
M2: CMP #1,R0      ; R0=1 ?
    BEQ S          ; если ДА, то перейти на стирание
    BR R           ; иначе перейти на рисование
    .END
```

Листинг стал на четыре строки короче! А если подумать еще? Предлагаем читателям самостоятельно поискать другие решения.

Команда EMT 20

Но вернемся опять к выводу текста и познакомимся с командой EMT 20:

```
    MOV #T,R1      ; занести метку T в R1
    MOV #0,R2      ; очистить R2 (можно CLR R2)
    EMT 20         ; вывести слово "Текст" на экран
T:  .ASCIZ/Текст/
    .END
```

Команда EMT 20 выводит текст, который записан в строке с меткой T после .ASCIZ между дробными чертами. Первые две строки нашей программы подготавливают необходимую для EMT 20 информацию. Сама запись .ASCIZ не является мнемоникой ассемблера и называется МАКРОКОМАНДОЙ. Их существует около десятка (см. описание транслятора*). Для вывода текста используется эта макрокоманда и похожая на нее .ASCII:

- .ASCIZ применяют, если написанный после нее текст занимает только одну эту строку и заканчивается нулевым байтом. Об этом напоминает последний символ в слове ASCIZ («Z» = «Zero» = «Нуль»);
- .ASCII используется, если текст не помещается в одной строке и продолжается в следующей, но при выводе на экран обе строки должны «слиться в одну».

Теперь немного улучшим программу «Линия»:

```
    MOV #1,R0       ; нарисовать точку
    MOV #777,R1
    MOV #0,R2
M1: EMT 30
    MOV #4000,R3    ; время задержки = 4000
    SUB #2,R1       ; X=X-2
    INC R2          ; Y=Y+1
    CMP #300,R2     ; ограничение длины линии 300 точками
    BEQ B           ; если вся линия нарисована, то
                    ;  переход на метку B
1:  SOB R3,1        ; иначе задержка (замедление вывода)
    BR M1           ; идти к новой точке
B:  MOV #40,R1 MOV #20,R2 EMT 24   ; установить курсор
    MOV #T,R1 CLR R2 EMT 20        ; выдать текст
T:  .ASCIZ/Задание выполнено./
    .END
```

Здесь в одной строке содержится по несколько команд. Это не ошибка — транслятор Turbo позволяет это делать**.

Генерация звука

Перейдем теперь к программе, которая вызывает звуковой сигнал:

```
MOV #7,R0
EMT 16
......
EMT 16
.END
```

Чем больше команд EMT 16 мы напишем, тем дольше будет слышен звуковой сигнал.

И снова возникает мысль: можно ли упростить написание программы так, чтобы не вводить так много EMTов, а время звучания не уменьшалось? Конечно же, можно! Вот листинг доработанной программы:

```
MET: MOV #7,R0
     EMT 16
     BR MET
     .END
```

---

* Набор макрокоманд зависит от конкретного типа ассемблер-транслятора. Так, в популярном трансляторе M18 вместо указанных двух макрокоманд .ASCII и .ASCIZ используется «обобщенная» .A:. В любом случае список макрокоманд («псевдокоманд») должен иметься в документации к транслятору. Кстати, макрокоманда .ASCII не добавляет в конце текста нулевой байт — об этом должен позаботиться сам программист, добавив после завершающей дробной черты символы <0>. — *Прим. ред.*

** Другие трансляторы, например тот же M18, допускают только одну команду в каждой строке. Поэтому при вводе этого листинга желательно разбить строку с меткой B и следующую за ней на несколько, по одной команде в каждой. Вообще же подобное оформление листинга (кроме разве лишь программ «для собственного пользования») противоречит его унификации и не приветствуется. — *Прим. ред.*

Запустим ее — раздается звуковой сигнал. Чтобы его прекратить, нажмите «СТОП». Опять у нас получилась программа, которая без нажатия на «СТОП» не заканчивает свою работу. Модифицируем ее, как мы делали это раньше. Ограничить время звучания можно так:

```
        MOV #100,R3
            ; пусть будет 100 щелчков
MET:    MOV #7,R0
        EMT 16
        SOB R3,MET
        .END
```

В R3 записываем количество подаваемых «элементарных» звуков (100). Команда SOB каждый раз вычитает из содержимого R3 единицу и проверяет, какое число осталось в R3. Если оно больше нуля, то SOB передает управление на строку с меткой MET, чтобы выдать очередной щелчок. И так до тех пор, пока в R3 не окажется нуль. Тогда SOB разрешит процессору перейти к следующей строке, т. е. к макрокоманде .END .

Чтобы убедиться в том, что длительность звучания действительно составляет 100 отдельных щелчков, попробуйте их сосчитать*. Для этого надо «заставить» программу выдавать щелчки пореже, чтобы был слышен каждый из них в отдельности. Для этого добавим задержку выдачи звуков, переделав нашу программу вот так:

```
        MOV #1000,R4
            ; время задержки = 1000
        MOV #100,R3     ; 100 щелчков
MET:    MOV #7,R0
                ; выдать один щелчок
        EMT 16          ;
1:      SOB R4,1        ; задержка
        SOB R3,MET
; если в R3 еще не 0, то идти к MET
        .END    ; иначе - конец работы
```

В строке с меткой 1 записана уже знакомая нам команда SOB. Она вычитает из R4 (а в начале туда заносится число 1000) по единице и, если в R4 еще не 0, переходит на метку 1, т. е. на саму себя. Хотя электроны бегают очень быстро, им все-таки требуется заметное время, чтобы 1000 раз пробежаться по замкнутому кругу вокруг SOB**. А нам этого и надо — теперь мы слышим отдельные звуки всей трели!

Теперь сосчитали? Действительно 100 щелчков или нет? Некоторые «слушатели» не могут досчитать до 100. А вы? Может быть, это потому, что звуки все-таки выдаются слишком часто? Тогда сделайте их еще более редкими! Для этого замените строку с меткой 1 на две другие:

```
1:  NOP         ; дополнительная задержка
    SOB R4,1    ;  с помощью команды NOP
```

Если даже это не помогло, то не спешите во всем обвинять БКшку. Поменяйте в листинге константу времени задержки #1000 на большую и посмотрите (точнее, послушайте) результат.

В данном случае для хранения чисел мы использовали регистры R3 и R4. Выбирать регистры для своих программ вы должны сами. Только следите, чтобы используемый регистр не содержал уже какую-нибудь важную для дальнейших вычислений информацию. Иначе работа программы где-то позже будет нарушена***.

Попробуйте вместо кода #7 вписать в команду MOV #7,R0 другие числа, желательно из диапазона 41_8—177_8 и 160_8—255_8. Теперь вместо выдачи звука на экране будет печататься повторяемый несколько раз символ, код которого записан вместо числа #7. Действительно ли выводится столько символов, сколько вы задали? Если нет, то в чем дело? А что произойдет, если использовать числа, выходящие за пределы указанного диапазона? Почему? (Ответьте на эти вопросы самостоятельно и проверьте ответы экспериментально.)

* Вообще говоря, устный счет от 1 до 100 относится к «тем еще удовольствиям», памятным всем со школьной скамьи. Не утруждая себя рутиной и оставляя притом все основные алгоритмические принципы «в целости и сохранности», можно заменить в листинге число 100 на 10 и тогда считать придется в десять раз меньше. — *Прим. ред.*

** «Беготня электронов вокруг SOB», конечно же, только образное сравнение. На самом деле имеется в виду тысячекратное повторение внутри процессора пары соответствующих команде SOB «элементарных действий» — «вычесть 1» и «перейти к метке, если не нуль». Суммарные затраты времени на них и составляют задержку. Электроны же (как и цифровые сигналы в цепях компьютера) здесь, в общем-то, ни при чем. — *Прим. ред.*

*** Кроме того, нужно помнить, что регистры R6 (SP) и R7 (PC) служат для специальных целей (как указатели текущей ячейки стека и следующей машинной команды соответственно), поэтому при работе с ними нужно соблюдать определенные правила. Если же пяти доступных пользователю регистров недостаточно, можно хранить данные в ячейках ОЗУ, обращаясь к ним по метке ячейки, или «позаимствовать» временно какой-либо регистр (содержащееся в котором число на данном участке алгоритма не изменяется, а лишь хранится для использования позже), сохранив его содержимое в стеке командой MOV Rn,-(SP) и восстановив после работы с данным регистром командой MOV (SP)+,Rn (Rn в обеих командах — сохраняемый регистр). — *Прим. ред.*

А теперь добавим в программу «Линия» еще одно улучшение: пусть она сообщает звуком о выполнении задания:

```
        MOV #1,R0    ; нарисовать точку
        MOV #777,R1
        MOV #0,R2
M1:     EMT 30
        MOV #4000,R3
             ; время задержки = 4000
        SUB #2,R1                ; X=X-2
        INC R2                   ; Y=Y+1
        CMP #300,R2
             ; ограничение длины линии
        BEQ B
1:      SOB R3,1                 ; задержка
        BR M1
B:      MOV #20,R3          ; 20 щелчков
B2:     MOV #7,R0
        EMT 16
        SOB R3,B2
        .END
```

Тот же способ можно использовать в качестве «будильника», сообщающего, что ожидаемое событие произошло. А если у вас плохо со слухом, то вставьте перед .END такую группу строк:

```
MOV #T,R1
CLR R2
EMT 20
HALT
T: .ASCIZ/Задание выполнено!/
```

Чтобы это сообщение еще и смотрелось красиво, перед приведенным выше фрагментом добавьте строки:

```
MOV #40,R1
MOV #20,R2
EMT 24
```

В качестве «домашнего задания» придумайте, как можно еще улучшить нашу программу.

Какая клавиша нажата?

Ассемблер дает возможность проверять в программе, нажата ли заранее оговоренная клавиша (в данном примере — цифра «1»):

```
N:  EMT 6          ; ожидание нажатия
    CMP #61,R0     ; нажата клавиша "1" ?
                   ;  (сравнение числа 61 - кода символа "1" с кодом в R0)
    BEQ M1         ; если ДА (совпадают), переход к M1
    BR N           ; иначе - в начало
M1: MOV #T,R1      ; вывод текста: "Вы угадали!"
    CLR R2
    EMT 20
    BR N
T:  .ASCIZ/Вы угадали!/
    .END
```

Эту программу, как и предыдущую, можно совершенствовать и облагораживать по своему вкусу до бесконечности, насколько хватит вашего воображения. Например так, как показано ниже. Здесь листинг содержит по нескольку команд в одной строке и отсутствуют комментарии — вам и так должно быть все понятно. Turbo4 допускает такую структуру листинга, это бывает очень удобно, например когда в программе повторяются одинаковые группы команд. При их написании легче обнаружить свои ошибки по несходству строк. А если листинг нужно печатать на принтере, то дополнительно обеспечивается экономия бумаги и удобство чтения*.

```
N:  EMT 6          CMP #61,R0
    BEQ M1
    MOV #T2,R1     CLR R2    EMT 20
    BR N
M1: MOV #T1,R1     CLR R2    EMT 20
    BR N

T1: .ASCIZ<12><234>/Вы
угадали!/<234>
T2: .ASCIZ<12>/Вы ошиблись./
    .END
```

Здесь в макрокомандах .ASCIZ текст между дробными чертами дополнен следующими «управляющими» байтами:

- <12> — переход на новую строку («возврат каретки»);
- <234> — «включение/выключение инверсии символов».

* Напоминаем, что не все трансляторы правильно воспринимают запись нескольких команд в одной строке. — *Прим. ред.*

Команда CLR R2 заменяет прежнюю MOV #0,R2 , число 61 означает код клавиши «1» (см. таблицу кодов клавиш в №4 за 1995 г.), а команда CMP сравнивает источник (код 61) с приемником (содержимое регистра R0).

Организация меню

Во многих программах, в том числе и игровых, часто используется принцип меню — пользователю дается возможность выбирать одно из предусмотренных дальнейших действий из предлагаемого списка вариантов. Например, это может быть выбор способа управления игрой: с клавиатуры, от джойстика или мыши. Напишем простейшее меню* на два варианта:

```
N:  EMT 6 ;ожидание нажатия                                        ;1
    CMP #61,R0   BEQ M1 ;если нажата "1", то перейти к M1          ;2
    CMP #62,R0   BEQ M2 ;если нажата "2", то перейти к M2          ;3
    CAL M3  ;если нажата не "1" и не "2", то перейти к M3          ;4
    BR N                                                           ;5
M1: MOV #T1,R1   CLR R2   EMT 20                                   ;6
    HALT                                                           ;7
M2: MOV #T2,R1   CLR R2   EMT 20                                   ;8
    HALT                                                           ;9
M3: MOV #T3,R1   CLR R2   EMT 20   RET ; подпрограмма              ;10
    HALT                                                           ;11
T1: .ASCIZ<12>/Нажата клавиша "1"/                                 ;12
T2: .ASCIZ<12>/Нажата клавиша "2"/                                 ;13
T3: .ASCIZ<12>/Вы ошиблись!/                                       ;14
    .END
```

По правому краю листинга программы даны порядковые номера строк для ссылки на них при обсуждении алгоритма**.

Строка 1 начинается с метки N (по первой букве русского слова «начало», которое автору как русскому человеку более понятно и приятно, чем иноязычное «start»). Увы, разработчики ассемблеров явно страдают «низкопоклонством перед Западом» и никто не научил БКшку «русскому» ассемблеру, поэтому нам остается только приспосабливаться***. Придется все команды подавать латинским шрифтом: метки M1, M2 и M3 названы от слова «метка», а T1, T2 и T3 — от слова «текст».

---

* Вообще-то в данном случае речь идет только о выборе по нажатию той или иной символьной клавиши (чаще всего такой вариант интерфейса называется механизмом «горячих» клавиш). Настоящее же меню предполагает вывод на экран (например, построчно) всего списка пунктов, выбор любого из них с помощью перемещаемого клавишами-стрелками «курсора» (подсветки) и подтверждение выбора клавишей «ВВОД». — *Прим. ред.*

** При вводе этого листинга в ассемблер-транслятор номера строк лучше всего пропускать, чтобы зря не расходовать ОЗУ на ряды пробелов. Кроме того, в большинстве трансляторов не допускается написание комментариев после макрокоманд для ввода строк текста. Для используемой автором статьи версии Turbo этот запрет редакцией не проверялся, но номера 12, 13 и 14 (вместе с двоеточиями перед ними) в строках с командами .ASCIZ желательно не набирать. — *Прим. ред.*

*** Дело здесь, конечно же, вовсе не во всеобщем преклонении перед иностранщиной. Просто так договорено в компьютерном мире — считать английский язык общепринятой основой для языков программирования. Да и сам БКшный ассемблер был рожден за рубежом вместе с первыми моделями процессоров фирмы DEC, а позже просто «срисован» советскими инженерами вместе со схемотехникой компьютера БК-0010. То же в большинстве случаев верно и для прочих языков программирования. Впрочем, были и попытки «программировать по-русски» (например, команды в некоторых версиях ЛОГО представлены русскими словами). Но не все созданные «русские трансляторы» получили широкое распространение. Кстати, если об «обрусении» языков программирования еще можно спорить, то выводимые на экран БК сообщения готовых программ должны быть русскими (если нет достаточно весомых аргументов в пользу «латиницы»). Пользователя (который, может быть, в школе или вузе изучал немецкий или французский язык) вряд ли обрадует необходимость для работы с программой учить еще и «инглиш». — *Прим. ред.*

Рассмотрим структуру программы по отдельным строкам.

Строка 1 — ожидание нажатия на любую клавишу.

Строка 2 — команда CMP #61,R0 сравнивает число 61 (код клавиши «1») с числом, записанным в R0 командой EMT 6 после нажатия на клавишу. Команда BEQ M1 оценивает результат этого сравнения и:

если оба числа совпадают, то передает управление на метку M1 (строка 6), пропуская строки 3—5. Тогда строка 6 выдает на экран фразу «Нажата клавиша "1"»;

если числа не совпадают, то управление передается на строку 3.

Строка 3 работает аналогично строке 2, но выводит фразу «Нажата клавиша "2"» или передает управление на строку 4.

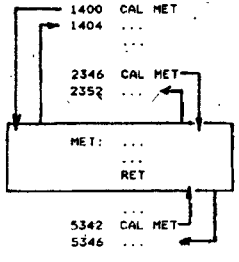
Строка 4 — здесь использована новая для нас команда CAL M3, означающая «перейти на строку с меткой M3 (строка 10) и работать до тех пор, пока не встретится команда RET, а после этого перейти на строку сразу после CAL (на строку 5)»*. Такая цепочка команд, вызываемая по ее начальной метке командой CAL и завершающаяся RET, называется ПОДПРОГРАММОЙ.

Особенность сочетания двух команд CAL и RET состоит в том, что:

они всегда работают в паре;

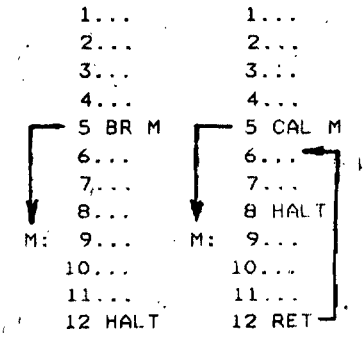
команда CAL MET может стоять В ЛЮБОМ МЕСТЕ программы (в том числе может быть несколько вызовов одной и той же подпрограммы с меткой MET из разных участков листинга);

команда RET всегда ВОЗВРАЩАЕТ управление на строку программы, находящуюся после вызвавшей команды CAL.



Так, при обращении по CAL из строки с адресом 2346 (см. предыдущий рисунок) после выполнения подпрограммы, начинающейся со строки с меткой MET, управление передается на строку 2352. А при обращении к той же подпрограмме из других строк — 1400 и 5342 — управление будет передано соответственно на строки 1404 и 5346.

Две следующие диаграммы показывают отличие команды BR от пары CAL—RET:



По команде BR строки 6, 7 и 8 окажутся просто пропущены, т. е. выполняться никогда не будут. По командам же CAL и RET сначала выполняются строки 1—5, затем 9—12 и наконец 6—8. Участок строк 9—12 может быть вызван на исполнение несколько раз, в том числе из разных мест программы. В нашем примере подпрограмма, начинающаяся от метки M3, вызывается один раз.

Но продолжим разбор алгоритма. Строки 7, 9 и 11 содержат команду HALT. Не слишком ли их много? Уберите хотя бы одну и запустите программу. Ну как? Уберем HALTы или все-таки оставим? Окончательный «приговор» за вами. Однако не зря существует поговорка: «Нет такой программы, которую нельзя было бы укоротить». Попробуйте — вдруг получится?**

* Стандартный ассемблер содержит команды вызова подпрограммы JSR Rn,<метка> и возврата RTS Rn (где Rn — один из регистров). Именно так они записываются в большинстве трансляторов, но в некоторых из них (в частности, в Turbo) для упрощения работы дополнительно предусмотрены команды CAL (или CALL) и RET (или RETURN), фактически означающие JSR PC,... и RTS PC соответственно. — *Прим. ред.*

** Очевидное решение задачи (правда, не уменьшающее длину программы) — добавление в конце строк с метками M1 и M2 команд BR для перехода на строку 11 (соответственно, нужно ей присвоить какую-нибудь метку), тогда HALT в строках 7 и 9 окажутся лишними. А если в конце строк 6 и 8 поставить BR N , то после вывода текстового сообщения вновь будет запрашиваться нажатие клавиши. — *Прим. ред.*

Разобравшись в работе этой программы, вы сможете написать меню на любое количество пунктов.

А вот еще одна интересная программа:

```
N:  EMT 6
    CMP #65,R0    BEQ M1
    CMP #121,R0   BEQ M2
    CAL M3        BR N
M1: MOV #T1,R1    CLR R2    EMT 20      CAL M4    BR N
M2: MOV #T2,R1    CLR R2    EMT 20      CAL M4    BR N
M3: MOW #T3,R1    CLR R2    EMT 20      RET
M4: MOV #20,R1
1:  MOV #7,R0     EMT 16    SOB R1,1    RET
T1: .ASCIZ<12>/Нажата клавиша 1./
T2: .ASCIZ<12>/Нажата клавиша 2./
T3: .ASCIZ<12><233>/Вы ошиблись!/<233>
    .END
```

Здесь показано обращение к подпрограмме из разных точек ОСНОВНОЙ программы (самостоятельно найдите в листинге эту подпрограмму и точки ее вызова). Опробуйте новый вариант в работе. Если вы не сможете по листингу определить «правильные» клавиши, то жмите все подряд, не глядя на экран, пока БК не «закричит» о том, что вы угадали. Попробуйте переделать эту программу так, чтобы она сообщала звуком о вашей ошибке.

Подсчитайте, сколько потребовалось бы строк экрана, чтобы разместить этот листинг по одной команде в строке, как это делают другие ассемблеры. Для этого нужно 30 строк, т. е. увидеть на экране весь листинг целиком невозможно. А Turbo это допускает (спасибо его создателям В. Коренкову и А. Надежину).

Команда EMT 22

До сих пор мы выводили текст только в «рабочую» область экрана. А как поместить его в служебной строке? Это поможет сделать команда EMT 22 (вывод символа в служебную строку). Вот пример ее работы:

```
; Текст в служебной строке
    CLR R0      EMT 22    ; очистить служебную строку
    EMT 6                 ; ждать нажатия любой клавиши
    MOV #40130,@#160      ; адрес начала текста в служебной строке
    MOV #T,R1   CLR R2    EMT 20    ; выдать текст
    HALT
T:  .ASCIZ/Текст в служебной строке./
    .END
```

Вообще-то здесь команда EMT 22 используется лишь частично — для очистки служебной строки от старого содержимого. Поскольку она служит для посимвольного вывода (см. описание EMT в №4 за 1995 г.), что не всегда удобно, мы выводим текст все той же командой EMT 20, «перенаправляя» ее действие на служебную строку с помощью оператора MOV #40130,@#160 — «занести число 40130 в системную ячейку по адресу 160». Чтобы понять его действие, займемся изучением памяти БК. Для этого нам понадобится системная программа под названием «отладчик».

Память БК

Все программы, которые мы загружали с магнитной ленты либо диска (например, ассемблер-транслятор) или писали сами, попадали в память БК. Всего БК имеет 200000 ячеек памяти* (в восьмеричном исчислении). В некоторые из них центральный процессор (ЦП)

* Реальное количество ячеек памяти несколько меньше, так как большинство адресов в диапазоне от 177600 до 177777 в БК ничему не соответствует и только несколько из них указывают на регистры (особые устройства, хотя и похожие по принципу обращения к ним на ячейки памяти, но, строго говоря, таковыми не являющиеся). — *Прим. ред.*

может заносить числа для временного хранения, как мы кладем вещи в отсеки камеры хранения на вокзале. Такие ячейки составляют оперативное запоминающее устройство — ОЗУ.

Отсеки камеры хранения занумерованы порядковыми номерами, которые помогают нам в любое время быстро найти «свою» дверку. Конструкторы ЭВМ пошли по тому же пути и занумеровали все ячейки номерами от 0 (нуль) до 177777. Почему только до 177777? Да просто потому, что других ячеек нет*.

Из всего набора адресов (или, как еще говорят, «адресного пространства») к ОЗУ относятся ячейки с адресами от 0 до 77777, из них мы пока можем безбоязненно использовать только адреса от 1000 до 37777**

Загрузим отладчик OTL16 в ОЗУ в адреса 30000. Его длина 10000, поэтому он займет участок памяти до 37777 включительно. Для написания нашей программы остается фрагмент ОЗУ от 1000 до 30000. Для начала этого более чем достаточно.

Запустим отладчик. В левом верхнем углу экрана появляется строка «000000» — шесть нулей. Будем нажимать на клавишу «ЗАПЯТАЯ». По левому краю экрана в колонку сверху вниз выводятся шестизначные числа. Это и есть номера ячеек памяти. Присмотревшись к ним, видим, что значения чисел увеличиваются сверху вниз, начиная от нуля с шагом 2. Нажимая на «МИНУС», «О» (латинская заглавная буква), «ЗАПЯТУЮ» и «ПОВТОР», можно «погулять» по памяти.

Номера ячеек принято называть АДРЕСАМИ, по сходству с адресами домов на улицах. Перемещаясь по ячейкам памяти, мы видим только четные адреса. Это похоже на то, как занумерованы дома на четной стороне улицы. Но на противоположной стороне дома имеют нечетные номера. В памяти БК тоже есть нечетные адреса, но пока они от нас скрыты. Подобно тому как в туманную погоду противоположная, нечетная сторона улицы может быть не видна, но пешеход твердо знает, что она ЕСТЬ, так и здесь мы видим только четные номера, зная о существовании нечетных. А как их увидеть? Подождем немного, пусть «туман рассеется».

Есть и отличие от нумерации домов на улице. Номер самого первого дома всегда равен 1. В старые времена люди не догадывались о том, что нуль — тоже «нормальное» число. (Древние римляне когда-то занумеровали первый год нашей эры числом 1, а не 0. Отсюда и всемирная путаница — встреча XXI века ожидается всеми в 2000 году, хотя истинная дата — с 31 декабря на 1 января 2001 года!) Создатели вычислительных машин оказались умнее, и с их легкой руки самому первому адресу «приклеен» номер НУЛЬ. Это касается не только адресов ячеек, но верно и в любых других случаях (нумерация секторов на дискете, элементов массивов в БЕЙСИКе и т. п.).

Итак, начнем с нуля. (Вот, оказывается, откуда пошла старинная поговорка — с древних... компьютеров!?) Справа от колонки с адресами видны непонятные надписи. Вспомним школьное правило: перед началом урока надо подготовить классную доску. — очистить ее от прежних записей. Очень полезное правило. Применим его и здесь. Очистим память от «мусора». Для этого наберем такую цепочку команд (в кавычках указаны названия клавиш, а в скобках — примечания для ясности; в виде текста их набирать не надо):

1000A30000□0 ВВОД

0 (буква, а не нуль!) **1000A,** (запятая) ПОВТОР

Теперь во всех ячейках с адресами от 1000 до 27777 включительно записано слово HALT (команда останова), а правее него — колонка с нулями, что говорит об очистке памяти. Можете это проверить: взгляните на содержимое ячейки 30000, подав команду **30000A**. Что вы нашли в ней? HALT? Нет! Там «сидит» что-то другое. Значит, команда 30000□0 ячейку 30000 не

---

* Максимальное значение адреса 1777778 ограничено в БК разрядностью процессора. На других ЭВМ диапазон адресов может быть большим. — *Прим. ред.*

** Адресное пространство БК условно разделяется на следующие зоны:
- 0—777 — системные ячейки;
- 1000—37777 — ОЗУ пользователя;
- 40000—77777 — видеопамять;
- 100000—177577 — ПЗУ;
- 177600—177777 — область системных регистров.

Свободна для любых действий программиста только область 1000—37777 (кроме ячеек, занятых транслятором, отладчиком или другой аналогичной программой), а для работы в других диапазонах адресов нужны дополнительные знания. — *Прим. ред.*

очистила. И правильно! Ведь по адресам с 30000 до 37777 размещается сам отладчик, и портить его никак нельзя! Значит, только участок памяти от 1000 до 27777 очищен от всяких записей и мы в нем можем писать... что-нибудь.

Начнем с адреса 1000 (можно и с другого, но только не меньшего 1000!). Автор предпочитает именно 1000: это такое красивое и круглое число, легко запоминается и просто пишется...

Наберем такую цепочку команд:

```
0 (буква) 1000A "ВВОД" ADD R5,R4 "ВВОД"
```

Отладчик выдает в ответ строку:

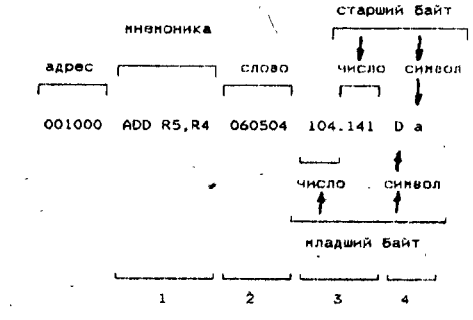
```
001000 ADD R5,R4 060504 104.141 Da
```

Сделаем «грамматический разбор» этого «предложения» (как когда-то делали в школе):

- 001000 — адрес ячейки, в которой сделана запись;
- ADD R5,R4 — мнемоника (словесное представление) команды, записанной нами в ячейку 1000. Ее смысл: копию числа из регистра R5 прибавить к содержимому R4;
- 060504 — числовой КОД записанной команды в виде восьмеричного шестизначного числа, которое называется восьмеричным СЛОВОМ;
- 104.141 — числовой КОД записанной команды в виде двух восьмеричных трехзначных чисел, каждое из которых называется ПОЛУСЛОВОМ или БАЙТОМ. Левый байт называют МЛАДШИМ, а правый — СТАРШИМ;
- D a — символьное (буквенное) представление этих байтов.

Примечание. Слово «восьмеричное» обычно заменяется подстрочным индексом «8» после числа в данной системе счисления (как мы и будем делать далее). Например, фраза «восьмеричное 23» заменяется на 23_8. Аналогично и для других систем счисления: «$_2$» — двоичное, «$_{10}$» — десятичное.

Представим «грамматический разбор» в виде рисунка:



Для вывода этой строки на экран отладчик проделал следующую работу:

- «увидел» мнемонику и преобразовал ее в 16-битное (16-разрядное, но не 16-ричное!) двоичное число, которое, однако, на экран не выдается;
- 16-битное число превратил в восьмеричное слово (060504);
- это слово преобразовал в два числовых байта-полуслова, выводя их через десятичную точку (104.141);
- расшифровал байты и выдал соответствующие им символы (D a).

Итак для любой ячейки памяти отладчик всегда показывает содержимое в четырех видах: мнемоника, слово в восьмеричном виде, два байта-полуслова в восьмеричном виде, два байта-полуслова в символьном виде.

Если мы будем вводить вместо мнемоники восьмеричное число, отладчик все равно «не растеряется» и выдаст точно такую же строку, как и для мнемоники. Для проверки наберите цепочку команд:

```
0 (буква) 1002A "ВВОД" 60504 "ВВОД"
```

В русском языке понятие «полуслово» означает половинку слова. Сложите выданные отладчиком числа 104 и 141 — получилось ли «полное слово» 60504? Нет? Может быть, вы не так складываете? Или их надо просто записать друг за другом без разделительной точки?

Чтобы понять, как слово делится на байты, обсудим этот вопрос подробнее.

Слова и байты

Числа, с которыми работает ЭВМ, могут быть только двоичными 16-разрядными (16-битными), другие форматы представления чисел БК не понимает. Названия же расшифровываются так:

- ДВОИЧНОЕ — потому что каждый разряд может принимать только два значения — нуль («0») или единица («1»);
- 16-БИТНОЕ — потому, что число состоит ВСЕГДА из 16 цифр (нулей или единиц). Оно называется МАШИННЫМ СЛОВОМ, а каждый разряд машинного слова называется термином «БИТ». Следует помнить, что это не имеет никакого отношения к понятию «шестнадцатеричное». (Последний термин мы даже не будем рассматривать, так как в БК такой формат записи чисел не применяется.)

Пример 16-битного двоичного числа (слова): **1001011100101001** . Прочтите его вслух. Если вас это затрудняет, то надо себя заставить: других чисел БКшка не «понимает»*.

Биты в СЛОВЕ нумеруются справа налево от 0 до 15:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

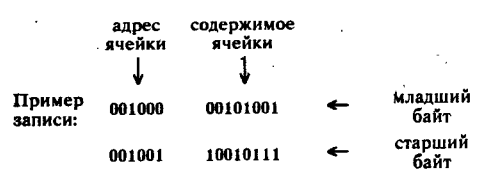
Запишем приведенное выше 16-битное слово под этой «шкалой»:

16 битов (разрядов) = СЛОВО

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	1	1	0	0	1	0	1	0	0	1

Теперь любой из шестнадцати битов слова имеет «личный номер» и мы всегда однозначно можем сказать, о КАКОМ из них идет речь.

Иногда используются половинки слова («полуслова», о которых мы говорили чуть раньше), получающиеся «механическим разрезанием» 16-битного слова посередине. Например, «разрезав» показанное выше число, мы получим «10010111» и «00101001». Каждая такая половинка называется БАЙТОМ, причем левый байт называют СТАРШИМ, а правый — МЛАДШИМ. Соответственно, в любую ячейку ОЗУ можно поместить только одно 8-битное число (один байт). Но процессор БК способен работать и с 16-битными числами — словами. Для хранения такого числа используются две расположенные рядом ячейки. Например, когда правая половина числа (младший байт) записывается в ячейку с адресом 1000, а левая (старший байт) — с адресом 1001:



Процессор редко работает с отдельными байтами, а чаще со словами. Поэтому приходится байты объединять в слова. Проделаем эту операцию с предыдущим примером: напишем старший байт слева, а младший — справа:

адрес	ст. байт	адрес	мл. байт
001001	10010111	001000	00101001

Сгруппируем адреса ячеек слева, а их байты справа:

адрес ст. байта	адрес мл. байта	ст. байт	мл. байт
001001	001000	10010111	00101001

Адресом слова договорились считать адрес младшего байта (четный). Поэтому в отладчике в колонке адресов показаны только адреса младших байтов, являющиеся одновременно и адресами слов:

001000 10010111 00101001

или, если байты соединить в слово:

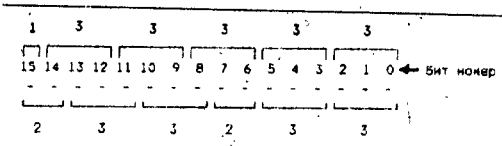
001000 1001011100101001

Любые программы состоят из именно таких чисел — слов. Прочтите приведенное выше слово вслух и попробуйте повторить, не подглядывая. Пользователи первых ЭВМ тоже занимались этим безрадостным делом, пока кто-то из них не догадался упростить это дело для понимания даже начинающими программистами. Идея заключалась в замене 16-битного отображения числа его восьмеричным эквивалентом**. Делается это так. Биты СЛЕВА НАПРАВО группируют:

- для слова — по 1, 3, 3, 3, 3 и 3 бита (верхняя часть рисунка);
- для байтов — по 2, 3, 3 и снова 2, 3, 3 бита (нижняя часть).

* Не унывайте! Ведь существуют программы, которые помогают компьютеру преобразовывать текст или восьмеричные числа в требуемый ему двоичный формат и обратно. Примеры таких программ — те же ассемблер-транслятор и отладчик. — *Прим. ред.*

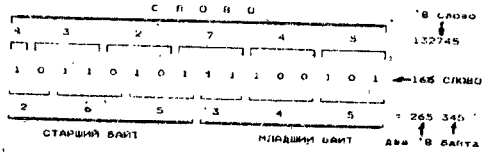
** Замена громоздких двоичных чисел компактными восьмеричными была «первой революцией» в области информатики. «Вторая революция» — переход к текстовым (мнемоническим) обозначениям команд процессора — ассемблеру. «Третья революция» — создание трансляторов с языков «высокого уровня» (БЕЙСИК, ПАСКАЛЬ, ФОРТРАН, СИ и др.), «умеющих» преобразовывать в команды процессора строки листингов, написанные в привычном для пользователя «операторно-формульном» виде. Четвертый «революционный» этап — разработка систем «программирования без программирования» (типа пакета Eureka на IBM, где пользователю достаточно ввести математические формулы и исходные значения, а компьютер сам формирует алгоритм вычислений) и систем на основе искусственного интеллекта. — *Прим. ред.*



Двоичн.	Восьм.
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Каждая тройка битов переводится в восьмеричную цифру по правилу, приведенному в таблице слева. Если «тройка» двоичных цифр неполная, например включающая только два или один бит, то недостающие позиции слева заполняют нулями.

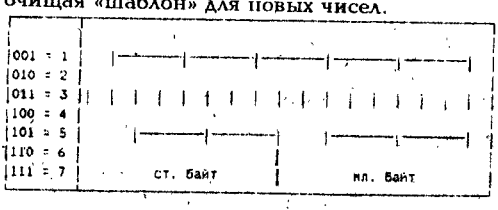
Переведем верхнюю и нижнюю части предыдущего рисунка в восьмеричные числа. Сверху получим слово, а снизу — два байта.



(Здесь символы «'8» означают восьмеричный формат, а «16б» — 16-битную запись.)

Вот теперь видно, что два байта составляют одно целое слово, хотя их арифметическая сумма не равна слову. Подобным способом можно переводить 16-битное слово в восьмеричное (или в два восьмеричных байта), восьмеричное слово в два восьмеричных байта (превратив его сначала в 16-битное), или два восьмеричных байта в одно восьмеричное слово (также через двоичный формат).

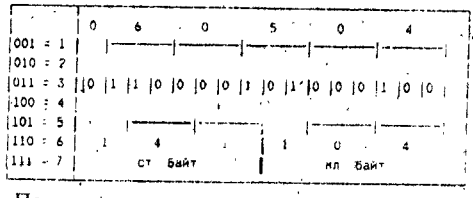
Таким преобразованием слов в пары байтов и обратно вам придется заниматься всю оставшуюся программистскую жизнь! Но не пугайтесь — говорят, что можно выучить даже китайский язык. А тут всего-навсего простая арифметика из восьми цифр. Для облегчения решения таких задач автор советует сделать простейшее пособие: на листочке бумаги, выдерживающей многоразовое пользование ластиком, нарисуйте тушью или фломастером «заготовку» для 16-битного слова с разбивкой на тройки для слова и байтов. На таком «шаблоне» можно писать мягким карандашом конкретные числа, а затем стирать их ластиком, очищая «шаблон» для новых чисел.



Но вернемся к нашему примеру:

```
001000 ADD R5,R4 060504 104.141 D a
```

Проделаем преобразование слова 60504 в два байта. Сначала сверху над каждой триадой битов (включая и неполную) надпишем по порядку слева направо цифры восьмеричного слова и переведем их в биты. Затем полученные биты переведем в восьмеричные цифры для байтов и подпишем их снизу:



Посмотрев на наш пример, списанный с отладчика, увидим, что последовательность написания байтов не совпадает с полученной нами картинкой. Но мы не ошиблись! Это автор отладчика OTL16 умышленно поменял местами байты. Зачем — поговорим позже, а пока просто примем это к сведению. Если у вас отладчик другого типа, то проверьте порядок вывода байтов — он может отличаться от показанного здесь. Тогда, читая эту статью, вносите соответствующие поправки в свои действия.

Итак, ассемблер или отладчик, увидев мнемонику, работает с 16-битными двоичными числами, а для нас превращает их в восьмеричные. Найдите в памяти (с помощью отладчика) какое-нибудь восьмеричное слово и его байты. Проделайте вручную обратное преобразование, т. е. впишите в шпаргалку байты и получите слово. Такая тренировка в будущем очень и очень пригодится!

Из Turbo в OTL

Если при работе с магнитофоном вам нужно перейти из отладчика в Turbo или наоборот, то можно обойтись без длительной перезагрузки системной программы с магнитной ленты. Для этого считайте в ОЗУ обе системные программы сразу, но так, чтобы они не мешали друг другу и вашей программе. Например, Turbo (его длина 11270) загрузите с адреса 16000. Это удобно, так как число 16000 легко запоминается, что потребуется в дальнейшем. Тогда конец транслятора придется на адрес 16000+11270=27270.

Внимание! Изменять адрес загрузки можно не у всех программ, а только у перемещаемых, т. е. написанных с соблюдением спе-

циальных правил (Turbo сделан именно так). Если вы пользуетесь другими ассемблер-транслятором и отладчиком, описанный здесь прием может не сработать. Например, отладчик Mirage — не перемещаемый. Конкретные сведения о перемещаемости можно найти в документации к программам.

Отладчик OTL16 загрузите с адреса 30000, тогда его конец будет приходиться на адрес 37776. Участок же ОЗУ с адресами от 1000 до 15776 остается в вашем распоряжении. Пишите в нем что хотите, только не «залезайте» выше 16000, чтобы не испортить Turbo.

Вся память БК при этом будет распределена таким образом:

0...777	**системная область и стек (не трогать!)**	
1000...15777	**наша рабочая область**	**ОЗУ для наших программ**
16000...27271	**здесь размещен Turbo**	
27272...27777	**в данном случае не используется**	
30000...37777	**здесь находится OTL16**	
40000...77777	**область ОЗУ экрана**	
100000...117777	**системное ПЗУ (монитор БК)**	
120000...175577	**ПЗУ с транслятором БЕЙСИКа. При работе с дисководом здесь может находиться ДОС (дисковая операционная система) ANDOS, NORD и др. При подключении блока МСТД здесь будут располагаться ФОКАЛ и тесты МСД**	
177600...177777	**область системных регистров**	

Проделаем все это на практике:

- загрузим OTL16 с адреса 30000;
- очистим ОЗУ от 1000 до 27776: **1000A 30000$0 «ВВОД»** ;
- загрузим Turbo4 с адреса 16000 и дадим команду SC;
- напишем программу

```
MOV #270,R0
EMT 16
.END
```

- дадим команды «ВС», «ТАБ», CO, RU, ST.

Теперь, чтобы перейти в OTL, наберите такую цепочку команд: **MO S30000 «ВВОД» 1000A «ЗАПЯТАЯ»**. Мы видим программу, написанную в Turbo. Для перехода же из OTL в Turbo надо набрать: **K S16000 «ВВОД»** .

При этих переходах написанная в Turbo программа сохраняется в адресах от 1000 и до 16000. Перейдя в OTL, вы увидите ее с адреса 1000 и сможете заняться отладкой. Но, вернувшись в Turbo, вы свою программу уже не найдете! Поэтому перед выходом из Turbo записывайте ассемблерные листинги в файлы на диске (или магнитной ленте) по команде ST, а вернувшись в Turbo, загружайте командой LO. Впрочем, для работающих с дисководом проблемы перехода из Turbo в OTL нет — чтение и запись любых файлов здесь дело недолгое.

Чтобы написанную в отладчике программу (или вообще любую в машинных кодах) загрузить в Turbo, надо получить ее мнемонический текст. Такой процесс называется дезассемблированием и возможен с помощью системной программы Dizas.WS.

Теперь, написав свою программу в Turbo, запишите ее для сохранения по команде ST. Затем перейдите в OTL и посмотрите, как она там выглядит: **1000A «ЗАПЯТАЯ» ...** Можно запустить ее, выполнить пошаговую трассировку, проследить изменения содержимого регистров и т. д. В отладчике же можно посмотреть, как выглядят «внутренности» Turbo. Наберите **16000A «ЗАПЯТАЯ» «ПОВТОР» ...** А чтобы увидеть «устройство» самого OTL16, наберите **30000A «ЗАПЯТАЯ» «ПОВТОР» ...**

ОЗУ экрана

Оперативная память с адресами от 40000 до 77776 называется ЭКРАННОЙ. Как только в нее попадает какая-либо информация, она тут же интерпретируется как закодированное изображение и выводится на экран дисплея.

Указание места на экране для вывода символа или текстовой строки возможно двумя способами: по команде EMT 24 и через ячейку 160, а графики — по командам EMT 30 и EMT 32, а также прямой записью с помощью MOV (как это делается при выводе спрайтов).

Команда EMT 24

Команда EMT 24 (с ней мы уже познакомились раньше) задает координаты знакоместа, с которого будет выводиться очередной текст по EMT 20 или символ по EMT 16. Координаты указываются содержимым регистров R1 (номер позиции в строке, от 0 до 31. или 63. в зависимости от режима 32/64 символа в строке) и R2 (номер строки, от 0 до 23.). Здесь числа с ТОЧКОЙ — десятичные. Признаком «десятичности» числа в ассемблере является завершающая ТОЧКА или присутствие в нем цифр 8 и (или) 9. Иногда вместо ТОЧКИ еще пишут «Д» или «D». Пример листинга:

```
MOV #21,R1   MOV #12,R2  EMT 24 ; задание координат курсора
MOV #277,R0  EMT 16             ; вывод символа с кодом 277
.END
```

Меняйте числа в первой строке — выводимый символ будет смещаться по горизонтали и вертикали (по координатам X и Y) на величину, кратную ширине/высоте курсора. Следующая программа дает возможность проследить, в какие строки можно установить курсор по EMT 24, а заодно подсчитать количество строк на экране. Должно быть 24. А что получается у вас?

```
    MOV #10000,R3  ; константа задержки
    MOV #0,R2      ; координата Y
N:  MOV #20,R1     ; координата X
    EMT 24         ; вывод
    INC R2         ; Y=Y+1
M:  SOB R3,M       ; задержка
    BR N
    .END
```

Ячейка 160

В служебную ячейку 160 монитор заносит абсолютный адрес ОЗУ экрана, с которой затем будет выводиться текст (верхняя графическая строка текущего знакоместа).

Адреса ОЗУ экрана размещаются на экране так:

```
┌──────────────────────────────┐
|40000 40001 40002 ... 40077|
|40100 40101 40102 ... 40177|
|40200 40201 40202 ... 40277|
 ..... ..... ..... ... .....
|77700 77701 77702 ... 77777|
└──────────────────────────────┘
```

Например, адреса 40100..40177 соответствуют второй сверху графической строке экрана. Вообще же экран состоит из 777−377=400=256. точечных строк, или из 40077−37777=100=64. позиций в строке, или из 256.*64.=16384. знакомест. (Ширина каждого знакоместа соответствует одному байту информации*.)

А теперь немного поиграем с графикой. Проделайте в отладчике следующие действия:

50070A ВВОД **377** ВВОД (младший байт);

51070A ВВОД **100377** ВВОД (младший байт с границей старшего байта);

52070A ВВОД **177777** ВВОД (слово = 2 байта);

53070A ВВОД **100001** ВВОД (границы слова);

54070A ВВОД **177401** ВВОД (граница младшего байта и старший байт);

55070A ВВОД **111111** ВВОД (6 точек с интервалом по 2 точки).

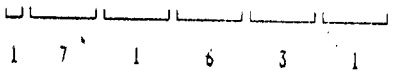
Каждому биту соответствует одна графическая точка. Значит, каждый байт может высветить до 8 точек**. Обратите внимание на то, что младший байт рисуется слева, а старший — справа. Луч кинескопа, как известно, движется по точечным строкам слева направо. Каждый байт рисуется лучом кинескопа тоже слева направо от младшего бита (с номером 0) к старшему (с номером 7). Для ясности напишем некоторое 16-битное слово, соответствующее четырем группам точек с интерва-

* Эти сведения соответствуют режиму вывода 64 символов в строке, для 32 символов число позиций в строке и знакомест на экране уменьшается вдвое по сравнению с указанным, а ширина каждого знакоместа соответствует двум байтам. — *Прим. ред.*

** Опять-таки это верно только в режиме 64 символа в строке. При 32 символах в строке каждая точка кодируется парой битов, т. е. слово соответствует восьми точкам, а байт — четырем. — *Прим. ред.*

лом по два нуля, причем с правой стороны записана одна точка, а с левой — четыре.

Переведем его в восьмеричное слово:



А теперь добавим его в отладчике к предыдущему примеру:

```
56070A   "ВВОД"   171631   "ВВОД"
```

На экране нарисована полоска с одной точкой слева и с четырьмя — справа, т. е. на экран выводится «зеркальное» отображение заданного слова. Это надо запомнить! (Адрес ОЗУ экрана можно брать любые, но подумайте, почему автор предлагает именно эти.)

Показанное изображение, к сожалению, теряется при очистке экрана, перезапуске отладчика или после команды О. Но можно написать в Turbo и «сбросить» на диск или ленту программу*:

```
MOV #377,@#50070
MOV #100377,@#51070
MOV #177777,@#52070
```

и т. д.

Вывод точек на экране цветного монитора смотрится интереснее. Пусть, например, мы хотим увидеть точки, расположенные в таком порядке цветов: **КчСчККчЗ** (здесь «ч» — черная точка, а «К», «С» и «З» — красная, синяя и зеленая соответственно). Здесь в слове кодируется только 8, а не 16 точек, так как для вывода одной цветной точки требуется 2 бита (один байт может высветить до четырех цветных точек). Указанное расположение цветных точек выбрано не случайно: при их выводе на экран можно точно определить точки, относящиеся к разным байтам.

Вспомним, что слова (и байты) на экран выводятся в зеркальном отображении относительно «рисунка» числа на бумаге. Поэтому заданную последовательность цветов мы запишем в зеркальном виде **ЗчККчСчК**, а при выводе на экран она вернется к требуемому порядку чередования точек.

Кодирование цвета отдельной точки делается так: 01 — синий, 10 — зеленый, 11 — красный, 00 — черный. Закодируем заданные восемь цветных точек парами битов:

```
З  ч  К  К  ч  С  ч  К
10 00 11 11 00 01 00 11 =
= 1000111100010011
```

Получилось 16-битное слово. Преобразуем его в восьмеричную форму, которую уже можно будет ввести в нашу программу:

1 000 111 100 010 011 = 107423

Наберем в Turbo листинг маленькой программки:

```
MOV #107423,@#60040
HALT
.END
```

Запустим ее и посмотрим результат. Ну как? Наше пожелание свершилось? Добавьте в листинг строки, которые отличаются от адреса 60040 на 1, 2, 100 в сторону увеличения или уменьшения адреса, чтобы прочувствовать их связь с получаемой картинкой.

А теперь опробуйте в работе такую программу:

```
    MOV #30, R3
    MOV #60040,R2
M:  MOV #107423,(R2)
    ADD #100,R2
     ; вместо 100 впишите 101, 102,
    SOB R3,M 201,
          ; 202, 77 и другие числа
    HALT
    .END
```

Попытайтесь найти объяснение возникающим картинкам.

А теперь вернемся к рассмотренному несколькими главами выше примеру, чтобы взглянуть на него «с новых позиций»:

```
; Текст в служебной строке
    CLR R0      EMT 22
       ; очистить служебную строку
    EMT 6
     ; ждать нажатия любой клавиши
    MOV #40130,@#160
; адрес начала текста в сл. строке
    MOV #T,R1  CLR R2   EMT 20
                      ; выдать текст
    HALT
T:  .ASCIZ/Текст в служебной строке./
    .END
```

* Фактически эта программа демонстрирует простейший способ вывода спрайта на экран. Правда, «настоящие» спрайты представляют собой массивы байтов с закодированными изображениями, а программа должна предоставлять возможность их вывода в любом месте экрана. — *Прим. ред.*

Команду MOV #40130,@#160 мы понимаем как «число 40130 поместить в ячейку 160». А эта ячейка служит для хранения адреса верхнего левого угла курсора в памяти экрана, отсюда же начнется вывод очередного символа. В качестве адреса курсора мы задаем число 40130, с этого адреса (в служебной строке!) и выводится задаваемый нами текст.

Наберите такую программу:

```
MOV #50020,160
        ; задание начального адреса
MOV #177,R0      EMT 16
        ; вывод символа с кодом 177
MOV #50017,160
 ; задание нач. адреса для символа
277
MOV #277,R0      EMT 16
        ; вывод символа с кодом 277
.END
```

Эта программа интересна тем, что на экране рядом выведены два символа и курсор, причем четко видна разница в занимаемой ими площади. Так, курсор и символ с кодом 277 одинаковы. Если посмотреть через лупу, можно увидеть, что они занимают прямоугольник 8·10. точек.

Поменяйте в третьей строке на единицу последовательно вторую, третью, четвертую и пятую цифры (кроме начальной «5») и проследите изменения в результатах работы программы. Выводы сделайте сами.

Адреса ячеек в каждой графической строке ОЗУ изменяются от 0 (слева) до 77 (справа), или в десятичном виде — от 0 до 64., т. е. в каждой строке может поместиться 100=64. байтов. Следовательно, изменение адреса на единицу смещает выводимый символ по горизонтали на ширину одного «узкого» курсора. Изменение адреса на 100 означает смещение символа по вертикали на одну графическую строку. А изменение адреса на 1200 дает смещение символа по вертикали на 10. точечных строк, т. е. на высоту курсора (на следующую символьную строку). Вывод: адрес, задаваемый через ячейку 160, указывает положение верхнего левого угла знакоместа, в котором печатается очередной символ.

В команде MOV #40130,@#160 мы задали адрес 40130. Это означает вторую графическую строку (**401**) и тридцатую (восьм.) позицию в строке (***30**). Здесь в скобках указанный нами адрес условно разложен на «вертикальную» и «горизонтальную» части, а звездочки заменяют не существенные для каждой из них цифры. Изменяя третью цифру, вы увидите смещение текста по вертикали, а четвертую и пятую — по горизонтали.

Команда EMT 30

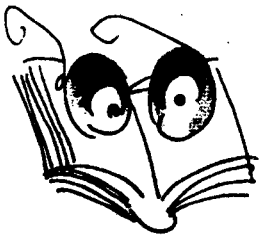
Используя EMT 30 (с ней мы также уже познакомились раньше), можно вывести графическую точку в одну из 377 (255.) графических строк и в одну из 100·10=1000 (64.·8=512.) точечных позиций в строке. Всего же точек на экране может быть 255.·512.=130560.

(Продолжение следует.)

Поправка к ранее опубликованной статье

В статье «Подключение дисковода к БК» (№4 за 1995 г.) было сказано, что система NORD вряд ли подходит для серьезной работы, так как дискета быстро «забивается» одноименными файлами. Данное замечание было бы справедливо в отношении устаревшей версии NORD 2.4, семейство же версий NORD 3.xx не обладает указанным недостатком. А именно: при первой записи одноименного файла в имя его старого варианта добавляется символ «"» («кавычки»), а при второй — файл с символом «"» в имени автоматически удаляется. Добавлю, что программа частичного автоматического сквизирования при записи позволяет работать в NORD третьей версии практически без использования специальных программ-сквизеров.

А. Г. Прудковский,
автор ОС NORD



Природа дала человеческому зрению такое важное свойство, как способность восприятия окружающего мира в пространственной (объемной) форме. Но наибольшая часть иллюстративного материала (рисунки, чертежи, фотографии и т. п. в книгах и журналах, на экране дисплея, в кино и на телевидении) лишена объемности. Конечно, голография позволяет сохранять полноценные копии предметов, но стоимость оборудования и фотоматериалов для получения голограмм пока еще слишком велика для повседневного использования.

Однако есть и другие способы передачи в плоском изображении иллюзии объемности, достаточно дешевые, но требующие значительного объема математических расчетов. Помочь в этом может персональный компьютер, в том числе БК.

Д. Ю. Усенков,
Москва

Стереоскопические изображения на БК

Насколько важным является для нас пространственное (бинокулярное) зрение, наглядно показывает следующий пример. Вденьте нитку в иглу — на эту несложную операцию вам, скорее всего, потребуется лишь 3—5 секунд. А теперь попытайтесь сделать то же самое, закрыв один глаз. Если и получится, то только после большого количества попыток: без объемного видения трудно определить взаимное расположение нитки и иголки.

Может быть, сознательный отказ от возможности видеть обоими глазами покажется читателю несколько абсурдным. Но разве не то же самое происходит, когда мы рассматриваем иллюстрации в книге или журнале? Конечно, в большинстве случаев, чтобы правильно понять изображенное на рисунке, вполне достаточно плоской картины или, в крайнем случае, изометрии. Но для понимания сложных чертежей изометрии иной раз недостаточно. Вот, например, полипептидная цепь в молекуле белка (рис. 1, слева): даже специалисту — биологу нелегко разобраться в этом переплетении линий. Насколько удобнее и понятнее было бы объемное изображение молекулы, даже если мы не имеем возможности рассматривать ее с разных сторон, а только определяем, какой из отрезков находится ближе к нам, а какой — дальше. Можно ли добиться стереоскопического эффекта, не прибегая к дорогостоящей голографии?

Рис. 1. Полипептидная цепь молекулы белка (стереоскопическое изображение)

Чтобы ответить на поставленный вопрос, вначале необходимо разобраться, как «устроено» объемное зрение. Для этого нам придется совершить небольшой экскурс в биологию.

Основы бинокулярного зрения

Бинокулярное (объемное) зрение присуще, как правило, хищным животным, вынужденным охотиться за подвижной добычей. Их глаза располагаются на передней части головы так, что при разглядывании далеких предметов главные оси глаз (лучи зрения) параллельны между собой, а видимые области для обоих глаз перекрываются (один и тот же объект одновременно виден двумя глазами). В отличие от хищников, глаза большинства травоядных животных

расположены по бокам головы, так что общее поле зрения охватывает почти полную сферу: вероятность вовремя заметить приближающегося врага существенно повышается.

Человек, как и другие приматы, обладает «взглядом хищника»: его глаза обеспечивают стереоскопическое видение предметов практически во всем поле зрения (кроме небольших участков по краям слева и справа). Расстояние между зрачками человека, называемое *стереобазисом*, колеблется в пределах от 54 до 72 мм и в среднем (по результатам большого числа наблюдений) равно 63—64 мм. Это расстояние сравнительно невелико, поэтому мы можем видеть в объеме только сравнительно близкие предметы (на расстоянии до нескольких десятков метров), а при удалении к горизонту стереоэффект уменьшается. (Используя стереотрубы, можно увеличить стереобазис и получить возможность пространственного видения далеких объектов.)

Когда мы фиксируем взгляд на выбранной точке какого-либо предмета, производится *аккомодация* хрусталиков (аналог «наведения на резкость» объектива фотоаппарата) и, кроме того, оси обоих глаз направляются на данную точку (*конвергенция*). При этом получающиеся на сетчатке каждого глаза изображения немного отличаются друг от друга (1 и 2 на рис. 2), а мозг, анализируя эти отличия, вырабатывает ощущение глубины пространства. При этом расстояние между проекциями одной и той же точки наложенных друг на друга изображений на сетчатке правого и левого глаза (*параллакс*) тем больше, чем дальше данная точка отстоит в пространстве от некоторой «нулевой» плоскости (перпендикулярной лучу зрения и проходящей через точку, на которую сфокусирован взгляд).

Чтобы получить объемное изображение, мы должны научиться рисовать отдельные картинки (части *стереопары*) для правого и левого глаза и рассматривать каждую картинку только тем глазом, для которого она предназначена. Но как обеспечить подобную избирательность зрения — «заставить», например, правый глаз видеть только «правую» картинку, но не «левую»?

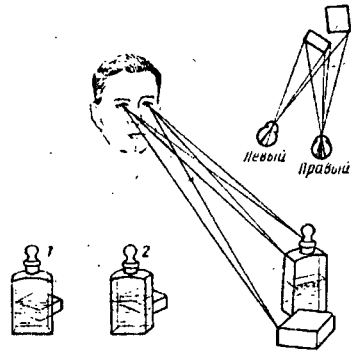


Рис. 2. Различие изображений, видимых левым (1) и правым (2) глазом

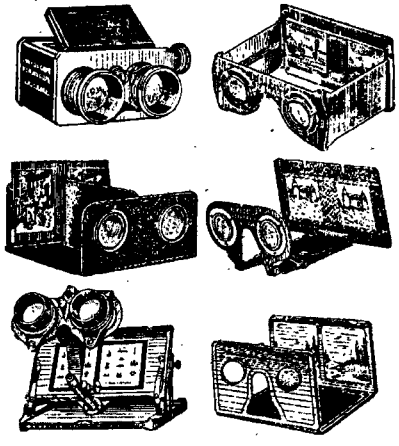


Рис. 3. Различные варианты стереоскопов для рассматривания стереопар (фотографий и рисунков)

Технические приемы рассматривания стереопар

Рассмотрим несколько основных способов, позволяющих обеспечить раздельное видение «левой» и «правой» стереопар соответствующим глазом.

Наиболее простой всем известный метод — использование несложного устройства, именуемого *стереоскопом*. Сегодня стереоскопы применяют в основном для просмотра слайдов на просвет, но два-три десятка лет назад аналогичные приборы существовали и для рассматривания стереофотографий ([1]). Основной принцип работы стереоскопа — искусственное разделение поля зрения на «левую» и «правую» области и размещение в них соответствующих частей стереопары (рис. 3 и 4).

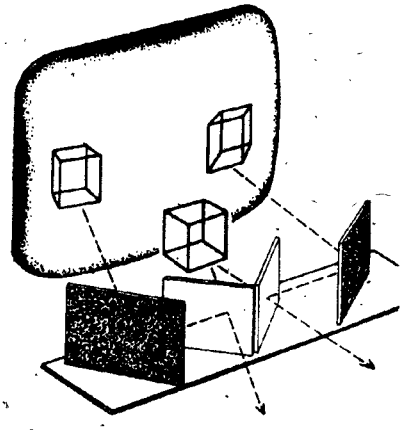


Рис. 4. Принцип работы стереоскопа (при рассматривании стереопары на экране дисплея)

Однако стереоскоп обладает и значительным недостатком: он рассчитан на индивидуальное использование, тогда как в некоторых случаях желательно обеспечить возможность рассматривания объемного изображения одновременно несколькими зрителями. Сделать это позволяет другой метод, называемый *анаглифическим*. Здесь «левое» и «правое» изображения вычерчиваются соответственно красным и зеленым (или синим) цветом и накладываются на экране одно поверх другого. Рассматривают такой чертеж через специальные очки со светофильтрами тех же цветов, расположенными таким образом, чтобы правый глаз видел сквозь, например, зеленый светофильтр вычерченную красным цветом «правую» картинку, а левый сквозь красный светофильтр — «левую» зеленого цвета ([2]). Очевидный недостаток данного метода — невозможность получения многоцветных пространственных изображений.

Следующий, *поляроидный* метод свободен и от этого недостатка. Здесь вместо цветных светофильтров используются поляризационные с взаимно перпендикулярными плоскостями поляризации (рис. 5). Именно на таком принципе работает стереокинотеатр «Октябрь» на Новом Арбате в Москве.

И наконец, последний из рассматриваемых нами метод — использование *растровых систем* и его разновидность *интегральная фотография* позволяют вообще отказаться от специальных очков. Основной принцип здесь заключается в наложении на состоящее из чередующихся «левых» и «правых» полос изображение сетки из вертикальных поглощающих свет линий, треугольных призм или цилиндрических линз (рис. 6), обеспечивающих раздельное видение левым и правым глазом соответствующих полос (конечно, на рис. 6 ширина полос преувеличена по сравнению с реальной).

Какой же из этих способов наиболее подходит для получения стереоизображения на экране компьютера и объемных иллюстраций в книгах или журналах? Учитывая, что чертежи чаще всего выполняются одним цветом, можно было бы выбрать анаглифический метод. Но наиболее простым и вместе с тем универсальным является использование раздельных стереопар, к тому же рассматривать их можно и без всякого стереоскопа! Вернемся к рис. 1. Поднесите его к глазам на расстояние 15—20 см и смотрите как бы сквозь него вдаль. При этом левый глаз будет видеть левый рисунок, а правый глаз — правый. Оба изображения начнут сближаться и вскоре сольются в единый объемный рисунок. Возможно, при первых попытках у вас возникнут некоторые трудности. Тогда разделите картинки полоской бумаги, установленной перпендикулярно странице. Или вначале поднесите рисунок еще ближе к глазам (пусть даже он будет виден нерезко), а после слияния двух картинок в одну постепенно отодвигайте страницу от глаз, сохраняя слияние, пока изображение не станет резким (в данном случае близорукие читатели оказываются в несколько выигрышном положении по сравнению с имеющими нормальное зрение). Есть и другой способ: глядя на стереопару, скосите глаза на воображаемую точку ПЕРЕД рисунком — картинки тоже сольются в одну объемную. При этом левый глаз будет видеть правую картинку, а правый — левую. Для рис. 1 это несущественно, но при рассматривании таким образом стереофотографии нужно поменять местами ее левую и правую части (по сравнению с рассматриванием «вдаль»), иначе изображение на ней «вывернется наизнанку».

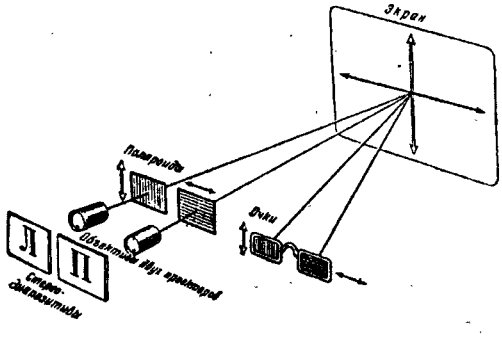


Рис. 5. Проекция стереопары на экран в поляризованном свете

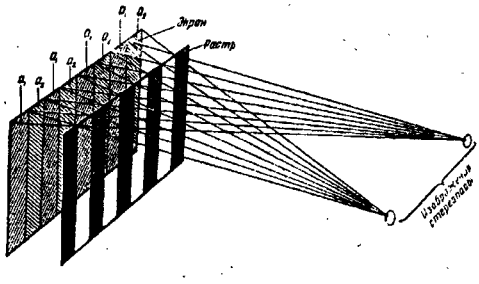


Рис. 6. Использование растровой системы

Использование стереопар имеет, помимо простоты и возможности работы с цветом, еще одно преимущество. Рассматривать их можно с одинаковым успехом как непосредственно на экране дисплея (кстати, появившиеся недавно специальные «видеоочки» для систем виртуальной реальности также представляют собой своего рода стереоскоп), так и после распечатки графической копии экрана на принтере. Поэтому отдадим предпочтение именно данному методу.

Итак, мы изучили принцип бинокулярного восприятия и определили метод рассматривания стереоскопических изображений. Осталось только научиться рисовать стереопары требуемой объемной фигуры. Очевидно, для этого потребуется значительное количество расчетов, посильное разве что для компьютера. Но сначала необходимо «научить» этому ЭВМ — вывести формулы и составить алгоритм.

Математические основы стереоскопии

Исходные данные (рис. 7—9): **(Xa,Ya,Za)** — координаты в мм некоторой точки пространства A, проекции которой на «левой» и «правой» частях стереопары необходимо определить; **F** — расстояние от наблюдателя до стереопары или экрана, мм; **D** — расстояние от наблюдателя до плоскости, проходящей через начало координат, мм (D ≥ F); **d** — стереобазис, мм; **h** — «рост» наблюдателя, мм (высота точки, откуда производится наблюдение, относительно горизонтальной плоскости, проходящей через начало координат).

Изменение стереобазиса d определяет глубину стереоэффекта, а отдельные плоскости стереопары и начала координат введены для удобства (стереоэффект лучше всего проявляется, когда плоскость стереопары проходит ближе к передней части изображаемого предмета, но это не всегда удобно для отсчета координат его точек).

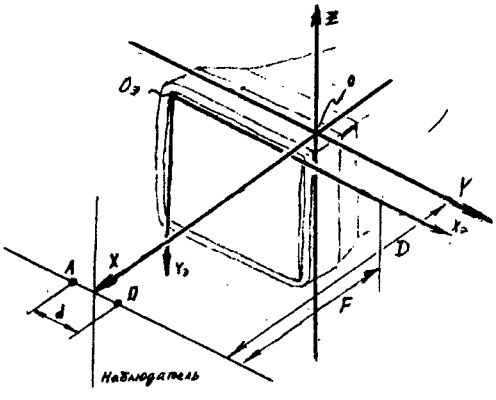


Рис. 7. Системы координат и основные параметры, используемые в расчетах

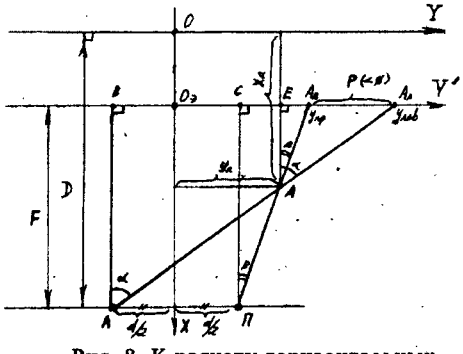


Рис. 8. К расчету горизонтальных координат проекций точки для левого и правого глаза

Для вывода на основе рис. 8 необходимых формул достаточно школьных знаний геометрии и алгебры, поэтому ниже мы не будем подробно расписывать все этапы вывода, а ограничимся лишь основными из них.

Горизонтальная координата проекции точки A для левой части стереопары.

$$\begin{cases} \frac{d}{2} + y_{лев} = F \cdot tg\alpha; \\ y_a + (x_a - D + F) \cdot tg\alpha = y_{лев}; \end{cases} \Rightarrow \begin{cases} tg\alpha = \frac{(y_{лев} - y_a)}{(x_a - D + F)}; \\ \frac{d}{2} + y_{лев} = F \cdot \frac{(y_{лев} - y_a)}{(x_a - D + F)}; \end{cases}$$

Отсюда

$$y_{лев} = \frac{F \cdot y_a + \frac{d}{2} \cdot (x_a - D + F)}{D - x_a}$$

Рис. 9. К расчету вертикальной координаты проекций точки (одинакова для левого и правого глаза)

Горизонтальная координата проекции точки A для правой части стереопары.

$$\begin{cases} y_{пр}=\frac{d}{2}+F\cdot tg\beta; \\ y_{пр}=y_a+(x_a-D+F)\cdot tg\beta; \end{cases} \Rightarrow \begin{cases} tg\beta=\frac{(y_{пр}-y_a)}{(x_a-D+F)}; \\ y_{пр}=\frac{d}{2}+F\cdot\frac{(y_{пр}-y_a)}{(x_a-D+F)}; \end{cases}$$ Отсюда $$y_{пр}=\frac{F\cdot y_a-\frac{d}{2}\cdot(x_a-D+F)}{D-x_a}$$

Обратите внимание: хотя изображение на рис. 8 на первый взгляд несимметричное, в выведенных формулах явно видно подобие друг другу (дроби для $y_{лев}$ и $y_{пр}$ отличаются только знаком «плюс» или «минус» в числителе), отображающее симметрию бинокулярного зрения. Эта своеобразная красота математики является косвенным подтверждением правильности математических выкладок.

Вертикальная координата проекции точки A (одинакова для левой и правой частей стереопары — см. рис. 9).

$$tg\gamma=\frac{h}{F+a}=\frac{z_a}{x_a-D+F+a}=\frac{z_{пл}}{a}$$ Отсюда $$z_{пл}=\frac{F\cdot z_a-h\cdot(x_a-D+F)}{D-x_a}$$

Параллакс (вспомогательное вычисление для проверки выведенных формул).

$$P=-\frac{d\cdot(x_a-D+F)}{D-x_a}$$ Действительно, при $x>0$ значение параллакса $P<0$ (см. рис. 8).

Использование компьютера для построения стереопар

Полученные по выведенным нами формулам координаты (в мм) проекций точки A необходимо преобразовать в координаты их изображений на экране. Исходные данные: ($y_{лев}$, $y_{пр}$, $z_{пл}$) — координаты проекций точки, мм; ($x_{эл}$,$y_{эл}$), ($x_{эп}$,$y_{эп}$) — координаты изображений «левой» и «правой» проекций точки A на экране в пикселах; ($X_{0эл}$,$Y_{0эл}$), ($X_{0эп}$,$Y_{0эп}$) — экранные координаты «левой» и «правой» проекций точки (0,0,0), в пикселах; K_x, K_y — «скейл-фактор» (число точек на 1 мм экрана или на бумаге при печати на принтере по горизонтали и вертикали соответственно).

Для компьютера БК при работе с монитором 32ВТЦ201 значения K_x и K_y, определенные по формуле: K=<кол-во точек в линии>/<длина линии в мм>, равны

в режиме 32 символа в строке $\begin{cases} K_x=1.25 \text{ точки/мм}, \\ K_y=1.67 \text{ точки/мм}; \end{cases}$

в режиме 64 символа в строке $\begin{cases} K_x=2.5 \text{ точки/мм}, \\ K_y=1.67 \text{ точки/мм}. \end{cases}$

Кроме того, нам понадобится ввести поправку высоты проекции точки. Если выбрать значения F, D и h достаточно большими, изображение может «уплыть» с экрана. Чтобы этого не произошло, необходимо уменьшать вычисленное ранее значение $z_{пл}$ на величину

$$z'=\frac{h\cdot(D-F)}{D},$$

что является математическим аналогом смещения «прозрачного окна, через которое мы рассматриваем объект». При этом проекция точки с координатами (0,0,0) должна оказаться на оси [$X_{0эл}$,$X_{0эп}$].

Тогда экранные координаты проекций точки A будут равны:

для левой части стереопары: $\begin{cases} x_{эл}=X_{0эл}+K_x*y_{лев}, \\ y_{эл}=Y_{0эл}-K_y*(z_{пл}-z'); \end{cases}$

для правой части стереопары: $\begin{cases} x_{эп}=X_{0эп}+K_x*y_{пр}, \\ y_{эп}=Y_{0эп}-K_y*(z_{пл}-z'). \end{cases}$

Следует обратить внимание на то, что вышеприведенные формулы для вычисления $(x_{эл},y_{эл})$ и $(x_{эп},y_{эп})$ соответствуют «нормальной» стереопаре, для получения «обратной» стереопары (взгляд скосив глаза на точку перед бумагой) нужно поменять местами левую и правую части изображения.

Теперь нетрудно преобразовать выведенные формулы в строки программы на БЕЙСИКе. Приведенный ниже листинг демонстрирует построение стереоскопического изображения куба (со всеми его диагоналями).

```
5 DIM A(10,3)
10 CLS
15 H=150                          ' h
20 D=300                          ' D
25 F=D                            ' F=D
30 DG=65                          ' d (стереобазис)
40 FOR I=1 TO 8                   ' чтение
50 READ A(I,1),A(I,2),A(I,3)      '  координат
60 NEXT                           '  вершин куба
70 ? CHR$(155);                   ' режим 64 с/стр
80 LINE (0,170)-(512,170)         ' черчение систем
90 LINE (127,0)-(127,240)         '  координат левой
100 LINE (384,0)-(384,240)        '  и правой частей
110 FOR I=1 TO 8
120 FOR J=I TO 8
130 GOSUB 1000                    ' вычисление координат
140 NEXT J,I
150 ? CHR$(155);
160 END ' ------------ координаты вершин куба: -------
200 DATA 40,40,40
```

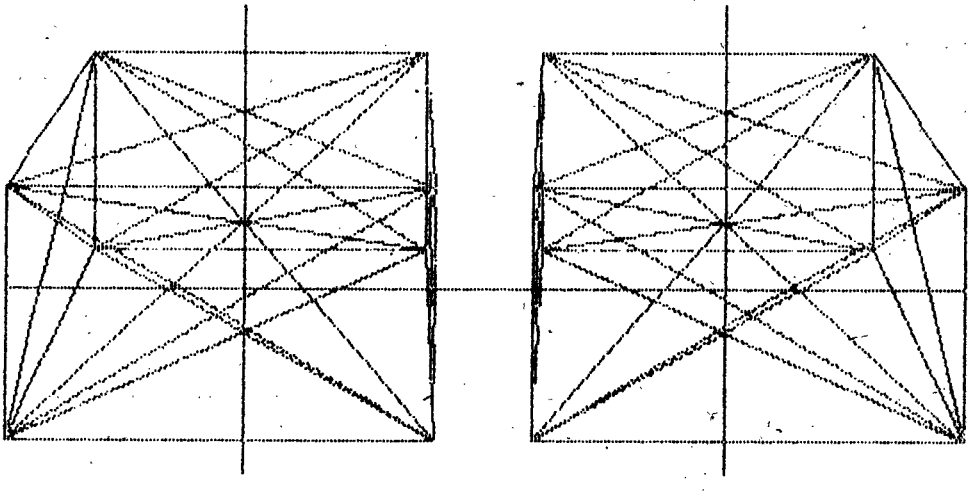


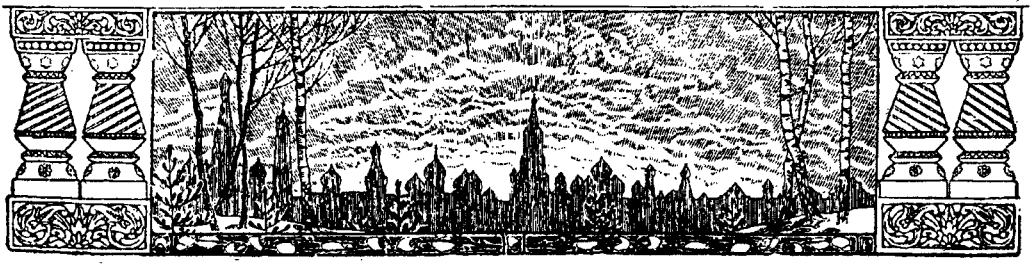
Рис. 10. Стереопара (куб со всеми его диагоналями), построенная на БК-0010.01

```
210 DATA -40,40,40
220 DATA 40,-40,40
230 DATA 40,40,-10
240 DATA -40,-40,40
250 DATA 40,-40,-10
260 DATA -40,40,-10
270 DATA -40,-40,-10
300 ' -------- расчет координат проекций точек --------
310 ' A(I,1) = xa; A(I,2) = ya; A(I,3) = za
320 ' A(J,1) = xb; A(J,2) = yb; A(J,3) = zb
330 ' где (xa,ya,za) и (xb,yb,zb) - пара вершин куба;
340 ' (X,Y), (X1,Y) - экранные координаты вершин (без поправки z')
350 ' -------- правая часть стереопары: ---------------
1000 X=(F*A(I,2)-DG/2*(A(I,1)-D+F))/(D-A(I,1))
1010 X=127+2.5*X ' выводится в левую половину экрана
1020 Y=170-1.67*(F*A(I,3)-H*(A(I,1)-D+F))/(D-A(I,1))
1030 X1=(F*A(J,2)-DG/2*(A(J,1)-D+F))/(D-A(J,1))
1040 X1=127+2.5*X1
1050 Y1=170-1.67*(F*A(J,3)-H*(A(J,1)-D+F))/(D-A(J,1))
1060 LINE (X,Y)-(X1,Y1)
1065 ' -------- левая часть стереопары: ---------------
1070 X=(F*A(I,2)+DG/2*(A(I,1)-D+F))/(D-A(I,1))
1080 X=384+2.5*X ' выводится в правую половину экрана
1090 X1=(F*A(J,2)+DG/2*(A(J,1)-D+F))/(D-A(J,1))
1100 X1=384+2.5*X1
1110 LINE (X,Y)-(X1,Y1) ' Y одинаково для левой и правой частей
1120 RETURN
```

В данном случае выполняется построение «обратной» стереопары (рис. 10) без учета поправки, так как значение z' в силу исходного условия D=F равно нулю. Структура листинга не сложна, и думается, читатели сами смогут разработать на базе приведенного листинга свои программы, например полнофункциональный графический редактор с возможностью построения (по заданию пользователя) стереопары или изометрии. При этом можно обеспечить выбор точки зрения (поворот объекта относительно наблюдателя) и учесть все необходимые поправки.

Литература

1. *Иванов Б. Т., Левингтон А. Л.* Стереоскопическая фотография. М.: Искусство, 1959.
2. *Климанов А. И.* Стереоскопические чертежи //Квант. 1984. №7. С. 45.
3. *Величкин С.* Стереоизображение на экране компьютера //Наука и жизнь. 1990. №12. С. 132.



Продолжаем публикацию руководства по программированию на языке ассемблера микроЭВМ «Электроника МС 0513» (БК-0011), начатую в №4 за 1995 г. В руководстве описаны состав и функции базовой операционной системы ЭВМ (далее — БОС) и способы обращения к ним из пользовательских программ.

Руководство системного программиста БК-0011(М). Базовая операционная система

Описание системных запросов базовой операционной системы

Ниже содержатся краткий перечень и подробное описание запросов базовой операционной системы БК-0011. Для облегчения программирования на языке ассемблера под управлением операционной системы ОС БК-11 в ее комплекте поставляется макробиблиотека, содержащая макроопределения системных вызовов. При описании функций БОС приводится два формата вызовов: макрокоманда и макрорасширение, которое может использоваться при программировании без макробиблиотеки. По умолчанию применяются команды JSR. Для использования EMT следует присвоить служебной переменной ...EMT значение, отличное от нуля.

Перечень вызовов БОС

- .BINIT (JSR PC,@140010, EMT 0): инициализация БОС.
- .BEXIT (140012, EMT 1): выход в клавиатурный монитор.
- .BTRPS (140014, EMT 2): перехват вектора 4.
- .BTIM (140016, EMT 3): управление таймером.
- .BEMT (140020, EMT 4): управление обработкой EMT.
- .BMEM (140030, EMT 10): чтение состояния памяти.
- .BTEST (140032, EMT 11): чтение статуса страницы.
- .BPAGE (140034, EMT 12): подключение страницы ОЗУ/ПЗУ.
- .BWORK (140036, EMT 13): назначение/чтение рабочей страницы.
- .BGETW (140040, EMT 14): чтение слова из рабочей страницы.
- .BGETB (140042, EMT 15): чтение байта из рабочей страницы.
- .BPUTW (140044, EMT 16): запись слова в рабочую страницу.
- .BPUTB (140046, EMT 17): запись байта в рабочую страницу.
- .BMOVE (140050, EMT 20): пересылка массива слов.
- .BMOVB (140052, EMT 21): пересылка массива байтов.
- .BJSR (140054, EMT 22): вызов подпрограммы из рабочей страницы.
- .BJMP (140056, EMT 23): переход в рабочую страницу без возврата.
- .BKINI (140070, EMT 30): инициализация драйвера клавиатуры.
- .BKSET (140072, EMT 31): установка режима драйвера клавиатуры.
- .BKGET (140074, EMT 32): чтение режима драйвера клавиатуры.
- .BTTIN (140076, EMT 33): ввод символа с ожиданием.
- .BINKEY (140100, EMT 34): ввод символа без ожидания.
- .BGLIN (140102, EMT 35): ввод строки с редактированием.
- .BEDIT (140104, EMT 36): редактирование существующей строки.
- .BKRES (140106, EMT 37): сброс буфера клавиатуры.
- .BSFUN (140110, EMT 40): установка функциональных клавиш.
- .BGFUN (140112, EMT 41): чтение функциональных клавиш.
- .BTINI (140130, EMT 50): инициализация драйвера экрана.
- .BTSET (140132, EMT 51): установка режима драйвера экрана.
- .BTGET (140134, EMT 52): чтение режима драйвера экрана.

- .BSCOL (140136, EMT 53): установка цветов.
- .BGCOL (140140, EMT 54): чтение цветов.
- .BPAL (140142, EMT 55): установка/чтение палитры.
- .BBUF (140144, EMT 56): переключение буферов экрана.
- .BSPOS (140146, EMT 57): установка координат алфавитно-цифрового курсора.
- .BGPOS (140150, EMT 60): чтение координат алфавитно-цифрового курсора.
- .BSCRL (140152, EMT 61): сдвиг экрана («скроллинг»).
- .BCLS (140154, EMT 62): очистка экрана.
- .BTOUT (140156, EMT 63): вывод символа.
- .BPRIN (140160, EMT 64): вывод строки символов (формат RT-11).
- .BSTR (140162, EMT 65): вывод строки символов.
- .BSGRF (140164, EMT 66): установка текущих графических координат.
- .BGGRF (140166, EMT 67): чтение текущих графических координат.
- .BSCGR (140170, EMT 70): установка графического цвета.
- .BGCGR (140172, EMT 71): чтение графического цвета.
- .BSMOD (140174, EMT 72): установка/чтение режима рисования.
- .BSPNT (140176, EMT 73): вывод графической точки.
- .BGPNT (140200, EMT 74): чтение цвета графической точки.
- .BVECT (140202, EMT 75): вывод вектора (отрезка).
- .BRECT (140204, EMT 76): вывод закрашенного прямоугольника.
- .BCIRC (140206, EMT 77): вывод дуги или окружности.
- .BFILL (140210, EMT 100): закраска замкнутой области.
- .BSOUN (140212, EMT 101): выдача звука.
- .BSWIN (140214, EMT 102): установка графического окна.
- .BGWIN (140216, EMT 103): чтение графического окна.
- .BSTYP (140222, EMT 105): установка типа вектора.
- .BGTYP (140224, EMT 106): чтение типа вектора.
- .BGOSET (140226, EMT 107): установка/чтение ориентации графических символов.
- .BGOUT (140230, EMT 110): графический вывод символа.
- .BMOT (140240, EMT 114): управление двигателем магнитофона.
- .BMB10 (140242, EMT 115): вызов формата БК-0010.
- .BPINI (140260, EMT 124): инициализация драйвера принтера.
- .BPSET (140262, EMT 125): установка режима печати.
- .BPGET (140264, EMT 126): чтение режима печати.
- .BPOUT (140266, EMT 127): вывод символа на печать.
- .BPRDY (140270, EMT 130): определение готовности принтера.
- .BPPOS (140272, EMT 131): определение позиции печатающей головки.
- .BABOOT (JSR PC,@#160000): автоматическая загрузка с НГМД.
- .BBOOT (160002): загрузка с выбранного привода НГМД.
- .BBLCK (160004): чтение/запись по номеру блока.
- .BSECT (160006): чтение/запись по номерам сектора и дорожки.
- .BDINI (160010): инициализация драйвера НГМД.
- .BFORM (160012): форматирование дорожки НГМД.

Общесистемные запросы

Инициализация базовой операционной системы .BINIT .
Макрокоманда: .BINIT
Расширение: JSR PC,@140010 (EMT 0)
Производится инициализация драйверов БОС.
Драйвер клавиатуры:
- настраиваются вектора прерывания клавиатуры;
- сбрасывается пользовательский вектор обработки клавиатуры;
- разрешаются прерывания от клавиатуры;
- разрешается щелчок при нажатии клавиш;
- сбрасывается буфер ввода с клавиатуры;

- сбрасываются значения функциональных клавиш;
- запрещается перекодировка клавиш;
- устанавливается режим ввода в коде КОИ-8.

Драйвер экрана:
- устанавливается режим 32 символа в строке и палитра 15 (белые символы на черном фоне);
- в ОЗУ копируется стандартный знакогенератор из ПЗУ;
- очищаются оба буфера экрана;
- включается запись и отображение буфера 0;
- выключается режим двойной ширины и двойной высоты символов;
- выключается режим инверсии и подчеркивания символов;
- выключается режим плавного сдвига экрана;
- включается отображение курсора в виде прямоугольника;
- включается режим нормальной обработки управляющих символов.

Драйвер магнитофона:
- выключается мотор магнитофона.

Драйвер принтера:
- на принтер (в случае готовности) посылается символ возврата каретки;
- обнуляется счетчик позиции печатающей головки;
- устанавливается режим вывода в КОИ-8.

Драйвер НГМД:
- выключается двигатель накопителя.

Выход в клавиатурный монитор .BEXIT .

Макрокоманда: .BEXIT
Расширение: JSR PC,@140012 (EMT 1)

Производится выход из программы пользователя в клавиатурный монитор. Настраиваются вектора клавиатуры, при этом возможен возврат в программу пользователя по команде «Р».

Перехват вектора 4 .BTRPS .

Макрокоманда: . BTRPS ADDR
Расширение: MOV ADDR,R0
JSR PC,@140014 (EMT 2).

Устанавливается адрес пользовательской программы для обработки вектора 4. Состояние останова (команда HALT или клавиша «СТОП») не перехватывается, в этом случае всегда происходит выход в монитор. Программа обработки должна заканчиваться командой RTI. Вызов с нулевым значением аргумента ADDR отменяет обработку вектора 4 в пользовательской программе. В обоих случаях вектор 4 настраивается на обработчик БОС.

Управление таймером .BTIM .

Макрокоманда: .BTIM ARG
Расширение: MOV ARG,R0
JSR PC,@140016 (EMT 3)

Если бит 15 ARG равен «1», то в R0 возвращается признак: 0 — таймер выключен, 1 — таймер включен. Если же бит 15 ARG равен «0», то при нулевом значении ARG происходит выключение таймера, при ненулевом — включение. Перед включением таймера следует подготовить вектор 100.

Управление обработкой EMT .BEMT .

Макрокоманда имеет две формы запроса.

1. *Макрокоманда:* .BEMT ON
Расширение: JSR PC,@140020
...EMT=1

Содержимое вектора EMT, если оно не равно адресу диспетчера EMT, помещается в ячейку 166 (настраивается пользовательский обработчик EMT). Макробиблиотека настраивается на использование команд EMT (ниже по тексту программы).

2. *Макрокоманда:* .BEMT OFF
Расширение: MOV @#166,@#30
...EMT=0

Восстанавливается прямая обработка EMT пользовательской программой. Макробиблиотека настраивается на использование команд JSR (ниже по тексту программы).

Запросы драйвера ОЗУ/ПЗУ

Чтение состояния памяти .BMEM .
Макрокоманда: .BMEM
Расширение: JSR PC,@140030 (EMT 10)
В младший байт R0 помещается номер страницы, подключенной к адресу 40000, а в старший — номер страницы, подключенной к адресу 100000. Номера страниц 0—7 относятся к ОЗУ, страницы 10—13 (числа восьмеричные) относится к страницам ПЗУ с номерами 0, 1, 2 и 3 соответственно.

Чтение статуса страницы ОЗУ/ПЗУ .BTEST .
Макрокоманда: .BTEST PAGE
Расширение: MOV PAGE,R0
JSR PC,@140032 (EMT 11)
Выдается состояние страницы ОЗУ/ПЗУ. В R0 помещается код:
- бит 0 — страница подключена по адресу 40000;
- бит 1 — страница подключена по адресу 100000;
- бит 2 — страница ОЗУ используется БОС или страница ПЗУ не подключена;
- бит 3 — страница является рабочей.

Примечание. Страница ПЗУ считается подключенной, если в ней присутствует хотя бы одна микросхема ПЗУ.

Подключение страницы ОЗУ/ПЗУ .BPAGE .
Макрокоманда: .BPAGE PAGE,ADDR
Расширение: MOV #<ADDR*400>+PAGE,R0
JSR PC,@140034 (EMT 12)

В младший байт R0 помещается номер подключаемой страницы, в старший — код адреса подключения страницы. Нулевое значение означает область адресов 40000—100000, любое ненулевое — область 100000—140000. В случае нормального завершения бит C в слове состояния процессора сброшен, при ошибке — установлен в «1». В этом случае в байт с адресом 52 (восьмеричное) помещается код ошибки:
- 1 — страница отсутствует;
- 2 — страница не может быть подключена по заданному адресу (попытка подключить страницу ПЗУ по адресу 40000—100000);
- 3 — страница используется базовой операционной системой для своих внутренних целей и запрещена для подключения.

Назначение рабочей страницы .BWORK .
Макрокоманда: .BWORK ARG
Расширение: MOV ARG,R0
JSR,PC,@140036 (EMT 13)
Если бит 15 аргумента равен «0», то происходит назначение рабочей страницы; если бит 7 равен «0», то страница с номером в младшем байте аргумента назначается в качестве рабочей для записи/чтения. Если бит 7 равен «1», то страница назначается в качестве рабочей для макрокоманд .BJSR и .BJMP. Если бит 15 аргумента равен «1», то в R0 помещается код рабочих страниц: в младшем байте номер рабочей страницы чтения-записи, в старшем — номер страницы вызовов (со старшим битом, равным «0»).

Чтение слова из рабочей страницы .BGETW .
Макрокоманда: .BGETW ADDR
Расширение: MOV ADDR,R0
JSR PC,@140040 (EMT 14)
Из рабочей страницы по указанному адресу читается двухбайтное машинное слово и помещается в R0. При попытке прочесть слово из несуществующей страницы R0 не изменяется, а бит C слова состояния процессора устанавливается в «1». Старшие два бита адреса игнорируются.

Чтение байта из рабочей страницы .BGETB .
Макрокоманда: .BGETB ADDR
Расширение: MOV ADDR,R0
JSR PC,@140042 (EMT 15)

Из рабочей страницы по указанному адресу читается байт и помещается в младший байт R0. Старший байт R0 обнуляется . При попытке прочесть байт из несуществующей страницы R0 не изменяется, а бит С слова состояния процессора устанавливается в «1».

Запись слова в рабочую страницу .BPUTW .
Макрокоманда: .BPUTW ADDR,DATA
Расширение: MOV ADDR,R1
MOV DATA,R0
JSR PC,@140044 (EMT 16)

В рабочую страницу по указанному адресу записывается двухбайтное машинное слово. При попытке записи в ПЗУ, несуществующую страницу или системную область ОЗУ устанавливается бит С. (Системной областью считается область системной страницы с относительными адресами от 0 до 7777.)

Запись байта в рабочую страницу .BPUTB .
Макрокоманда: .BPUTB ADDR,DATA
Расширение: MOV ADDR,R1
MOVB DATA,R0
JSR PC,@140046 (EMT 17)

В рабочую страницу по указанному адресу записывается байт. При попытке записи в ПЗУ, несуществующую страницу или системную область ОЗУ устанавливается бит С.

Пересылка массива слов .BMOVE .
Макрокоманда: .BMOVE AREA
Расширение: MOV AREA,R0
JSR PC,@140050 (EMT 20)

Производится пересылка массива слов между текущим адресным пространством пользователя и заданным скрытым пространством. В R0 помещается адрес области параметров.

Формат области параметров:

```
.WORD MEMADR      ;начальный адрес в памяти
.WORD HIDADR      ;начальный адрес в скрытой памяти
.BYTE LOW,HIGH    ;номера страниц скрытой памяти
.WORD BCOUNT      ;счетчик слов
```

Начальные адреса в памяти должны лежать выше адреса 400, конечные адреса (начальный адрес + счетчик) в памяти не должны превышать 160000. Параметры LOW и HIGH задают номера страниц, подключаемых к адресам 40000 и 100000 соответственно. Выходной массив (в который осуществляется запись) не должен лежать в ПЗУ или системной области ОЗУ. Положительное значение счетчика слов задает пересылку из рабочей страницы в память (чтение), отрицательное (при этом используется абсолютное значение счетчика) — обратную операцию (запись). При правильно заданных параметрах и успешной пересылке бит С слова состояния процессора и байт 52 обнуляются. Возможны следующие ошибки (при этом бит С устанавливается в «1» и в байт 52 помещается соответствующий код):

- неверно задан начальный адрес (код 1) — пересылка не производится;
- неверно задан конечный адрес (код 2) — пересылка производится до достижения допустимого конечного адреса, в R0 помещается количество непереданных слов;
- при пересылке возникло прерывание по вектору 4: попытка записи в ПЗУ или несуществующую область (код 3) — пересылка прекращается, в R0 помещается количество непереданных слов.

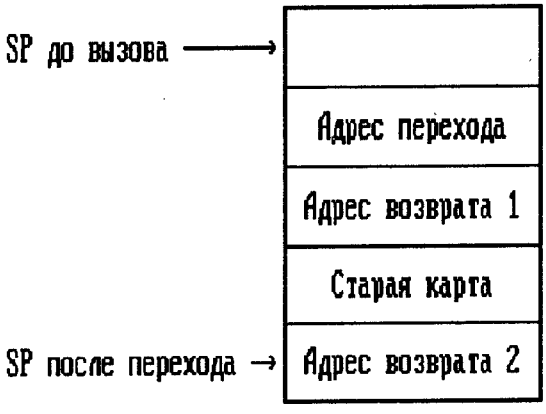
Пересылка массива байтов .BMOVB .
Макрокоманда: .BMOVB AREA
Расширение: MOV AREA,R0
JSR PC,@140052 (EMT 21)

Производится пересылка массива байтов между текущим адресным пространством пользователя и заданным скрытым пространством. Параметры аналогичны пересылке слов (если в тексте описания предыдущей команды заменить «слова» на «байты»).

Вызов подпрограммы из рабочей страницы .BJSR .
Макрокоманда: .BJSR ADDR
Расширение: MOV ADDR,-(SP)
JSR PC,@140054

Рабочая страница подключается по адресу, определенному аргументом ADDR (если он находится в диапазоне 40000—47776, то к 40000; если в диапазоне 100000—140000, то к 100000), и производится вызов командой JSR PC,ADDR.

Регистры общего назначения и слово состояния процессора (кроме бита С) при выполнении вызова не изменяются. Стек не должен находиться в переключаемой странице. В стек помещается следующая информация:



Здесь <адрес перехода> — адрес подпрограммы, заносимый пользователем в стек перед командой JSR, <адрес возврата 1> — адрес возврата в пользовательскую программу, <старая карта> — содержимое регистра управления памятью до вызова, <адрес возврата 2> — адрес возврата в БОС.

При выходе из подпрограммы регистры общего назначения и слово состояния программы сохраняются.

При попытке обращения к подпрограмме в несуществующей странице или системной странице ОЗУ обращение к подпрограмме выполняться не будет и производится выход в монитор с соответствующей диагностикой.

После выхода из подпрограммы восстанавливается старое подключение обеих страниц ОЗУ/ПЗУ, даже если подпрограмма переключала одну или обе страницы. Допускается вложенное использование макрокоманды .BJSR. Параметр вызова из стека удаляется.

Переход в рабочую страницу без возврата .BJMP .
Макрокоманда: .BJMP ADDR
Расширение: MOV ADDR, -(SP)
JSR PC,@140056

Макрокоманда выполняется аналогично .BJSR, но дополнительная информация в стек не помещается, адрес возврата не сохраняется. Если страница не существует или занята системой, параметры вызова удаляются из стека и производится выход в монитор с соответствующей диагностикой.

Примечания.

1. В диагностических сообщениях, выдаваемых по .BJMP, .BJSR и прерыванию по вектору 4, выводится содержимое PC **после** выполнения запроса, т. е. адрес следующей за JSR или недопустимой командой ячейки.
2. В связи с использованием стека, для передачи параметра при вызове следует использовать только команды JSR, но не EMT.

Запросы драйвера клавиатуры

Инициализация драйвера клавиатуры .BKINI .
Макрокоманда: .BKINI
Расширение: JSR PC,@140070 (EMT 30)
Выполняется инициализация драйвера клавиатуры, описанная для макрокоманды .BINIT.

Установка режима драйвера клавиатуры .BKSET .
Макрокоманда: .BKSET MODE
Расширение: MOV MODE,R0
JSR PC,@140072 (EMT 31)

Устанавливается режим драйвера клавиатуры в соответствии со словом MODE. Назначение разрядов слова:

Разряд	Значение
0	«0» — КОИ-8, «1» — КОИ-7
1 (для КОИ-7)	«0» — КОИ-7Н1, «1» — КОИ-7Н2
6	«0» — щелчок при нажатии клавиш разрешен, «1» — запрещен
7	«0» — прямая передача кодов, «1» — перекодировка (см. приложение 1)

Кроме этого, программа пользователя имеет возможность задать подпрограмму завершения при нажатии каждой клавиши. Программным вектором считаются ячейки 110 и 112. При нулевом содержимом ячейки 110 считается, что подпрограмма завершения отсутствует. В ячейку 112 записывается значение регистра управления памятью для подпрограммы завершения или 0 для текущего распределения. При вызове подпрограммы завершения в R0 передается код нажатой клавиши. Выход из подпрограммы выполняется командой RTS PC. В подпрограмме завершения можно использовать все регистры общего назначения.

Если при выходе из подпрограммы завершения бит C установлен в «1», драйвер считает, что символ с клавиатуры не обработан и передает его программе пользователя по .BTTIN. Если же при выходе из подпрограммы завершения бит C сброшен в «0», символ считается обработанным и в буфер не записывается.

Чтение режима работы клавиатуры .PKGET .
Макрокоманда: .BKGET
Расширение: JSR PC,@140074 (EMT 32)

Текущий режим клавиатуры помещается в R0. Формат слова режима см. описание MODE для команды .BKSET.

Ввод символа с ожиданием .BTTIN .
Макрокоманда: .BTTIN
Расширение: JSR PC,@140076 (EMT 33)

В младший байт R0 помещается очередной символ из буфера ввода клавиатуры. Старший байт R0 обнуляется.

Ввод символа без ожидания .BINKEY .
Макрокоманда: .BINKEY
Расширение: JSR PC,@140100 (EMT 34)

В случае отсутствия символов в буфере содержимое регистра R0 обнуляется, а бит C устанавливается в «1». В противном случае бит C сбрасывается в «0», а в R0 передается код очередного символа.

Ввод строки с редактированием .BGLIN .
Макрокоманда: .BGLIN ADDR,LENGTH
Расширение: MOV ADDR,R0
MOV LENGTH,R1
JSR PC,@140102 (EMT 35)

Осуществляется ввод с клавиатуры строки символов с редактированием. Ввод строки заканчивается при нажатии клавиши «ВВОД» («↵»), необязательно в конце строки. Код клавиши «ВВОД» в буфер не помещается, в конце строки добавляется нулевой байт. Параметр ADDR задает адрес буфера, куда помещается строка, LENGHT — максимальную длину вводимой строки. При нулевом значении LENGHT длина строки принимается равной максимально возможной (128 символов). При вводе отрабатывается клавиша «ЗАБОЙ». После возврата в R0 находится адрес байта, следующего за нулевым, а в R1 — реальная длина введенной строки.

Редактирование существующей строки .BEDIT .
Макрокоманда: .BEDIT ADDR,LENGHT
Расширение: MOV ADDR,R0
MOV LENGHT,R1
JSR PC,@140102 (EMT 36)

Строка, находящаяся по адресу ADDR и заканчивающаяся нулевым байтом, выводится на экран. Курсор помещается в конец выведенной строки, после чего осуществляется ее редактирование, аналогичное .BGLIN. Отредактированная строка помещается на место исходной.

Сброс буфера клавиатуры .BKRES .
Макрокоманда: .BKRES
Расширение: JSR PC,@140104 (EMT 37)
Осуществляется инициализация кольцевого буфера клавиатуры.

Установка функциональных клавиш .BSFUN .
Макрокоманда: .BSFUN KEY,ADDR
Расширение: MOV KEY,R0
MOV ADDR,R1
JSR PC,@140106 (EMT 40)

Макрокоманда позволяет присвоить десяти функциональным клавишам (аргумент KEY=1..10 — номер клавиши) текстовые строки, которые будут вводиться при нажатиях этих клавиш. Функциональные клавиши вводятся при одновременном нажатии клавиши «АР2» и соответствующей цифровой (номеру 10 соответствует клавиша «0»). Аргумент KEY, равный 0, вызывает сброс всех функциональных клавиш. Аргумент ADDR задает адрес строки, которая должна заканчиваться нулевым байтом. Максимальная длина присваиваемой строки — 64.

Чтение функциональных клавиш .BGFUN .
Макрокоманда: .BGFUN KEY,ADDR
Расширение: MOV KEY,R0
MOV ADDR,R1
JSR PC,@140110 (EMT 41)

Производится пересылка строки, соответствующей клавише с номером KEY, в пользовательскую область по адресу ADDR.

(Окончание следует.)

Внимание! Опечатка

В первой части «Руководства программиста БК-0011(М)», опубликованной в №4 за 1995 г., на с. 25 (четвертый абзац сверху) нижние индексы, означающие восьмеричную систему счисления, ошибочно напечатаны как цифры единичных разрядов чисел. Текст данного абзаца следует читать:

«ОЗУ буфера экрана считается «замкнутым в кольцо». Экран состоит из 256 строк по 64 байта в каждой. Строки начинаются с адресов, кратных 100_8. Первый байт каждой строки расположен в ее левом конце, байты отображаются начиная с младшего бита. Смещение задается в строках, т. е. изменение кода смещения на 1 приводит к кольцевому сдвигу информации на экране на одну строку растра. Исходному состоянию (когда первый байт буфера экрана отображается в самой верхней строке растра) соответствует значение смещения 330_8. Увеличение значения соответствует сдвигу информации вверх, уменьшение — вниз.»

* * *

В статье С. К. Румянцева «Наклонные эллипсы в БЕЙСИКе БК-0010.01» (№3 за 1995 г., с. 21) строка 1130 листинга должна выглядеть следующим образом:

```
1130  LINE (X0-X1,Y0-Y)-(X0-X2,Y0-Y),C2
```

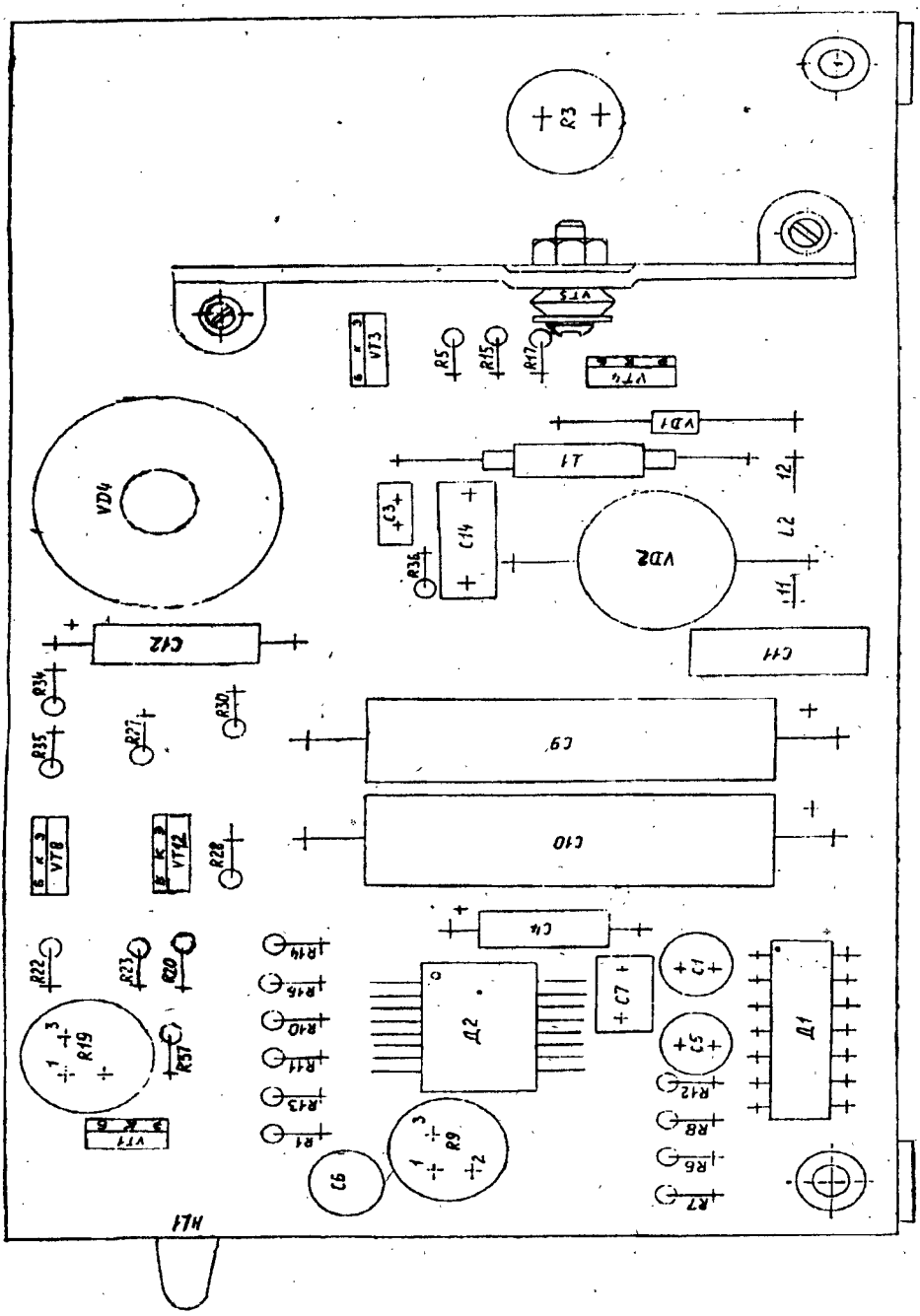
Аналогично в конце строк 1160—1180 необходимо добавить после закрывающей скобки символы ,C1.

* * *

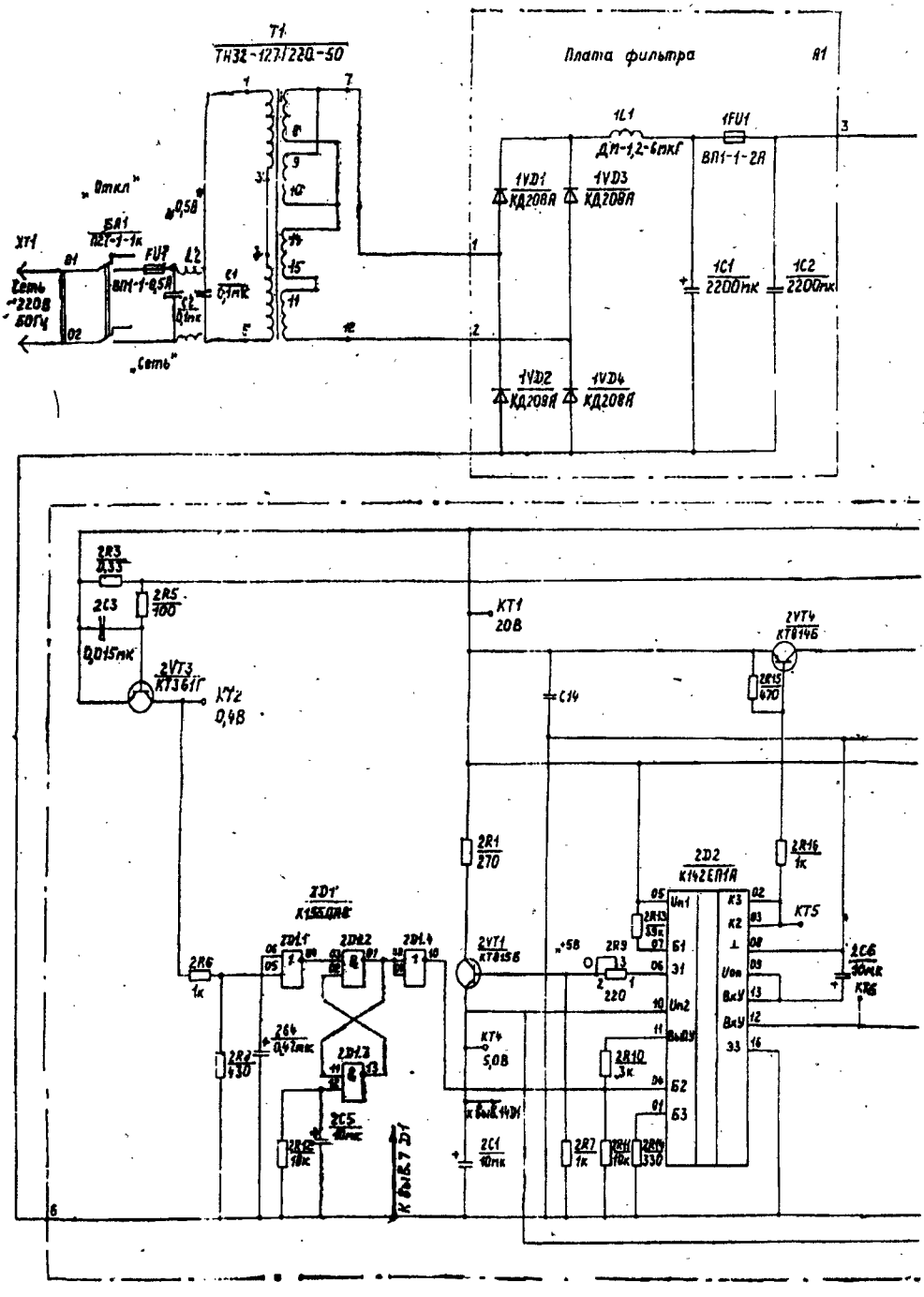
В справочном листке (№4 за 1994 г., с.28) в первой колонке нижней графы таблицы функций, отрабатываемых БК-0010 при замыкании цепей X—Y, ошибочно указан номер цепи Y8. Правильное обозначение — Y9.

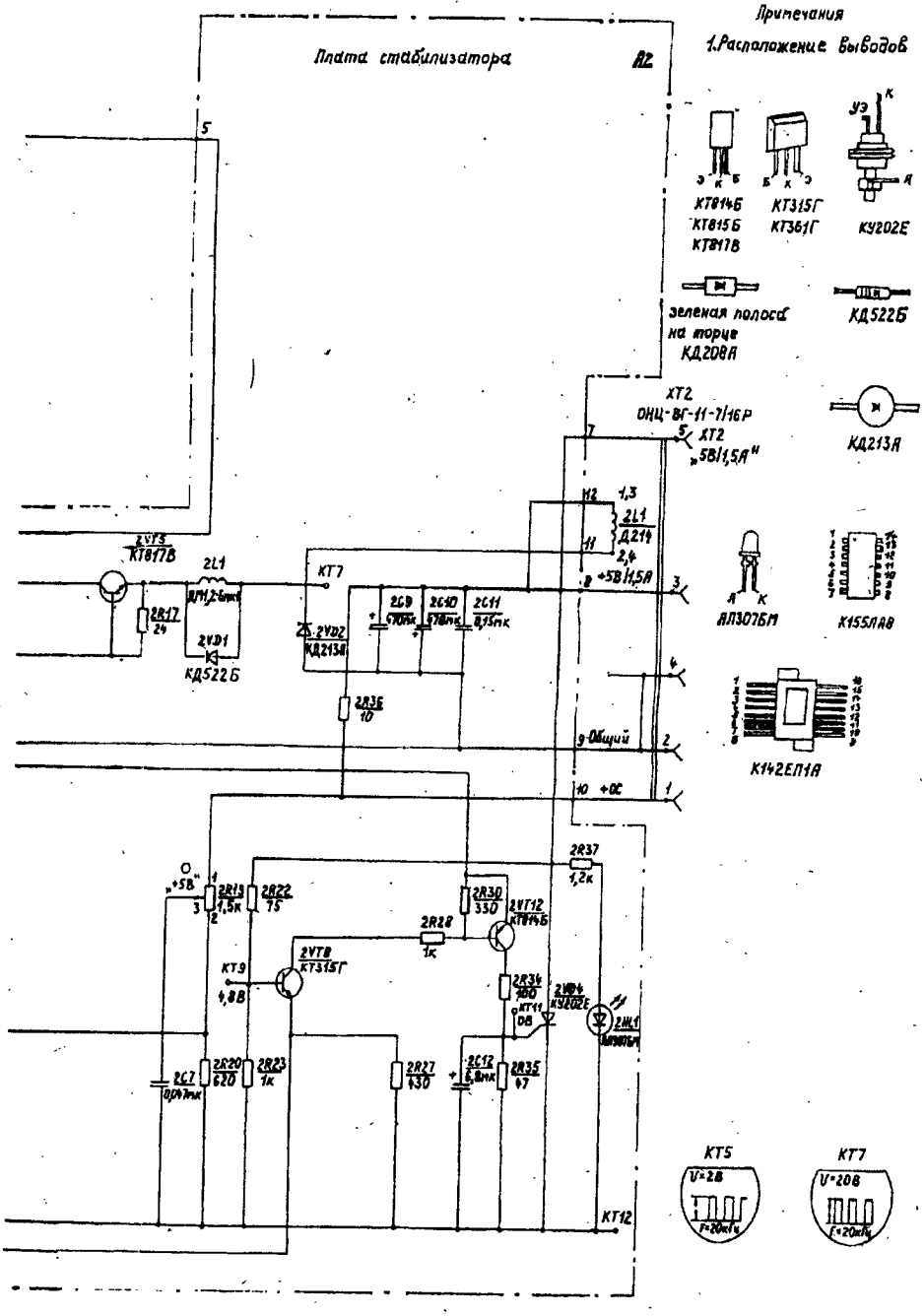
СПРАВОЧНЫЙ ЛИСТОК

Типовой блок питания для БК-0010(.01) и БК-0011(М)

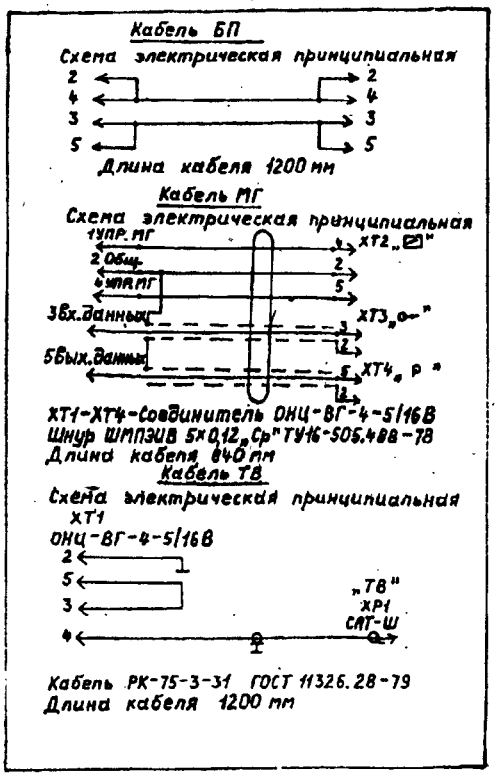


Плата стабилизатора


Плата фильтра
A1
T1
ТН32-127/220-50
1L1
ДМ-1,2-6мкГ
1FU1
ВП1-1-2А
1VD1 КД208А
1VD3 КД208А
1VD2 КД208А
1VD4 КД208А
1C1 2200мк
1C2 2200мк
SA1
FU?
ВП1-1-0,5А
XT1
Сеть ~220В 50Гц
„Откл"
„Сеть"
C1 0,1мк
C2 0,1мк
2R3 0,33
2R5 100
2C3 0,015мк
2VT3 КТ361Г
КТ2 0,4В
КТ1 20В
2VT4 КТ814Б
2R15 470
C14
2R1 270
2D1 К155ЛА6
2D2 К142ЕН1А
2R16 1к
КТ5
2C6 30мк К16
2R6 1к
2R2 430
2C4 0,47мк
2C5 40мк
2R12 10к
2VT1 КТ815Б
„+5В
2R9 220
КТ4 5,0В
2R13 33к
2R10 3к
2R7 1к
2R11 10к
2R14 330
2C1 10мк
К 6,8,7 D1
6


Плата стабилизатора
A2
Примечания
1.Расположение выводов
КТ814Б
КТ815Б
КТ817В
КТ315Г
КТ361Г
КУ202Е
Зеленая полоса на торце
КД208А
КД522Б
КД213А
ХТ2
ОНЦ-ВГ-11-7/16Р
ХТ2
"5В1,5А"
АЛ307БМ
К155ЛА8
К142ЕП1А
КТ5
КТ7
КТ12
+5В1,5А
Общий
+С
+5В
2VT5
КТ817В
2L1
2R17
24
2VD1
КД522Б
2VD2
КД213А
2C9
2C10
2C11
2R36
10
2R37
1,2к
2R13
1,5к
2R22
75
2R30
330
2VT12
КТ814Б
2R28
1к
2VT8
КТ315Г
КТ9
2R34
360
2VD4
КУ202Е
2HL1
2C7
2R20
620
2R23
1к
2R27
430
2C12
2R35
67
КТ5
U=2В
КТ7
U=20В

Распайка блока нагрузок и кабелей, входящих в комплект БК-0010(.01) и БК-0011(М)



Блок нагрузок
Таблица соединений

Откуда идет	Куда поступает
XT1/A16	XT1/B24
XT1/A13	XT1/A24
XT1/B12	XT1/B23
XT1/B10	XT1/B17
XT1/B5	XT1/B20
XT1/B7	XT1/A20
XT1/B6	XT1/B22
XT1/A7	XT1/A23
XT1/A28	XT1/B31
XT1/B28	XT1/A31
XT1/A27	XT1/B32
XT1/B27	XT1/A32
XT1/A26	XT1/B30
XT1/B26	XT1/A29
XT1/A25	XT1/B29
XT1/B25	XT1/A30

Данные материалы предоставлены редакции для публикации С. М. Неробеевым и А. В. Сорокиным.

Нам пишут

В «Справочном листке», напечатанном в журнале «Персональный компьютер БК-0010 — БК-0011М» №3 за 1994 г. была приведена таблица команд ассемблера, их кодов, методов адресации и т. д. (автор А. Конюшенко). В списке методов адресации (стр. 63) с кодами 2—5 указано, что автоматический инкремент или декремент (в зависимости от команды) производится на 1 или на 2, в зависимости от того, производится ли операция с отдельными байтами или с двухбайтными машинными словами. Но при этом не отмечен тот факт, что при работе со стеком (адресация через регистр SP) автоинкремент и автодекремент во всех случаях производятся только на 2, независимо от байтового или словного типа команды.

В. С. Ивашков,
г. Полтава

Сегодня значительное количество программ для БК использует собственный шрифт вместо стандартного. Делается это по разным причинам: для улучшения внешнего оформления текстовой информации, повышения скорости вывода текста на экран или других специфических целей. Ниже рассказывается о том, как при помощи несложных программ на ассемблере реализовать на БК шрифты нестандартного вида.

В. В. Юров, Д. Ю. Усенков,
Москва

Нестандартные шрифты на ассемблере

Реализовать в своей программе новый шрифт можно двумя способами: нарисовав каждый символ по точкам и подсоединив к программе или написав специальные подпрограммы нестандартного вывода типового шрифта БК, зашитого в ПЗУ. Но рисованные шрифты требуют довольно большого объема памяти и поэтому используются в основном в играх для вывода коротких сообщений (тогда можно по крайней мере заготавливать изображения не для всех символов БК). Иногда потребность в рисованном шрифте возникает и в других случаях, чаще всего для последующего вывода текста на принтер. В системных же и пользовательских программах, где требуется экономия свободной памяти и максимальное быстродействие, лучше всего для выделения фрагментов текста и увеличения скорости его вывода на экран использовать специальные подпрограммы, выводящие зашитый в ПЗУ БК шрифт с некоторыми изменениями. Впрочем, у каждого из этих двух способов свои преимущества и недостатки, поэтому однозначно сказать, какой из них лучше, нельзя. Выбор конкретного метода — задача для разработчиков той или иной программы. Чтобы помочь им, опишем основные «плюсы» и «минусы» обоих подходов.

Как уже отмечалось, основной недостаток рисованного шрифта — большой объем занимаемой им памяти. Оценить этот объем можно путем несложных расчетов. Обычный размер символов рисованного шрифта — 8·8 цветных точек (16 байт на каждый символ), при этом в программе используются буквы (32 символа), цифры (10 символов) и знаки препинания (достаточно четырех). Итого требуется 16·(32+10+4)=736 (или 1340$_8$) байт для всего шрифта и еще около 60 байт для программы его вывода. Если хранить в ОЗУ черно-белый шрифт, а при выводе преобразовывать его в цветной, то это потребует почти вдвое меньшего объема памяти. В играх на ассемблере еще удается найти возможность разместить такой шрифт. Однако для системных и пользовательских программ необходим весь или почти весь набор знаков, т. е. 192 черно-белых символа размером 8·10 (10 байт на символ). А это уже 192·10=1920 байт — почти 2 Кб! Такие траты слишком велики даже при работе с ассемблером, а для программ на БЕЙСИКе они и вовсе неприемлемы.

Другой способ — нестандартный вывод типового шрифта из ПЗУ — с успехом используется не только в системных и пользовательских программах, но и в играх. Можно придумать множество алгоритмов получения шрифта необычного вида: от утолщенного до разноцветного с тенями. Причем расход памяти при этом не столь уж и велик, например в ANIMATICe v3.1 используется пять разных шрифтов, а программа их вывода занимает около 100 байт.

Все сказанное выше относилось к обоим типам компьютеров — БК-0010(.01) и БК-0011(М). На одиннадцатой модели имеется и еще одна многообещающая возможность: без затрат памяти допускается использовать в программах нестандартный шрифт размером 8·10 черно-белых точек, выводимый как в черно-белом режиме, так и в цветном. Дело в том, что на БК-0011(М) в режиме эмуляции БК-0010(.01) к адресам 100000—137777 подключается ОЗУ, куда загружается драйвер-мониторная система БК-0010 и дисковая операционная система. Внутри «десяточного» монитора хранится стандартный шрифт БК-0010(.01), который можно заменить любым другим рисованным шрифтом. Для БК-0011(М) уже созданы специальные утилиты, позволяющие перед загрузкой своей программы (или из самой этой программы)

заменять «десяточный» шрифт на любой вам необходимый. Это частично решает проблему организации собственного шрифта в программах, но и у этого метода имеются свои недостатки: программа будет работать только на БК-0011(М), к тому же нельзя создавать многоцветный шрифт или шрифт других размеров. Поэтому лучше всего использовать метод нестандартного вывода типового шрифта из ПЗУ БК-0010(.01), чем мы и займемся далее.

Ниже рассмотрены основные разновидности шрифтов, которые можно получить из «стандартного» шрифта, зашитого в ПЗУ. Поскольку обязательное требование к современным программам — достаточно высокое быстродействие, в качестве языка программирования выбран ассемблер.

Для реализации предлагаемых алгоритмов необходимо знать механизм вывода символов на экран БК, используемый стандартными функциями монитора (например, EMT 16). Экранное ОЗУ занимает адреса 40000$_8$—77777$_8$, по 100$_8$ байт на одну графическую строку. Это позволяет кодировать 512x256 черно-белых или 256·256 цветных точек. Один байт кодирует 8 черно-белых или 4 цветные точки. Левой верхней точке (с координатами (0,0)) белого цвета соответствует установленный в «1» нулевой бит в байте с адресом 40000$_8$ в черно-белом режиме или установленные в «1» нулевой и первый биты того же байта для точки красного цвета — в цветном режиме. Правой нижней же точке белого цвета соответствует установленный в «1» седьмой бит в байте с адресом 77777$_8$ в черно-белом режиме или установленные в «1» шестой и седьмой биты того же байта для точки красного цвета — в цветном режиме. Изображения символов (так называемый «знакогенератор») с кодами 32—63 хранятся в ПЗУ по адресам 112276$_8$—112775$_8$, латинских заглавных букв — 112776$_8$—113475$_8$, латинских строчных — 113476$_8$—114175$_8$, полуграфики — 114176$_8$—114675$_8$, русских строчных букв — 114676$_8$—115375$_8$, русских заглавных — 115376$_8$—116076$_8$. Каждый символ изображен в черно-белом режиме 8·10 точек, т. е. изображение одного символа занимает 10 байт. (Дополнительно по адресам 112050$_8$—112274$_8$ содержатся выводимые в режиме БЛОК РЕД знаки редактирующих клавиш, а по адресам 112036$_8$—112046$_8$ — неизвестно для чего используемое изображение «инверсного Р».)

Соответствующие каждому символу 10 байтов, в каждом из которых единичные биты соответствуют точкам изображения символа, а нулевые — точкам фона, записаны в ПЗУ «пачка за пачкой» в порядке возрастания кодов символов от 32 («ПРОБЕЛ», адрес в ПЗУ 112276$_8$) до 255 («Ъ») с пропуском кодов, соответствующих командам управления дисплеем (т. е. после 10 байтов символа с кодом 127 вплотную следуют байты символа «Л» с кодом 160).

Стандартный драйвер монитора для вывода каждого очередного символа выполняет приблизительно такую последовательность действий.

1. Преобразование кода символа и вычисление адреса начала его изображения в прошивке ПЗУ (отдельно учитываются режимы БЛОК РЕД и ИНД СУ, здесь мы их не рассматриваем). Если код символа больше 160 (240$_8$), из него вычитается число 32 (40$_8$), таким образом учитывается отсутствие в ПЗУ прошивки для кодов команд управления выводом на дисплей. Затем из полученного кода вновь вычитается 32 (40$_8$), результат умножается на 10 и к полученному произведению прибавляется адрес начала прошивки для пробела (112276$_8$).

2. Определение адреса экранного ОЗУ, соответствующего текущей знаковой позиции (с учетом рулонного смещения экрана, наличия режима РП и т. д.).

3. Извлечение из ПЗУ (по вычисленному на шаге 1 адресу) первого байта изображения символа.

4. При черно-белом режиме вывода на экран (64 символа в строке) извлеченный из ПЗУ байт копируется в текущую ячейку ОЗУ экрана. В цветном режиме дополнительно производится «распаковка» байта в машинное слово с заменой единичных и нулевых битов «шаблонного» байта на пары битов в зависимости от установленных цветов переднего плана и фона соответственно, после чего полученное двухбайтное слово выводится в пару байт экранного ОЗУ.

5. Выбирается следующий по порядку байт прошивки символа в ПЗУ и вычисляется адрес очередной ячейки ОЗУ экрана (по формуле **<старый адрес>−100$_8$**, так как байты изображения символа должны выводиться один под другим в виде вертикальной «стопки»). Дополнительно учитывается рулонное смещение, режим РП и т. д., так что реально верхняя часть символа может быть выведена в старшие адреса экранного ОЗУ, а нижняя — в младшие.

6. Шаги 3—5 повторяются 10 раз, после чего рассчитываются новые значения координат текущей знаковой позиции и программа готова к выводу следующего символа.

(Кстати, из рассмотренного алгоритма следует, что прошитая в ПЗУ монитора БК-0010(.01) подпрограмма вывода символов может быть использована и для вывода заданных пользователем небольших спрайтов (8·10 точек). Как это сделать, рассказано в журнале «Вычислительная техника и ее применение», №8 за 1990 г., с. 42.)

После подробного теоретического вступления перейдем к программированию. Ниже приведены подпрограммы, написанные на ассемблере стандарта М18 и демонстрирующие вывод текста пятнадцатью различными шрифтами. Подпрограммы перемещаемы и не используют вызовов EMT. Основная программа должна реализовывать функции очистки экрана, вывода текста, ожидания нажатия клавиши и др., где функция вывода текста, в свою очередь, вызывает какую-либо из подпрограмм вывода символов. В последних в качестве входных данных используются следующие регистры процессора и ячейки памяти:

R1 — адрес первого байта прошивки в ПЗУ требуемого символа стандартного шрифта шрифта;

R3 — адрес в ОЗУ экрана для вывода измененного шрифта (верхний левый байт знакоместа);

CLRCHR — цвет символа;

CL2CHR — второй цвет символа.

В качестве демонстрационной программы можно использовать следующий ассемблерный листинг, к которому пристыковывается текст одной из приведенных далее подпрограмм. Думается, читатели смогут самостоятельно написать на основе этой «демонстрашки» программы вывода текста с учетом конкретных требований.

```
; Пример основной программы
;  для вызова подпрограмм
;   генерации нестандартных шрифтов
;
START:  EMT 14              ; очистка экрана
        MOV #50004,ADR      ; адрес экрана, куда выводить текст
        MOV #TXT,R1         ; адрес начала выводимого текста
        CLR R2              ;  (текст оканчивается нулевым байтом)
LOOP:   TSTB @R1            ; если нулевой байт, то
        BEQ ENDOUT          ;  закончить вывод текста
        JSR PC,CHROUT       ; вывод очередного символа
        INC R1              ; следующий символ
        INC ADR             ; следующий байт экранного ОЗУ
        BR LOOP             ;  и на продолжение цикла
ENDOUT: EMT 6               ; ожидание нажатия клавиши
        HALT                ; останов программы
ADR:    .#0
TXT:    .A: ПРОБА текста proba TEXTS 123!#% ♦♥♣♠π.
        .E
;
CHROUT: MOVB @R1,R0
        CMPB R0,#240        ; код меньше 240 ("π ")
        BLO 1               ; вычисление начала изображения
        SUB #40,R0          ;  символа, код которого в R0,
1:      SUB #40,R0          ;  в прошивке ПЗУ монитора
        BIC #177400,R0      ; обнуление старшего байта
                            ; умножение на 10 (дес.):
        ASL R0              ;  R0*2
        MOV R0,-(SP)        ;
        ASL R0              ;  R0*4
        ASL R0              ;  R0*8
        ADD (SP)+,R0        ;  R0*8+R0*2 = R0*10
        ADD #112276,R0      ; R0 <- адрес изображения символа
        MOV R5,-(SP)        ; сохранение
        MOV R4,-(SP)        ;  содержимого
        JSR R4,@#110346     ;   регистров
        MOV R0,R1           ;
        MOV ADR,R3          ;
        MOV #177777,CLRCHR  ; цвет символов
        MOV #125252,CL2CHR  ; второй цвет символов
```

```
        JSR PC,BEGIN        ; вызов испытываемой подпрограммы
        JSR R4,@#110362     ; восстановление
        MOV (SP)+,R4        ;  содержимого
        MOV (SP)+,R5        ;   регистров
        RTS PC              ; возврат
CLRCHR: .#0
CL2CHR: .#0
;
; Далее должна следовать испытываемая подпрограмма
;  нестандартного вывода шрифта
;  (начальная метка - BEGIN)
;
```

Во всех подпрограммах учитывается, что шрифт БК имеет высоту 10 точек, из которых буквами и цифрами используются только верхние девять, причем первая — только заглавной буквой «Й», а последняя (десятая) — только знаками полуграфики. Кроме того, нужно помнить, что рулонное смещение экрана в них не учитывается, следовательно, основная программа должна его нормализовать.

Следует отметить также, что все приведенные ниже подпрограммы годятся и для вывода на экран собственных рисованных шрифтов, если передавать в R1 адрес «пакета» из 10 байт изображения символа («прямое» начертание, режим 64 символа в строке). Если же собственный шрифт «широкий» (т. е. каждому символу соответствуют 10 двухбайтных машинных слов), подпрограммы, рассчитанные на вывод измененного ПЗУшного шрифта в режиме 64 символа в строке, можно адаптировать путем замены байтовых команд словными (например, ROL вместо ROLB). Кроме того, на БК-0011(М) можно загрузить в ОЗУ, где размещена копия «десяточного» монитора, собственный рисованный шрифт (как говорилось в начале статьи), и тогда все приведенные ниже подпрограммы на БК-0011(М) в режиме эмуляции «десятки» можно настроить на работу с ним.

Жирный шрифт

Это самый простой и наиболее часто используемый шрифт, обычно для выделения какой-либо информации на экране. Для получения жирного шрифта необходимо каждую точку символа повторить сбоку (обычно справа). Режим вывода символов — как при 64 символах в строке.

```
BEGIN:  MOV  #12,R2      ;Высота символа
1:      MOVB (R1),(R3)   ;Рисуется строчка стандартного символа
        ROLB (R3)        ;Все точки сдвигаются вправо
        BISB (R1)+,(R3)  ;Накладывается строчка стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке,
        SOB  R2,1        ;если эта не последняя
        RTS  PC
```

Жирный снизу

Алгоритм получения этого шрифта практически не отличается от предыдущего, но здесь верхние пять строк символа рисуются без изменений, а нижние пять — с утолщением вправо. Режим вывода — 64 символа в строке.

```
BEGIN:  MOV  #5,R2       ;Высота неизменяемой части символа
1:      MOVB (R1)+,(R3)  ;Рисуется строчка стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке,
        SOB  R2,1        ;если эта не последняя
        MOV  #5,R2       ;Высота жирной части символа
2:      MOVB (R1),(R3)   ;Рисуется строчка стандартного символа
        ROLB (R3)        ;Все точки сдвигаются вправо
        BISB (R1)+,(R3)  ;Накладывается строчка стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке,
        SOB  R2,2        ;если эта не последняя
        RTS  PC
```

Жирный слева

Для получения символов, жирных слева, необходимо левые четыре точки каждой строки стандартного шрифта сдвинуть вправо и сложить с исходной строкой. Режим вывода — 64 символа в строке.

```
BEGIN:  MOV   #12,R2       ;Высота шрифта
1:      MOVB  (R1),(R3)    ;Рисуется строчка стандартного символа
        BICB  #360,(R3)    ;Правая половина стирается
        ROLB  (R3)         ;Оставшиеся точки сдвигаются вправо
        BISB  (R1)+,(R3)   ;Накладывается строчка стандартного символа
        ADD   #100,R3      ;Переход к следующей строчке,
        SOB   R2,1         ;если эта не последняя
        RTS   PC
```

Жирный справа

Отличие этого шрифта от жирного слева заключается только в том, что сдвигать нужно четыре правые точки влево. Режим вывода — 64 символа в строке.

```
BEGIN:  MOV   #12,R2       ;Высота шрифта
1:      MOVB  (R1),(R3)    ;Рисуется строчка стандартного символа
        BICB  #17,(R3)     ;Левая половина стирается
        RORB  (R3)         ;Оставшиеся точки сдвигаются влево
        BISB  (R1)+,(R3)   ;Накладывается строчка стандартного символа
        ADD   #100,R3      ;Переход к следующей строчке,
        SOB   R2,1         ;если эта не последняя
        RTS   PC
```

Курсив (наклонный вправо)

Для наклона шрифта вправо следует верхние три строки стандартного шрифта рисовать со сдвигом вправо, следующие три — без сдвига, а последние четыре — со сдвигом влево. Режим вывода — 64 символа в строке.

```
BEGIN:  MOV   #3,R2        ;Количество строк, сдвигаемых вправо
1:      MOVB  (R1)+,(R3)   ;Рисуется строчка стандартного символа
        ROLB  (R3)         ;Все точки сдвигаются вправо
        ADD   #100,R3      ;Переход к следующей строчке,
        SOB   R2,1         ;если эта не последняя
        MOV   #3,R2        ;Количество строк, остающихся без изменений
2:      MOVB  (R1)+,(R3)   ;Рисуется строчка стандартного символа
        ADD   #100,R3      ;Переход к следующей строчке,
        SOB   R2,2         ;если эта не последняя
        MOV   #4,R2        ;Количество строк, сдвигаемых влево
3:      MOVB  (R1)+,(R3)   ;Рисуется строчка стандартного символа
        RORB  (R3)         ;Все точки сдвигаются влево
        ADD   #100,R3      ;Переход к следующей строчке,
        SOB   R2,3         ;если эта не последняя
        RTS   PC
```

«Обратный» курсив (наклонный влево)

Отличие шрифта наклонного влево от наклонного вправо состоит в том, что сначала нужно сдвигать строки влево, а в конце — вправо. Режим вывода — 64 символа в строке.

```
BEGIN:  MOV   #3,R2        ;Количество строк, сдвигаемых влево
1:      MOVB  (R1)+,(R3)   ;Рисуется строчка стандартного символа
        RORB  (R3)         ;Все точки сдвигаются влево
        ADD   #100,R3      ;Переход к следующей строчке,
        SOB   R2,1         ;если эта не последняя
```

```
        MOV  #3,R2       ;Количество строк, остающихся без изменений
2:      MOVB (R1)+,(R3)  ;Рисуется строчка стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке,
        SOB  R2,2        ;если эта не последняя
        MOV  #4,R2       ;Количество строк, сдвигаемых вправо
3:      MOVB (R1)+,(R3)  ;Рисуется строчка стандартного символа
        ROLB (R3)        ;Все точки сдвигаются вправо
        ADD  #100,R3     ;Переход к следующей точке,
        SOB  R2,3        ;если эта не последняя
        RTS  PC
```

Высокий шрифт

Этот шрифт удобно использовать в текстовых редакторах, например, как в VorteX! 1.05 Д. Романова. Высота такого шрифта составляет 12 точек. Для его получения следует первую и седьмую строку повторить дважды. Режим вывода — 64 символа в строке.

```
BEGIN:  MOVB (R1),(R3)   ;Рисуется первая строчка стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке
        MOV  #6,R2       ;Количество строчек до второго
                         ;повтора, включая первую строчку
1:      MOVB (R1)+,(R3)  ;Рисуется строчка стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке,
        SOB  R2,1        ;если эта не последняя
        MOVB (R1),(R3)   ;Рисуется седьмая строчка стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке
        MOV  #4,R2       ;Количество строчек до десятой,
                         ;включая седьмую
2:      MOVB (R1)+,(R3)  ;Рисуется строчка стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке,
        SOB  R2,2        ;если эта не последняя
        RTS PC
```

Шрифт двойной высоты

Шрифт двойной высоты удобно использовать в заголовках, как, например, в HCOPY6 и документации к ANIMATIC. Высота такого шрифта, естественно, равна 20 точкам. Есть две его разновидности: сплошной и «прерывистый». В первом случае каждую строку стандартного шрифта необходимо повторять дважды, а во втором — выводить строки стандартного шрифта на экран через одну. Наиболее приятно смотрится шрифт, полученный по второму способу (в комментариях даны изменения для первого способа). Режим вывода — 64 символа в строке.

```
BEGIN:  MOV  #12,R2      ;Высота шрифта
1:      MOVB (R1)+,(R3)  ; MOVB (R1),(R3) ;Рисуется строчка
                         ;стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке
        CLRB (R3)        ; MOVB (R1)+,(R3) ;Оставляется пустая
                         ;строчка  или рисуется строчка
                         ;стандартного символа
        ADD  #100,R3     ;Переход к следующей строчке,
        SOB  R2,1        ;если эта не последняя
        RTS PC
```

Контурный шрифт

Это наиболее сложный шрифт, но и самый красивый. Существуют две его разновидности: «округлый» и «угловатый». Высота шрифта равна 12 точкам. Суть метода получения обеих разновидностей состоит в обведении каждой точки и ее последующем стирании. В первом случае каждая точка обводится сверху, снизу, слева и справа, а во втором — еще и по углам. «Угловатый» шрифт смотрится лучше, для него и предназначена ассемблерная подпрограмма, листинг которой приведен ниже (для получения «округлого» шрифта строки, отмеченные звездочкой, следует удалить). Режим вывода — 64 символа в строке.

```
BEGIN:  MOV   #12,R2        ;Высота шрифта
        CLR   (R3)          ;Первая строчка пока оставляется пустой
        ADD   #100,R3       ;Переход к следующей строчке
        MOV   R1,-(SP)      ;Сохранение адресов
        MOV   R3,-(SP)      ;в стеке
1:      MOVB  (R1),R5       ;Строчка стандартного символа
                            ;запоминается в регистре
        BISB  R5,-100(R3)   ;Запомненная строчка накладывается
                            ;на строчку выше
        MOVB  R5,100(R3)    ;Запомненная строчка рисуется строчкой ниже
        RORB  R5            ;Все точки запомненной строчки
                            ;сдвигаются влево
        BISB  R5,-100(R3)   ;* Сдвинутая влево строчка
                            ;накладывается на строчку выше
        MOVB  R5,(R3)       ;Сдвинутая влево строчка рисуется
                            ;в текущей строчке
        BISB  R5,100(R3)    ;* Сдвинутая влево строчка
                            ;накладывается на строчку ниже
        MOVB  (R1)+,R5      ;Восстанавливается запомненная в
                            ;регистре строчка
        ROLB  R5            ;Восстановленная строчка сдвигается вправо
        BISB  R5,-100(R3)   ;* Сдвинутая вправо строчка
                            ;накладывается на строчку выше
        BISB  R5,(R3)       ;Сдвинутая вправо строчка
                            ;накладывается на текущую строчку
        BISB  R5,100(R3)    ;* Сдвинутая вправо строчка
                            ;накладывается на строчку ниже
        ADD   #100,R3       ;Переход к следующей строчке
        SOB   R2,1          ;Повтор, если эта не последняя строчка
        MOV   #12,R2        ;Высота шрифта
        MOV   (SP)+,R3      ;Восстановить адреса
        MOV   (SP)+,R1      ;из стека
2:      BICB  (R1)+,(R3)    ;Стереть "внутренние" точки
        ADD   #100,R3       ;Повтор, если эта строчка
        SOB   R2,2          ;не последняя
        RTS   PC
```

Инверсный шрифт

Этот шрифт обычно используется в меню, для указания активного элемента списка, или для других целей. Алгоритм его получения очень прост: белые точки сменяются черными, а черные — белыми. Режим вывода — 64 символа в строке.

```
BEGIN:  MOV   #12,R2        ;Высота шрифта
1:      MOVB  (R1)+,(R3)    ;Рисуется строчка стандартного символа
        COMB  (R3)          ;Инвертируется строчка
        ADD   #100,R3       ;Переход к следующей строчке,
        SOB   R2,1          ;если эта не последняя
        RTS PC
```

Подчеркнутый шрифт

Подчеркнутый шрифт, как правило, применяют для незначительного выделения текста. Для этого последнюю (десятую) строку шрифта нужно заменить горизонтальной линией длиной в 8 точек (байт #377) или, что еще лучше, инвертировать ее. Режим вывода — 64 символа в строке.

```
BEGIN:  MOV   #11,R2        ;Количество строчек до подчеркиваемой
1:      MOVB  (R1)+,(R3)    ;Рисуется строчка стандартного символа
        ADD   #100,R3       ;Переход к следующей строчке,
        SOB   R2,1          ;если эта не последняя
        MOVB  (R1)+,(R3)    ;Рисуется последняя строчка
```

```
        COMB (R3)       ;Инвертируется последняя строчка
        RTS PC
```

Вообще же эта подпрограмма, как и предыдущая, вряд ли будет использоваться вами достаточно часто, так как в БК имеются аналогичные стандартные режимы. Но для общего понимания эти подпрограммы, безусловно, полезны.

Узкий цветной шрифт

К сожалению, на БК режим 64 символа в строке (разрешение экрана 512x240 графических точек) для цветного монитора неудобен из-за пестроты получающегося текста, а 32 символов часто бывает недостаточно для отображения в строке всей необходимой информации. И разумеется, хотелось бы, чтобы программа работала на цветном мониторе не хуже, чем на черно-белом. Можно преобразовать стандартный шрифт шириной 8 точек в узкий цветной шириной 4 точки, правда, с некоторой потерей читаемости текста. Для этого необходимо все четные точки строки символа сместить влево, а нечетные — вправо и сложить их с исходной строкой. Цвет символа кодируется в ячейке CLRCHR, или его можно брать из системной ячейки с адресом 214_8.

```
BEGIN:  MOVB #12,R2     ;Высота шрифта
        MOVB CLRCHR,R0  ;Цвет символа запоминается в регистре
        COMB R0         ;Для дальнейшего использования
                        ;цвет инвертируется
1:      MOVB (R1),R5    ;Строчка стандартного символа
                        ;запоминается в регистре
        MOVB R5,(R3)    ;Рисуется запомненная строчка
        BICB #125,R5    ;Стираются все нечетные точки
        CLC             ;запомненной строчки
        RORB R5         ;Все четные точки запомненной строчки
                        ;сдвигаются влево
        BISB R5,(R3)    ;Сдвинутые влево точки накладываются
                        ;на текущую строчку
        MOVB (R1)+,R5   ;Строчка стандартного символа
                        ;снова запоминается в регистре
        BICB #252,R5    ;Стираются все четные точки
        CLC             ;запомненной строчки
        ROLB R5         ;Все нечетные точки запомненной строчки
                        ;сдвигаются вправо
        BISB R5,(R3)    ;Сдвинутые вправо точки накладываются
                        ;на текущую строку
        BICB R0,(R3)    ;Строчка окрашивается в заданный цвет
        ADD  #100,R3    ;Переход к следующей строчке,
        SOB  R2,1       ;если эта не последняя
        RTS  PC
```

* * *

Две подпрограммы (см. разделы «Широкий цветной шрифт» и «Широкий разноцветный шрифт») предназначены для вывода широкого шрифта (в режиме 32 символа в строке), поэтому необходимо в основной (испытательной) программе заменить оператор перехода к следующему байту экранного ОЗУ на переход к следующему двухбайтному слову (ADD #2,ADR), а пробный текст (метка TXT) немного укоротить. Ниже показан измененный фрагмент листинга испытательной программы.

```
LOOP:   TSTB @R1
        BEQ ENDOUT
        JSR PC,CHROUT
        INC R1
        ADD #2,ADR  ; следующее слово экранного ОЗУ
        BR LOOP
ENDOUT: EMT 6
        HALT
```

```
ADR:    .#0
TXT:    .A: ПРОБА текста proba .
        .E
;
```

Широкий цветной шрифт

Это обычный стандартный шрифт в режиме 32 символа в строке. Данная подпрограмма служит не столько практическим целям, сколько для демонстрации механизма «распаковки» прошитого в ПЗУ «узкого» монохромного шрифта в цветной широкий. Для «распаковки» необходимо каждый бит исходного байта изображения символа заменить парой битов (11, 10 или 01 в соответствии с маской цвета). При этом ширина символов увеличивается вдвое (до 16 черно-белых точек).

```
BEGIN:  MOV #12,R2          ;Высота шрифта
        MOV  CLRCHR,-(SP)   ;Цвет символа запоминается в стеке
        COM (SP)            ;Для дальнейшего использования
                            ;цвет инвертируется
1:      CLR R0              ;Очистка буферного регистра
        MOVB (R1)+,R5       ;Строчка стандартного символа
                            ;запоминается в регистре
        MOV #10,R4          ;Количество битов в байте
2:      ROLB R5             ;Сдвиг очередного бита исходного
                            ;байта в бит C
        BCC 3               ;Бит был нулевым?
        BIS #3,R0           ;Если нет, то выставить в буферном
                            ;регистре пару младших битов в единицы
3:      ASL R0              ;Проворот буферного регистра
        ASL R0              ;на две позиции в сторону старшего бита
        SOB R4,2            ;Если еще не все биты, повтор
        BIC (SP),R0         ;Строчка окрашивается в заданный цвет
        MOVB R0,(R3)        ;Младший байт, а затем старший
        SWAB R0             ;выводится из буфера на экран
        MOVB R0,1(R3)       ;(такой способ предусмотрен на
                            ;случай нечетного адреса в R3)
        ADD #100,R3         ;Переход к следующей строчке,
        SOB R2,1            ;если эта не последняя
        TST (SP)+           ;Освободить ячейку стека
        RTS PC
```

Эту подпрограмму при желании можно объединить с предыдущими для получения жирного, наклонного, контурного и т. п. шрифта в режиме 32 символа в строке. Как это сделать, предоставим читателям решить самостоятельно, ограничившись здесь только небольшой подсказкой: нужно не вызывать подпрограммы одну за другой, а скомпоновать в единую подпрограмму «рабочие команды» (ROLB, RORB, MOVB и т. д.) из нескольких.

Широкий разноцветный шрифт

Если обычный широкий шрифт покажется вам не очень интересным, его можно сделать разноцветным. Наиболее простой способ раскраски заключается в выводе верхней половины букв одним цветом, а нижней — другим. В остальном эта подпрограмма, по сути, состоит из двух однотипных модулей, аналогичных предыдущему листингу вывода широкого одноцветного шрифта.

```
BEGIN:  MOV #5,R2
        MOV CLRCHR,-(SP)
        COM (SP)
1:      CLR R0
        MOVB (R1)+,R5
        MOV #10,R4
2:      ROLB R5
        BCC 3
        BIS #3,R0
3:      ASL R0
        ASL R0
        SOB R4,2
        BIC (SP),R0
        MOVB R0,(R3)
        SWAB R0
        MOVB R0,1(R3)
        ADD #100,R3
        SOB R2,1
```

```
        MOV #5,R2
        MOV CL2CHR,(SP)
        COM (SP)
11:     CLR R0
        MOVB (R1)+,R5
        MOV #10,R4
12:     ROLB R5
        BCC 13
        BIS #3,R0
13:     ASL R0
        ASL R0
        SOB R4,12
        BIC (SP),R0
        MOVB R0,(R3)
        SWAB R0
        MOVB R0,1(R3)
        ADD #100,R3
        SOB R2,11
        TST (SP)+
        RTS PC
```

Тени от букв

Еще один интересный шрифтовой эффект — когда каждый символ отбрасывает тень, как будто буквы сделаны из проволоки и приподняты над поверхностью экрана. При этом мы будем работать с цветным (широким) шрифтом красного цвета (можно сделать и другой, но это несколько усложнит программу), а тени будут черными. Соответственно, чтобы они были заметны, необходимо предварительно закрасить фон всего экрана синим или зеленым цветом, например модифицировав начало листинга демонстрационной программы следующим образом:

```
START:  EMT 14
        MOV #52525,@#212 ; включить синий фон экрана
        MOV #14,R0       ; код очистки экрана в R0
        EMT 16           ; закраска экрана синим цветом
        MOV #50004,ADR
        . . . . . .
```

Эффект тени несложно получить, если после высвечивания очередной строчки символа красным цветом погасить ею же биты фона со смещением на один байт вправо и на три-четыре строки вниз (команда BIC).

```
BEGIN:  MOV #12,R2      ;Высота шрифта
      ;Распаковка узкого шрифта в широкий:
1:      CLR R0          ;Очистка буферного регистра
        MOVB (R1)+,R5   ;Строчка стандартного символа
                        ;запоминается в регистре
        MOV #10,R4      ;Количество битов в байте
2:      ROLB R5         ;Очередной бит регистра -> в бит C
        BCC 3           ;Бит нулевой?
        BIS #3,R0       ;Нет - установить пару младших битов
3:      ASL R0          ;Провернуть буферный регистр
        ASL R0          ;на две позиции влево
        SOB R4,2        ;Если еще не все биты, повторить
        BISB R0,(R3)    ;Высветить мл. байт строчки символа
        BICB R0,401(R3) ;и тень от него (смещена на 1 байт
                        ;вправо и на 4 строки вниз)
        SWAB R0         ;
        BISB R0,1(R3)   ;То же для старшего байта
        BICB R0,402(R3) ;и его тени
        ADD #100,R3     ;Перейти к следующей строчке,
        SOB R2,1        ;если эта не последняя
        RTS PC
```

* * *

Возможно, читателям удастся отыскать и другие интересные алгоритмы нестандартного вывода текста на базе прошитого в ПЗУ шрифта. Например, можно реализовать таким способом векторный шрифт, воспользовавшись алгоритмом векторизации растрового изображения, описанным в журнале «Персональный компьютер БК-0010 — БК-0011М» №1 за 1994 г. Своими программами (или просто идеями относительно возможного вида шрифтов) вы можете поделиться с другими пользователями на страницах журнала.

В предыдущей статье рассказывалось о том, как получить на базе стандартного шрифта БК текст необычного вида с помощью программ на ассемблере. Но, к сожалению, многим начинающим пользователям этот язык пока еще недоступен. В таком случае можно обратиться к БЕЙСИКу — его возможности вполне достаточны для получения многих интересных шрифтовых эффектов.

От редакции

Нестандартные шрифты средствами БЕЙСИКа

Для получения шрифта нестандартного начертания (жирный, курсивный, разноцветный, увеличенной высоты или ширины и т. д.) возможны три метода. Наиболее широкие перспективы открывает использование собственного шрифта, загружаемого в ОЗУ и фактически представляющего собой набор спрайтов с изображениями каждого символа (о реализации такого шрифта на БЕЙСИКе рассказано в журнале «Персональный компьютер БК-0010 — БК-0011М», №1 за 1994 г.). Недостаток загружаемого шрифта — слишком большой расход оперативной памяти. Второй способ — считывание прошитого в ПЗУ БК стандартного набора изображений символов, преобразование байтов и их вывод на экран. Но проще всего получить текст нестандартного вида, выводя оператором PRINT стандартную строку нужного текста в неиспользуемую для других целей («буферную») область экрана и перерисовав символы с помощью графических функций POINT и PSET. Именно этот способ мы и будем рассматривать далее.

Жирный шрифт

Для получения жирного шрифта нужно утолстить вертикальные линии символов, поставив дополнительные точки слева или справа.

```
10 CLS
100?AT(5,0);"Пример ТЕКСТА"
110 FOR Y=0 TO 10
120 FOR X=1 TO 255
130 IF POINT(X,Y)<>4 AND POINT(X-1,Y)=4 THEN PSET(X-1,Y),POINT(X,Y)
140 NEXT X,Y
150 END
```

Если утолщение линий производить не по всей высоте символов, а только, например, в нижней половине строки, то получим интересный эффект «приливов» слева у каждой буквы. Для этого в вышеприведенной программе нужно заменить оператор цикла в строке 110 на **FOR Y=4 TO 10.**

Оттененный шрифт

Если высвечивать дополнительные точки не тем же самым цветом, что и основной текст, а другим, то получится любопытный эффект оттененного шрифта.

```
10 CLS
100?AT(5,0);"Пример ТЕКСТА"
110 FOR Y=0 TO 10
120 FOR X=1 TO 255
130 IF POINT(X,Y)=1 AND POINT(X-1,Y+1)=4 THEN PSET(X-1,Y+1),2
140 NEXT X,Y
150 END
```

Если же немного изменить условие в операторе IF (строка 130), то получим эффект «световых лучей» от каждой точки символов.

```
10 CLS
100 ?AT(5,0);"Пример ТЕКСТА"
110 FOR Y=0 TO 10
120 FOR X=1 TO 255
130 IF POINT(X,Y)<>4 AND POINT(X-1,Y+1)=4 THEN PSET(X-1,Y+1),2
140 NEXT X,Y
150 END
```

Курсив

Для получения наклонного шрифта (курсива) достаточно сдвинуть две верхние графические строки изображения каждого символа на четыре точки вправо, следующие две — на три точки вправо, еще две — на две точки и очередные две — на одну. Последние две строки остаются без изменений.

```
10 CLS
100 ?AT(5,0);"Пример ТЕКСТА"
110 FOR Y=6 TO 0 ST -2
120 FOR Y1=Y TO 0 ST -1
130 FOR X=256 TO 1 ST -1
140 PSET (X,Y1),POINT(X-1,Y1)
150 NEXT X,Y1,Y
160 END
```

Чтобы получить «обратный курсив» (наклон влево), нужно аналогичным образом сдвигать строки не вправо, а влево, для чего требуется изменить порядок «прокручивания» внутреннего цикла на противоположный.

```
10 CLS
100 ?AT(5,0);"Пример ТЕКСТА"
110 FOR Y=6 TO 0 ST -2
120 FOR Y1=Y TO 0 ST -1
130 FOR X=0 TO 254
140 PSET (X,Y1),POINT(X+1,Y1)
150 NEXT X,Y1,Y
160 END
```

Контурный шрифт

Для получения контурного шрифта необходимо запрограммировать два цикла и использовать промежуточный цвет. В первом цикле зеленые точки контура ставятся рядом с красными (основными). А во втором внутренние красные точки (собственно текст) стираются, а оставшийся контур перерисовывается красным. (Можно и не делать этого, оставив контур зеленым, — для этого надо убрать из листинга строку 180.)

```
10 CLS
100 ?AT(5,0);"Пример ТЕКСТА"
110 FOR Y=0 TO 10
120 FOR X=0 TO 255
130 IF POINT(X,Y)=4 AND (POINT(X-1,Y)=1 OR POINT(X+1,Y)=1 OR
   POINT(X,Y-1)=1 OR POINT(X,Y+1)=1) THEN PSET(X,Y),2
140 NEXT X,Y
150 FOR Y=0 TO 10
160 FOR X=0 TO 255
170 IF POINT(X,Y)=1 THEN PSET(X,Y),4
```

```
180 IF POINT(X,Y)=2 THEN PSET(X,Y),1
190 NEXT X,Y
200 END
```

Высокий и широкий шрифт

Для получения символов двойной, тройной и т. д. высоты проще всего использовать привычный метод перебора графических строк изображения в цикле и операторы POINT и PSET. Следующая программа преобразует выведенную стандартным шрифтом текстовую строку в шрифт двойной высоты.

```
10 CLS
100 ?AT(5,0);"Пример ТЕКСТА"
110 FOR Y=9 TO 0 ST -1
120 FOR X=0 TO 255
130 A=POINT(X,Y)
140 PSET (X,2*Y),A
150 PSET (X,2*Y-1),A
160 NEXT X,Y
170 END
```

Аналогичным способом можно получить шрифт утроенной высоты.

```
10 CLS
100 ?AT(5,0);"Пример ТЕКСТА"
110 FOR Y=9 TO 0 ST -1
120 FOR X=0 TO 255
130 A=POINT(X,Y)
140 PSET (X,3*Y),A
150 PSET (X,3*Y-1),A
160 PSET (X,3*Y-2),A
170 NEXT X,Y
180 END
```

Шрифт удвоенной ширины делается аналогично, но теперь умножению на 2 в операторах PSET подвергается координата X.

```
10 CLS
100 ?AT(1,0);"Пример ТЕКСТА"
110 FOR Y=0 TO 9
120 FOR X=120 TO 0 ST -1
130 A=POINT(X,Y)
140 PSET (2*X,Y),A
150 PSET (2*X-1,Y),A
160 NEXT X,Y
170 END
```

Как написать программу для получения шрифта тройной ширины или с увеличением высоты и ширины одновременно, думаю, читатели догадаются сами.

Зеркальное отображение текста

Следующая программа позволяет увидеть, как выглядело бы изображение на дисплее, если взглянуть на него с внутренней стороны экрана.

```
10 CLS
100 ? AT(5,0);"*** Пример ТЕКСТА ***"
110 FOR Y=0 TO 9
120 FOR X=0 TO 127
130 A=POINT(X,Y)
140 PSET (X,Y),POINT(255-X,Y)
150 PSET (255-X,Y),A
160 NEXT X,Y
170 END
```

Аналогичным образом можно перевернуть текст вверх ногами. Для одной только первой строки экрана программа будет выглядеть так

```
10 CLS
100 ? AT(5,0);"*** Пример ТЕКСТА ***"
110 FOR Y=0 TO 4
120 FOR X=0 TO 255
130 A=POINT(X,Y)
140 PSET (X,Y),POINT(X,9-Y)
150 PSET (X,9-Y),A
160 NEXT X,Y
170 END
```

Окна на БЕЙСИКе

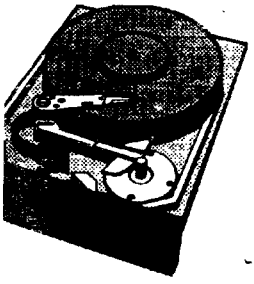
Оконный интерфейс давно уже завоевал прочные позиции на всех компьютерах — вспомним хотя бы повсеместное стремление на IBM-совместимых к переходу от MS-DOS к Norton Commander и далее к Windows. В последнее время и на БК «оконно-менюшный» интерфейс все чаще появляется в программах на ассемблере, БЕЙСИКе и даже не столь популярном сегодня ФОКАЛе. (Прочитать о реализации окон и меню на перечисленных языках программирования можно в третьем выпуске журнала «Персональный компьютер БК-0010 — БК-0011М» за 1994 г.) На БЕЙСИКе же вывести текст в окне проще простого, нужно лишь воспользоваться возможностью прямого перезадания цвета фона символов с помощью оператора POKE &O212,<цвет> и функцией AT в операторе PRINT. Вот, например, как можно создать окно без тени на черном фоне:

```
10 CLS
20 POKE &O212,&O125252 ' зеленый фон окна
30 ? CHR¤(&O221) ' красный цвет текста
40 ? AT(10,10);"┌─────────┐"
50 ? AT(10,11);"│ ПРИМЕР │"
60 ? AT(10,12);"└─────────┘"
70 POKE &O212,0% ' восстановить черный цвет фона
80 END
```

Думаю, читатели сами догадаются, как вывести окно с тенью, печатая пробелы с черным фоном в требуемых знакоместах ниже и правее рамки окна. (Разумеется, фон всего экрана в этом случае должен быть другим, например синим, чтобы черная тень окна была заметна.)

* * *

Итак, БЕЙСИК, несмотря на его относительную медлительность, достаточно богат графическими возможностями. И последние вовсе не исчерпываются перечисленными выше примерами нестандартного начертания символов. Думается, читателям удастся отыскать еще немало интересных шрифтовых эффектов и рассказать о своих находках на страницах нашего журнала.



Заманчивая идея подключить к своей БК жесткий диск сегодня обуревает многих. Большинство пользователей предпочитают приобретать готовые комплекты (винчестер и контроллер для его подключения к БК); возможно, найдутся и желающие подключить винчестер своими руками. Надеемся, что приведенные в этой статье рекомендации А. М. Надежина, знакомого нашим читателям по предыдущим публикациям, и руководителя фирмы «АльтПро» В. Е. Новака помогут им.

А. М. Надежин,
Москва

Подключение винчестера к БК: вопросы и ответы

После публикации во втором выпуске журнала за 1995 г. статей В. Новака, М. Королева и Д. Бутырского о подключении к компьютеру БК-0010(.01) и БК-0011(М) жесткого диска (винчестера) в редакцию и фирму «Альтек» обращается немало пользователей, желающих получить консультации по основным техническим аспектам, связанным с этой аппаратной доработкой. Приведенные ниже сведения, представленные в виде вопросов и ответов на них, конечно, не претендуют на абсолютную полноту освещения данной проблемы, но могут дать о ней хотя бы начальное представление.

Какой винчестер лучше всего подходит для подключения к БК?

К БК подключается практически любой IDE-винчестер, однако существуют отдельные их типы, не работающие с этим компьютером. Проверено на практике подключение жестких дисков фирм Seagate, Conner, IBM, Microsciense, Priritek, модели же фирмы Western Digital серии Caviar для БК непригодны. Относительно других типов винчестеров информации нет, и будут ли они работать на БК, можно узнать, только попытавшись их подключить.

Винчестеры MFM (использовавшиеся для ДВК, Электроники-85 и старых моделей IBM) к БК обычно не подключают (хотя подобная разработка существует — см. статью В. Макарова из Санкт-Петербурга в №1 журнала «Персональный компьютер БК-0010 — БК-0011М» за 1993 г.), так как контроллеры для них занимают почти столько же места, как сам БК, и потребляют (вместе с винчестером) более 5 А.

Винчестеры какого объема можно использовать на БК?

Существующие модели контроллеров поддерживают жесткие диски любого объема (вплоть до нескольких гигабайт), однако рекомендуется использовать винчестеры на 40 Мб, так как при минимальной цене их объем для БК наиболее оптимален. Но, разумеется, это не означает, что, если у вас уже имеется винчестер большего объема, его придется заменять на 40-мегабайтный.

Как узнать: конкретный винчестер типа IDE или нет?

Как правило, IDE-винчестеры имеют размер 3 дюйма и подключаются с помощью точно такого же, как у дисковода, разъема питания и информационного 40-контактного разъема, подобного используемому в стандартном контроллере НГМД для БК.

Существуют также двухдюймовые IDE-винчестеры для NoteBook. У них имеется всего один 44-контактный разъем с шагом 2 мм, поэтому для их подключения требуется переходник с двумя стандартными разъемами (часто он продается вместе с винчестером).

Чем двухдюймовый винчестер лучше трехдюймового?

И тот, и другой тип жесткого диска имеет свои преимущества и недостатки. Двухдюймовый винчестер меньше, а значит, его можно установить внутрь корпуса контроллера двойной высоты (от блока КРМП) или даже внутрь БК. Кроме того, в отличие от 3-дюймовых моделей, он

не требует подвода напряжения питания +12 В (достаточно +5 В), поэтому его можно подключать к блоку питания компьютера. Но зато трехдюймовый винчестер существенно дешевле.

Какое питание требуется винчестеру?

Трехдюймовый жесткий диск требует, как и дисковод, два напряжения питания: +5 В и +12 В, в среднем потребляя около 0.5 А по каждому напряжению. Однако в момент раскрутки маховика двигателя потребление по +12 В может достигать 1.5 А. Часто можно использовать для питания винчестера блок питания дисковода.

Двухдюймовый винчестер, как уже отмечалось, имеет только одно напряжение питания +5 В, поэтому он может подключаться непосредственно к шинам питания БК. Но при использовании для этой цели контактов «+ 5 В» и «Общий» разъема МПИ следует продублировать ведущие к ним печатные дорожки монтажными проводами, ведущими к МПИ непосредственно от разъема питания БК.

Нужен ли для подключения IDE-винчестера к БК специальный контроллер? Какие они бывают?

Для подключения жесткого диска к БК необходим специальный контроллер, подключаемый к разъему МПИ параллельно контроллеру дисковода. Существует два разных типа подобных устройств: «самарский» и «АльтПро». Первый из них (устанавливаемый в фирме «Альтек») монтируется только внутрь БК-0011М (но не на десятой модели!) вместо платы ПЗУ БЕЙСИКа. Этот контроллер не имеет своего ОЗУ и ПЗУ, поэтому необходима замена ПЗУ в контроллере дисковода, а чтобы узнать параметры разделов винчестера, приходится каждый раз считывать их с него побайтно.

Второй контроллер (устанавливаемый фирмой «АльтПро», а также автором этой статьи) представляет собой плату, которая может монтироваться внутрь контроллера дисковода «вторым этажом», внутрь БК или в отдельном стандартном корпусе типа МСТД (в последнем случае контроллеры дисковода и винчестера подключаются к разъему МПИ через разветвитель). Контроллер «АльтПро» позволяет работать с винчестером на любой модели БК и имеет собственное ПЗУ объемом 4 Кб с модифицированной 326-й прошивкой, управляющей работой контроллеров дисковода и винчестера, а также ОЗУ емкостью 4 Кб, размещаемое с адреса 170000, для хранения таблицы разделов винчестера, резидентных драйверов и программ пользователя. Все это обеспечивает значительно большее удобство пользования по сравнению с «самарской» моделью.

Как быть тем, кто не умеет работать с паяльником? Можно ли приобрести комплект для установки винчестера «под ключ»?

Обычно практикуется обмен вашего контроллера дисковода (от БК-0011М либо КНГМД фирмы «Альтек» или «АльтПро» от БК-0010(.01)) на сдвоенный контроллер дисковода-винчестера «АльтПро». При этом его стоимость будет той же, как при покупке отдельной платы контроллера винчестера.

А если пользователь захочет сам подключить плату контроллера винчестера, нужно ли дорабатывать контроллер дисковода?

Да, необходимо отключить внутреннюю микросхему ПЗУ контроллера дисковода, для чего достаточно вынуть кристалл из монтажной панельки или перерезать вывод 21 ПЗУ 1801РЕ2-326.

Какие дисковые операционные системы могут работать с винчестером?

С винчестером работают абсолютно все известные автору операционные системы (ОСБК-11 V4.0, ANDOS V3.1, MKDOS V3.15 и др.), причем с контроллером «АльтПро» становится возможным хранение на винчестере нескольких систем одновременно и загрузка любой из них по умолчанию, а остальных — при явном указании номера логического раздела в команде загрузки.

Что представляет собой винчестер с точки зрения пользователя, учитывая значительно больший по сравнению с дискетой объем этого устройства?

Винчестер по желанию пользователя разбивается на множество логических разделов — «дискет», доступных по стандартному обращению CALL @#160004 (вызов подпрограммы чтения/записи секторов, прошитой в ПЗУ контроллера). Объемы этих разделов можно произвольно менять.

Какие программы поставляются в комплекте с винчестером и контроллером «АльтПро»?

В комплект поставки входят следующие сервисные программные средства:

- SERVIS — программа обслуживания винчестера, позволяющая выполнять его разбиение на разделы произвольного объема, тестировать винчестер и контроллер, форматировать винчестер или его отдельные дорожки;
- DESS_HD — модифицированная версия DESS V2.7, позволяющая обращаться к любым разделам винчестера по присвоенным им буквам (C:, D:, E:, F: и т. д.) или по номерам, а также ко всему винчестеру как к единому устройству;
- XEROX2 — новая версия известной программы копирования дисков, позволяющая работать с любыми разделами винчестера.

Кроме того, в комплект входит документация для пользователя и системного программиста.

В. Е. Новак,
Москва

Контроллер IDE-винчестера для БК

Плата контроллера выполняет функции формирователя интерфейсных сигналов, дешифратора адресов для регистров НЖМД, имеет собственное ПЗУ (по адресам 160000—167777) со слегка модифицированной «326-й» прошивкой и базовым драйвером НЖМД, а также статическое ОЗУ (адреса 170000—176777). Драйвер расширенной арифметики из «326-й» прошивки в целях экономии места заменен на более короткий (старая длина — 2400 байт, новая — 776), но без поддержки вычислений с плавающей точкой. (Имеется и возможность подключения пользовательского драйвера арифметики.)

Примененная при разработке идеология позволяет подключать IDE-накопители емкостью до 2 Гб (63. сектора, 16. головок, 4095. дорожек). Накопитель может быть разбит не более чем на 124. логических дисков произвольного объема (от 5 Кб до 32 Мб каждый).

Регистры НЖМД

Адрес регистра	Назначение	
	Чтение	Запись
177740	статус	команда
177742	выбор НЖМД и головки	
177744	ст. байт номера дорожки	
177746	мл. байт номера дорожки	
177750	номер сектора	
177752	счетчик секторов	
177754	код ошибки	не используется
177756	данные чтения	данные записи
177741	адрес накопителя	не используется
177743	2-й статус	сброс

Примечания.

1. Ячейка 177741 — это не старший байт ячейки 177740, а самостоятельный регистр, расположенный по нечетному адресу (то же касается регистра 177743).

2. Ввиду того что у НЖМД и БК по-разному кодируются «0» и «1» на шине (единице на шине БК соответствует низкий уровень напряжения, а на шине НЖМД — высокий), под фразой «бит установлен» следует понимать его отсутствие, т. е. все данные представлены в инверсном коде.

3. Замечание для программистов: байтовые команды записи в ОЗУ платы контроллера НЕ РАБОТАЮТ.

Запросы к драйверу НЖМД

Блок параметров

Структура блока параметров драйвера НЖМД полностью идентична структуре блока параметров драйвера дисковода. Теми же остаются и точки входа для чтения/записи.

Под номером привода теперь следует понимать: 0 и 1 — дисководы (A: и B:), остальное — логические диски НЖМД (C:, D: и т. д.). Всего НЖМД может быть разбит не более чем на 124 логических диска произвольного объема.

Записи о логических дисках находятся в таблице разделов, к которой предусмотрен программный доступ (см. ниже). Нумерация логических дисков производится начиная с нуля (устрой-

ству C: соответствует логический диск с номером 0 и т. д.). Не рекомендуется разбивать накопитель на слишком большое количество логических дисков, потому что в этом случае затрудняется доступ к ним из операционных систем (не хватает букв), немного замедляется доступ при поблочном чтении через подпрограмму с адресом 160004 (так как каждый раз пересчитывается CRC таблицы разделов и области конфигурации), да и разобраться в таком обилии информации довольно тяжело.

Примечание. Здесь и далее не следует путать логические диски операционных систем (RT11, MKDOS и др.) с логическими дисками НЖМД.

Точки входа в драйвер

- 160000 — автоматическая загрузка ДОС. Параметры отсутствуют. Алгоритм загрузки описан ниже.
- 160002 — загрузка ДОС с выбранного привода (R0 — номер привода). Алгоритм загрузки описан ниже.
- 160004 — чтение/запись по номеру блока. Входные параметры: R0 — номер начального блока; R1 — длина пересылаемого массива данных (в словах), положительное число — чтение, отрицательное — запись (в последнем случае длина массива равна абсолютному значению числа); R2 — адрес массива данных; R3 — адрес начала блока параметров; ячейка 34(R3) — номер привода (0—124.).

Если номер начального блока находится вне пределов логического диска, то драйвером генерируется сообщение об ошибке 5 и обмен данными с системой не производится. Если при чтении/записи пересекается граница логического диска, также выдается ошибка 5, но передается максимально возможное количество данных. Если логический диск не определен, драйвер сообщает об ошибке 3. Если НЖМД не подключен или не выдает сигнал готовности в течение определенного промежутка времени, выдается ошибка 6.

При каждом новом обращении к НЖМД производится подсчет CRC таблицы разделов и области конфигурации (простое сложение без учета переноса) и сравнение ее с эталоном, находящимся в области служебной информации. В случае несовпадения таблица разделов и конфигурация повторно считываются в память, заново вычисляются контрольная сумма и количество доступных логических дисков, выполняется программный сброс накопителя, переустановка его параметров (как при загрузке) и выдается ошибка 13_8. Обмен данными не производится. Последующие обращения к НЖМД происходят нормально.

Ошибка 13_8 является сигналом порчи таблицы разделов или области конфигурации и позволяет выявить программу, которая умышленно или неумышленно это сделала.

Примечание. Под областью конфигурации понимаются поля HD$TRK, HD$HEA, HD$SEC и HD$LOG зоны служебной информации.

- 160006 — чтение/запись по номеру сектора, головки и дорожки. Входные параметры: R3 — адрес начала блока параметров; 26(R3) — адрес массива данных; 30(R3) — длина пересылаемого массива данных (в словах), положительное число — чтение, отрицательное — запись; 32(R3): младшие 4 бита — номер головки (0—15.), старшие 4 бита — биты 0—3 номера дорожки; 33(R3) — биты 4—11 номера дорожки; 34(R3) — номер привода (0—124.); 35(R3) — номер начального сектора.

Номер привода при чтении никакой существенной роли не играет, необходимо только, чтобы он был больше или равен 2. Запись же возможна лишь при номере привода, равном 377_8, при попытке записи с номером привода от 2 до 376_8 выдается ошибка 1. Чтение и запись возможны в пределах всего накопителя, разбиение на логические диски не анализируется. При выходе за пределы конфигурации НЖМД выдается ошибка 5.

Примечание. После окончания работы подпрограмм драйвера, связанных с передачей информации (160000—160006 и 160012), в R4 помещается адрес регистра ошибок с целью нейтрализации действий программ, которые заносят 0 (и не только) по адресу, содержащемуся в R4.

- 160010 — инициализация блока параметров. R3 — адрес блока параметров.
- 160012 — форматирование. Входные данные: R3 — адрес блока параметров; 32(R3): младшие 4 бита — номер головки (0—15.), старшие 4 бита — биты 0—3 номера дорожки; 33(R3) — биты 4—11 номера дорожки; 34(R3) — номер привода (0—124.).

При обращении к дисководам производится форматирование текущей дорожки с текущей стороны. Для НЖМД выполняется команда «верификация дорожки».

- 160016 — точка входа в драйвер расширенной арифметики. Эмулируются команды EIS: MUL, DIV, ASH и ASHC.

Чтобы подключить эмулятор, занесите адрес его начала в вектор 10_8. При прерывании по резервной команде происходит обращение к эмулятору. Если число, вызвавшее прерывание, не является кодом одной из вышеперечисленных команд, происходит выход по вектору 4 с установленным битом C (стандарт RT11), иначе выполняется программная эмуляция команды. В системе имеется возможность подключения пользовательского драйвера расширенной арифметики.

Таблица разделов и служебная информация

Статическое ОЗУ расположено по адресам 170000—176777. При работе базового драйвера используются следующие области памяти (в скобках указаны адреса ячеек в ОЗУ):

- HD$TRK (176776) — количество дорожек НЖМД;
- HD$HEA (176774) — количество головок НЖМД;
- HD$SEC (176772) — количество секторов на дорожке;
- HD$LOG (176770) — общее количество логических дисков (1—124.), в старшем байте — номер привода, с которого производится загрузка по умолчанию;
- HD$SPD (176000) — признак повышенной скорости обмена с накопителем (рекомендуется для современных моделей, повышает скорость чтения/записи в два и более раз за счет отсутствия ожидания готовности при чтении сектора, но на старых накопителях может привести к сбоям и потере информации, поэтому сначала проконсультируйтесь со специалистами). Признак установлен, если значение этой ячейки равно 125252;
- HD$USR (175776) — указатель на начало драйвера пользователя;
- HD$EIS (175774) — указатель на начало пользовательского драйвера расширенной арифметики (первой командой драйвера должна быть NOP, управление передается по JMP @175774, при этом регистры уже сохранены в стеке, в R0 — код команды, вызвавшей прерывание).

Таблица разделов НЖМД начинается с адреса 176766 и имеет стековую структуру (растет вниз). Каждая запись о логическом диске занимает два слова в следующем формате. Слово с большим адресом содержит номер дорожки и головки начала логического диска (младшие 4 бита — номер головки, остальные 12 битов — номер дорожки). Если это слово отрицательно (инверсия по COM), то считается установленной защита записи и при попытке изменения данных на диске выдается ошибка 1. Слово с меньшим адресом содержит длину логического диска в блоках (максимум 65535 блоков = 32 Мб). В слове, следующем за последней записью таблицы разделов, находится CRC таблицы разделов и области конфигурации.

В базовом драйвере НЖМД предусмотрен доступ к таблице разделов. (Для общей совместимости с последующими версиями рекомендуется пользоваться им, а не обращаться напрямую к области дополнительного ОЗУ.) Если при вызове 160004 (**но** не 160006!) указать в R1 количество слов для пересылки, равное нулю, драйвер возвратит следующую информацию: R0 — размер данного логического диска в блоках (номер привода берется из 34(R3)); R1 — номер дорожки/головки начала логического диска в формате, описанном выше; в область памяти начиная с адреса из R2 (если R2 не равно нулю) передается следующее:

Смещение	Содержимое
0	количество дорожек НЖМД
2	количество головок НЖМД
4	количество секторов на дорожке
6	количество логических дисков

Кроме того, вычисляются координаты (дорожка, головка, сектор) блока, который был указан в R0 на входе. Если номер блока находится за пределами логического диска, вычисляются координаты последнего доступного блока данного диска. Вычисленные параметры помещаются в соответствующее место блока параметров (см. описание точки входа 160006). После выполнения операции R2 портится.

Внимание! При подключенном драйвере пользователя эта функция не работает, если она не предусмотрена в последнем.

Под служебные нужды отводится нулевая сторона (головка 0) нулевой дорожки. Первые 7 блоков этой области занимает основной загрузчик (Master Boot Record, или MBR), который

при каждой загрузке с винчестера считывается в ОЗУ с адреса 10000 и запускается, если его первая команда — NOP.

Каждая перезагрузка с НЖМД вызывает обновление в памяти копии таблицы разделов, области конфигурации и проверку их CRC.

В седьмом блоке (восьмом секторе) хранится копия области ОЗУ 176000—177000.

Алгоритм загрузки

Алгоритм загрузки ДОС с винчестера или дисковода следующий:

- если НЖМД подключен, то при любой загрузке (в том числе с дисководов) с адреса 176000 считывается 8-й сектор;
- проверяется CRC. Если нужно, то конфигурация считывается из паспорта НЖМД, а таблица разделов инициализируется;
- выполняется программный сброс накопителя и команда инициализации параметров;
- если номер привода не был указан явно, берется номер по умолчанию (из HD$LOG+1). Если в итоге номер привода меньше 2, производится переход к стандартной загрузке;
- с адреса 10000 считывается MBR (первичная загрузка). Если его первая команда — NOP, то управление передается на нее (CALL @#10000). MBR, произведя необходимые действия (например, переконфигурацию области со служебной информацией, установку драйверов пользователя и т. д.), возвращает управление драйверу по RTS PC;
- в основную память с адреса 1000 считывается нулевой блок устройства с номером из 34(R3), и загрузка продолжается по стандартному алгоритму драйвера дисковода (вторичная загрузка).

В случае ошибки загрузка продолжается с дисководов A: и B: (если загрузка автоматическая) или, в противном случае, прекращается.

Подключаемый драйвер пользователя

Подключаемый драйвер пользователя может располагаться в любом месте ОЗУ. Чтобы его установить, необходимо по адресу 175776 записать адрес его начала. Первой командой драйвера должна быть NOP; далее идут:

- точка входа по 160004;
- точка входа по 160006;
- точка входа по 160012.

Предполагается, что драйвер пользуется тем же блоком параметров, что и базовый.

Управление драйверу пользователя передается сразу же после вызова базового драйвера, никаких действий по обработке параметров не производится, номер привода не анализируется, контрольная сумма на считается.

Драйвер пользователя возвращает управление базовому, признак ошибки должен находиться в блоке параметров (57(R3)). Коды ошибок — стандартные, бит C роли не играет. Управление может возвращаться несколькими способами:

- RTS PC — обычный возврат управления, после которого происходит выход из базового драйвера с установкой бита C и номера ошибки, если 57(R3) содержит ненулевое значение;
- ADD #2,@SP и RTS PC — управление напрямую передается драйверу ДИСКОВОДА;
- ADD #6,@SP и RTS PC — управление передается базовому драйверу, как будто драйвер пользователя не подключен.

Если RTS PC заменить на JSR PC,@(SP)+ (естественно, в первом случае этого делать не следует), то управление от драйверов дисковода или НЖМД будет по окончании работы возвращено в драйвер пользователя.

Существующая система подключаемых драйверов позволяет создавать уникальные драйверы на любой вкус. Например, можно «отключить» дисководы, «переместив» все логические диски вниз (чтобы не «пропадали» буквы, если вы работаете без дисководов), можно и обеспечить вызов электронного диска.

Распределение ДОЗУ по адресам 170000—175770

В процессе работы прикладные программы могут занимать участки ДОЗУ и освобождать их в произвольной последовательности. Для бесконфликтной работы программ необходимо придерживаться следующих требований:

- резиденты, размещаемые в ДОЗУ, должны быть перемещаемыми;

- программа, которой необходим участок ДОЗУ, должна захватывать область начиная с первой же свободной ячейки. Первое слово свободной области равно 0;
- в первое слово занимаемого участка вписывается его длина в байтах, в ячейку после этого участка заносится 0. Со второго слова участка начинается список адресов ячеек-указателей, через которые производится адресация к данному участку памяти, список оканчивается нулем. Со следующего адреса располагается тело резидентной программы или массив данных.

Если какая-либо программа будет освобождать участок ДОЗУ, она должна будет переместить в сторону меньших адресов все расположенные после этого участка резиденты, вычтя при этом величину смещения из содержимого ячеек-указателей, адреса которых находятся в списке перед телом каждого резидента. Пример:

```
170000: .WORD   TOP1-170000
        .WORD   ANDOS+654,0       ;взято для примера
         ...

TOP1:   .WORD   TOP2-TOP1         ;первый драйвер пользователя
        .WORD   JMP2TO1,0
        ...
START1: ...
        RTS     PC

TOP2:   .WORD   TOP3-TOP2         ;второй драйвер пользователя
        .WORD   HD¤USR            ;=175776
        .WORD   TOP1+2,0
        ...
START2: ...
        JMP     @(PC)+
JMP2TO1:.WORD   START1

TOP3:   .WORD   0

HD¤USR: .WORD   START2
```

Здесь после, например, переименования логических дисков вторым драйвером (часто требуется для «непонятливых» программ переименовывать диски в A: или B:), управление будет передано первому драйверу, о предназначении которого автору второго драйвера, скорее всего, неизвестно. Не составляет труда удалить второй драйвер и скорректировать HD$USR.

Приложение. Назначение выводов IDE-интерфейса

Все сигналы IDE-интерфейса являются TTL-совместимыми. Колонка I/O указывает направление сигнала: «I» означает вход жесткого диска, «O» — выход. Сигналы, начинающиеся с «/», при низком уровне имеют активное состояние.

№ контакта	Сигнал	I/O	Описание
2, 19, 22, 24, 26, 30, 40	GND		Общий провод
1	RES	I	Сигнал сброса от процессора
18, 3	D15, D7	I/O	Двунаправленная шина данных между процессором и жестким диском для передачи данных, информации о состоянии и управляющей информации. Линии переключаются в высокоомное состояние, если жесткий диск не выбран
16, 5	D14, D6		
14, 7	D13, D5		
12, 9	D12, D4		
10, 11	D11, D3		
8, 13	D10, D2		
6, 15	D9, D1		
4, 17	D8, D0		
20			Удаленный контакт. Используется в качестве ключа во избежание путаницы при переворачивании кабеля

№ контакта	Сигнал	I/O	Описание
21			Не используется
23	/IOWR	I	Сигнал записи для адресов портов ввода-вывода
25	/IORD	I	Сигнал чтения для адресов портов ввода-вывода
27	/IOCHRDY	O	Процессор при доступе к жесткому диску должен ожидать до тех пор, пока данный сигнал станет неактивен (у некоторых изготовителей не используется)
28	/ALE	I	Сигнал «Address Latch Enable» (существующими накопителями не используется и добавлен для совместимости с будущими моделями)
29			Не используется
31	/IRQ14	O	Сигнал «Interrupt Request» (запрос прерывания процессора)
32	/IO16	O	Указатель 16-битной передачи данных
34	/PDIAG	I/O	Сигнал «Passed Diagnostics»: Slave-диск сообщает Master-диску, что внутренняя диагностика отработана (в некоторых накопителях отсутствует)
33	A1	I	Биты адреса для выбора регистров
35	A0	I	
36	A2	I	
37	/CS0	I	Сигнал «Card Select» (выбор регистров накопителя)
38	/CS1	I	
39	/ACT	I/O	Может управлять светодиодом, чтобы указывать активность накопителя

Некоторые рекомендации по подключению винчестера к БК

При подключении винчестера обязательными являются некоторые условия по защите от наводок: шлейф «контроллер-винчестер» должен быть не длиннее 40 см; так как многие блоки питания дают электромагнитное излучение, их желательно располагать подальше не только от винчестера (а кстати, и от дисководов тоже), но и друг от друга, если их несколько (например, наблюдалось отрицательное влияние МС9011 на МС9016). Для питания БК и винчестера достаточно одного блока МС9016. При использовании 2.5-дюймового винчестера, установленного в корпус контроллера, необходимо усилить шины питания внутри БК (распаять провода с разъема питания на контакты разъема МПИ «общий» — А2, А3, В2, В3 и «+5 В» — А12, В12).

В случае использования 2.5-дюймового винчестера с переходником необходимо избегать давления на маленький разъем винчестера. Предпочтительно надевать сперва большой разъем, а затем уже весь переходник. Само собой, необходимо избегать ударов и вибраций, особенно во время работы.

При установке платы контроллера винчестера в КНГМД необходимо отключить ПЗУ в контроллере дисковода.

Идеология работы с винчестером основана на его разбиении на некоторое количество логических дисков, что отражается в таблице разделов. Логические диски — это «внутреннее понятие» зашитого в ПЗУ драйвера, для прикладных программ и ДОС они ничем не отличаются от физических устройств C:, D:, E:, F: и т. д. Все обслуживание жесткого диска осуществляется с помощью программы SERVICE, после добавления/удаления логических дисков нужно записать обновленную таблицу разделов на винчестер.

Форматирование винчестера производится заводом-изготовителем и впоследствии не требуется (за исключением появления дефектных секторов, выявляемых программой SERVICE, что наблюдается крайне редко). В этом случае может потребоваться форматирование отдельных фрагментов с помощью той же программы SERVICE. (Программа ANFORMAT и другие аналогичные дисковые утилиты вместо форматирования выполняет операцию верификации.)

При работе с винчестером могут возникнуть две ошибки, не связанные с неисправностями аппаратуры (случаи обращения к несуществующему логическому диску не в счет):

- ошибка #11 (в программах может отображаться как «ошибка #;»). Причина — какая-то программа испортила копию таблицы разделов в дополнительном ОЗУ по адресам

170000—177000. Эта ошибка устраняется автоматически, и повторное обращение к винчестеру проходит нормально;.

- ошибка #6 — иногда возникает при сетевых или исключительно серьезных программных сбоях (пропадает готовность винчестера). В этом случае НЕОБХОДИМО ПРОЯВИТЬ НЕКОТОРОЕ ТЕРПЕНИЕ: драйвер ожидает появления готовности в течение 5—8 с, после чего ошибка устраняется автоматически, как и в предыдущем случае.

Разработка специальной версии ANDOS для поддержки винчестера пока не завершена, а при работе с существующей версией есть некоторые ограничения: ANDOS работает только с устройствами объемом 1600_{10} блоков; «KT-Init» при первичной инициализации не работает — пользуйтесь специальной версией ANFORMAT (названной для предотвращения возможной путаницы «ANFORM»); для работы с диском E: необходимо отключить виртуальный диск с помощью SVD.M, при этом отключается и «KT-Init». В ANFORM при команде DIR сразу отпускайте кнопку имени устройства — иначе возможны неприятности.

Уже созданы версии программ XEROX (название — XEROX1) и DESS без ограничения по номерам устройств. Старый XEROX плохо реагирует на ошибочные ситуации при работе с винчестером — не следует допускать «попадания» XEROXа на конец устройства (такое бывает при размерах логического диска винчестера менее 1600 блоков).

Что же касается другой популярной ОС — MKDOS 2.30, эта версия для винчестера тоже не окончательная и будет дорабатываться. MKDOS 2.30 может быть установлен на диск только копированием устройства на устройство (XEROX). Не забудьте после этого сделать удобную установку панелей и сохранить ее командой «AP2»+«9»/«SavCNF». Обратите внимание в документации к MKDOS на то, что кнопки «раздвижка» и «сдвижка» переключают соответствующую панель на системное устройство. В программе СЕРВИС.USR из комплекта MKDOS есть одно неудобство: размеры логических дисков винчестера должны быть занесены в ее тело программой DESS. Таблица размеров устройств начинается с адреса, отображаемого DESSом как 1102. Первые 2 слова, которые относятся к дисководам A: и B:, равны 3100 (десятичное 1600), далее нужно вписать размеры дисков C:, D:, E:, F:, G:, H: и т. д. Для копирования физических устройств на логические диски MKDOS и обратно в комплекте MKDOS имеются программы TIRAGE и IMAGE.

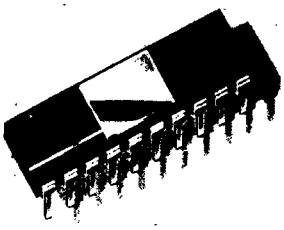
Кроме того, недавно появившаяся версия MKDOS 3.15 имеет «штатную» возможность установки на жесткий диск. В настоящее время эта ДОС наилучшим образом приспособлена для работы с винчестером. (Подробное описание новой версии дано в №2 за 1995 г., с. 66.)

По вопросам приобретения винчестеров, контроллеров для них, блоков питания, кабелей, разъемов и пр., а также необходимого программного обеспечения (в том числе MKDOS 3.15) можно обращаться в редакцию.

Уважаемые читатели!

По вине типографии в четвертом выпуске журнала «Персональный компьютер БК-0010 — БК-0011М» допущен брак: на с. 59 «срезана» часть текста по правому краю (правда, с незначительной потерей читаемости). Кроме того, в некоторых экземплярах возможно наличие пустых (непропечатанных) страниц. К сожалению, брак был обнаружен нами уже после отправки журналов на рассылку подписчикам. Редакция готова заменить экземпляры с непропечаткой страниц, для чего требуется переслать бракованный журнал по адресу: 125315, Москва, а/я 17 — или доставить его в редакцию лично. Вы также можете переслать нам извлеченные из журнала непропечатанные страницы — мы вышлем Вам их полноценные копии. (Не забудьте указать свой подробный почтовый адрес.)

Приносим читателям свои извинения



Отсутствие в заводской конструкции БК (кроме самой старой модели БК-0010) функции перезапуска — одна из наиболее неприятных проблем при работе с этим компьютером, поскольку существует большое количество программ (в основном игровых), которые не имеют «цивилизованного» выхода, а «большая красная кнопка» («СТОП») в них, как правило, надежно заблокирована. Этот недостаток давно обратил на себя внимание пользователей, в литературе уже неоднократно предлагались различные способы его устранения. Авторы ставят своей целью систематизировать имеющиеся сведения по этой проблеме и предложить наиболее оптимальные, на их взгляд, способы ее решения.

С. М. Неробеев, А. В. Сорокин,

Москва

Системный перезапуск («RESET») для БК-0010(.01) и БК-0011(М)

Прежде чем приступить к описанию предлагаемых доработок, поясним, как действует аппаратный механизм перезапуска БК. Микропроцессор К1801ВМ1 имеет два специальных входа: ACLO («Alternating Current LOose» — «исчезновение переменного тока») и DCLO («Direct Current LOose» — «исчезновение постоянного тока»). Сигнал аварии источника питания DCLO вызывает установку процессора в исходное состояние и выдачу на системную магистраль сигнала INIT, сигнал же аварии сетевого питания ACLO инициирует обработку прерывания по сбою питания (вектор 24). Однако в схеме БК нет устройства, которое отслеживало бы снижение сетевого напряжения ниже допустимого уровня, поэтому данные входы задействуются только один раз — при включении питания. Управление же напряжением на них позволило бы вызывать перезапуск процессора без выключения питания в любой требуемый момент времени.

Следует заметить, что блоки питания для компьютеров ДВК имеют в своем составе устройства, отслеживающие изменения сетевого и питающего напряжений и устанавливающие соответствующие уровни сигналов ACLO и DCLO, которые подаются через буферизующие элементы на одноименные выводы процессора 1801ВМ2 (ВМ3). Временные диаграммы этих сигналов показаны на рис. 1. При этом перезапуск вычислительной машины производится с пульта управления блока питания. Таким образом, пользователи, у которых БК работает именно от такого блока (вообще же авторы настоятельно рекомендуют вместо нескольких маленьких блоков питания для монитора, дисководов и вычислительной машины поставить один мощный блок, питающий все эти устройства одновременно), могут в полной мере воспользоваться его возможностями, снимая сигналы ACLO и DCLO непосредственно с блока питания. О том, как это сделать, предполагается рассказать в одном из следующих номеров журнала.

После включения питания, в соответствии с временными диаграммами на рис. 1, высокий уровень (неактивный) устанавливается не ранее чем через 5 мс на входе DCLO и еще через 70 мс (или более) — на входе ACLO. После перехода DCLO в неактивное состояние производится установка внешних устройств в исходное состояние и на выходе INIT появляется высокий уровень сигнала. После перехода сигнала ACLO в неактивное состояние (высокий уровень) разряды 0—7 счетчика команд (R7, или PC) очищаются, а разряды 8—15 загружаются содержимым соответствующих разрядов регистра 177716. В результате в счетчике команд на БК-0010(.01) формируется адрес 100000, а на БК-0011М — адрес 140000. Затем в слово состояния процессора заносится константа 340 и, если запросов на прерывание нет, начинается выполнение команды по адресу, записанному в счетчике команд. Таким образом, если в момент запуска в старшем байте регистра 177716 изменить содержимое некоторых разрядов, можно, соот-

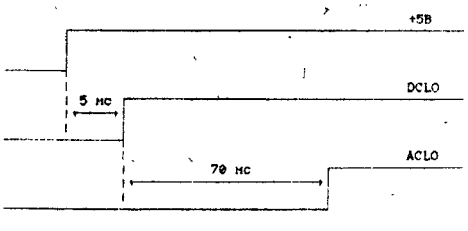


Рис. 1. Временная диаграмма сигналов ACLO и DCLO

ветственно, в достаточно широком диапазоне менять адрес старта процессора. Каким образом это сделать и для чего требуется, будет описано ниже.

Устройство, формирующее сигналы DCLO и ACLO, имеется в составе БК и собрано на микросхеме D6 К561ЛЕ5 для БК-0010(.01) (рис. 2) и на микросхеме D2 К561ПУ4 — для БК-0011М (рис.3). Эти сигналы формируются в соответствии с временными диаграммами при установке высокого уровня на входе устройства (09)D6.3 или (09)D2.3, соответственно. Если на нем установить сигнал низкого уровня, не выключая питания компьютера, сигналы DCLO и ACLO устанавливаются также в низкий уровень и происходит останов процессора. (Дальнейшие процессы его запуска те же, что при включении питания БК.) Сделать же это можно с помощью обычного переключателя, который подключает к общему проводу вход устройства. При этом, так как сигнал высокого уровня подается на вход не напрямую, а через резистор, короткого замыкания не происходит (резистор играет роль элемента нагрузки).

Таким образом, установка кнопки перезапуска не представляет трудности ни на БК-0010.01, ни на БК-0011М. Следует только отметить, что рекомендация подпаивать кнопку непосредственно к контактам А1 и А2 разъема системной магистрали является не совсем корректной. Дело в том, что перезапуск изначально предусмотрен в схеме как БК-0010.01, так и БК-0011М, а на плате вычислителя имеется разводка под стандартный переключатель «СТОП-ПУСК» типа П-2К*. В этом нетрудно убедиться, внимательно осмотрев участок платы между разъемом XT4 и микросхемой D12 на БК-0011М и между микросхемами D12 и DS18 на БК-0010.01 (в старой модели БК-0010 переключатель на этом месте уже установлен).

Однако пользователи БК, в особенности БК-0011М, могут столкнуться и с неожиданной трудностью: нажатие на кнопку перезапуска в некоторых (а иногда даже во всех) случаях вызывает зависание компьютера. Причиной является КМОП-микросхема К561ПУ4, у которой при броске входного напряжения проявляется так называемый тиристорный эффект: все транзисторы микросхемы входят в режим тиристоров, при этом через нее течет большой ток, вызывающий быстрый перегрев микросхемы и ее выход из строя.

Решить эту проблему можно подбором микросхемы, поскольку для конкретных ее экземпляров тиристорный эффект проявляется в разной степени, но авторы статьи предлагают более рациональный способ. Для ограничения бросков напряжения следует подключать вход устройства к общему проводу не напрямую, а через резистор с сопротивлением в несколько кОм (рис. 4). В каждом конкретном случае следует подобрать резистор возможно меньшего номинала, так как время разряда через него конденсатора С5 с ростом сопротивления увеличивается и при номинале 10 кОм может достигнуть 0.5 с.

Одна из особенностей БК-0011(М) состоит в том, что перезапуск в этой вычислительной машине происходит по адресу 140000. Но при работе в режиме эмуляции БК-0010 (именно так функционирует большинство операционных систем: MKDOS, ANDOS, NORD) перезапуск по этому адресу приводит к выходу в монитор БК-0011, а для передачи управления монитору БК-0010 и возврата в операционную систему требуется вводить с клавиатуры определенную последовательность команд (например **4;1С 100000G**). При этом теряется установленная до перезапуска палитра, что особенно неприятно для владельцев цветных мониторов. Поскольку пользователи БК-0011М в основном работают именно с «десяточными» программами, многие из которых

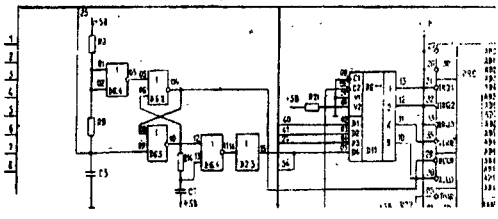


Рис. 2. Фрагмент принципиальной схемы БК-0010(.01)

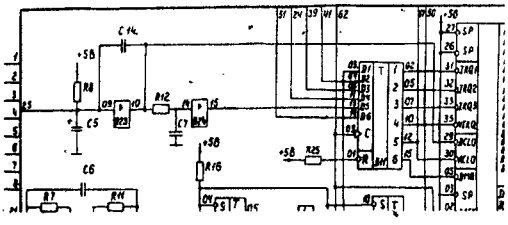


Рис. 3. Фрагмент принципиальной схемы БК-0011М

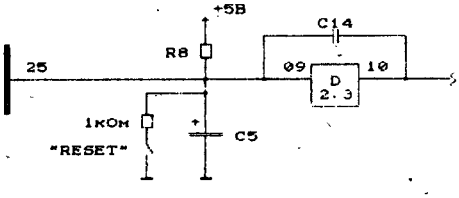


Рис. 4

* О подключении кнопки «RESET» на месте отсутствующего переключателя «СТОП-ПУСК» в БК-0010(.01) рассказано в №1 за 1994 г., с. 71, «От редакции». — *Прим. ред.*

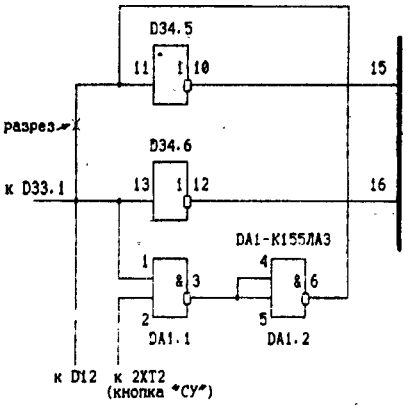


Рис. 5

(в основном игры) не имеют цивилизованного выхода, процедура возврата в ОС становится утомительной. Эту проблему можно решить установкой перезапуска по адресу 100000. (Некоторые программы, например TEAR OUT, позволяющая копировать содержимое экрана в файл или на принтер, требуют перезапуска именно по этому адресу.)

Идея перезапуска по адресу 100000 состоит в том, что во время повторного старта процессора обнуляется четырнадцатый разряд регистра 177716. В результате при переписывании разрядов 8—15 этого регистра в счетчик команд там оказывается адрес начального пуска 100000, а не 140000. Существуют различные реализации этой идеи, однако они или безграмотны схемотехнически, или неудобны в эксплуатации. В частности, непременным является требование просверлить в корпусе БК отдельное отверстие под кнопку нового перезапуска. Предлагаемый авторами данной статьи вариант свободен от всех этих недостатков и абсолютно не изменяет внешнего вида вычислительной машины, поскольку в качестве второй кнопки используется клавиша «СУ», а перезапуск по адресу 100000 осуществляется одновременным нажатием «СУ» и обычной кнопки перезапуска. На рис. 5 указан фрагмент схемы БК-0011М с доработкой, реализованной на микросхеме К155ЛА3 (DA1).

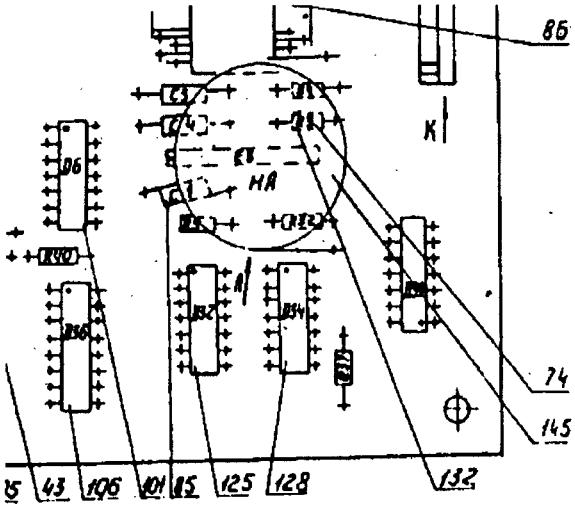


Рис. 6

Аналогичным образом возможна реализация перезапуска по другим, более «экзотическим» адресам (40000, 160000 и т. д.) с использованием иных нефиксирующихся клавиш (например, «АР2») или их комбинаций, однако это не представляется целесообразным ввиду ограниченных возможностей применения этих видов перезапуска.

Скажем несколько слов и о практическом проведении доработки. Микросхему DA1 целесообразно установить на плате БК-0011М поверх имеющейся там микросхемы D39 (рис. 6). Предварительно следует отогнуть в горизонтальное положение все выводы микросхемы DA1, за исключением седьмого и четырнадцатого, которые припаиваются к аналогичным выводам микросхемы D39. Указанный на рис. 5 разрез печатной дорожки желательно сделать под корпусом микросхемы D34 (например, иглой), если же это не удастся, нужно перерезать и отогнуть одиннадцатый вывод D34.

Нам пишут

Ю. В. Котов,
Москва

Реплика про графический курсор

Хотелось бы дополнить сведения о графическом курсоре, опубликованные в статье И. В. Христофорова, в журнале «Персональный компьютер БК-0010 — БК-0011М» №3 за 1994 г. (с. 4). Вопреки заявлению автора этой статьи EMT-функции с номерами 24 и 26 могут работать не только с текстовым, но и с графическим курсором БК. Обе эти функции вначале считывают значение ячейки 50 и далее работают с курсором в соответствии с ее содержимым.

Так, в подпрограмме реализации EMT 24, прошитой в ПЗУ с адреса 107272, по адресу 107332 расположены команды: 105767, 70512, 1020 — опрос байта 50. Для текстового курсора обработка продолжается со следующей команды, для графического — с адреса 107400. При этом маска из ячейки 155, перенесенная в регистр процессора R0, сдвигается влево на количество битов, соответствующее содержимому трех младших разрядов заданной в регистре R1 координаты X. Затем полученное значение переносится в ячейку 154, а из старших разрядов координаты X и координаты Y (с проверкой на выход за пределы экрана) компонуется и заносится в ячейку 166 относительный адрес курсора.

При отработке функции EMT 26 (с адреса 107502) также опрашивается байт 50 и, если он нулевой, выполняется переход к адресу 107526, где расшифровывается содержимое ячейки 166. В частности, маска из ячейки 154, перенесенная в регистр R0, сдвигается влево, пока ненулевой разряд не попадет в бит переноса. Одновременно инкрементируется содержимое регистра R1, тем самым производится подсчет координаты X (младшие разряды). Далее учитывается режим экрана (32/64 символа в строке) и определяется координата Y. И наконец значения координат X и Y возвращаются в регистрах R1 и R2. Поэтому, как правило, нет необходимости вручную расшифровывать содержимое ячеек 154 и 166.

Отдельно хочется сказать несколько слов и о текстовом курсоре. При его обработке формируется или анализируется содержимое ячейки 156. Вообще говоря, текстовый курсор вовсе не обязательно располагается там же, где и графический. Хотя при переходе в режим «ГРАФ» графический курсор принудительно выводится на месте левой верхней точки текстового курсора, обработка их координат ведется независимо.

Говоря о графическом курсоре, отметим значительное неудобство режима «ГРАФ»: в нем допускается нажатие только цифр и клавиш управления курсором, остальные же клавиши блокированы. В приведенной ниже USR-подпрограмме этот недостаток исправлен. Функция EMT 24 здесь используется для вывода изображения графического курсора (в ячейку 50 временно заносится значение 377). Текущие координаты X и Y размещаются в ячейках 37300 и 37302, поэтому для их возврата вызов EMT 26 не требуется. При нажатии клавиш перемещения курсора его координаты модифицируются, а курсор на экране соответственно сдвигается. Если клавишу удерживать нажатой, скорость движения курсора увеличивается. Нажатие других клавиш приводит к выходу в БЕЙСИК, где возвращенное оператором USR значение кода нажатой клавиши можно использовать для каких-либо интерфейсных целей. Графический курсор перед выходом в БЕЙСИК гасится или перезадается в углу экрана.

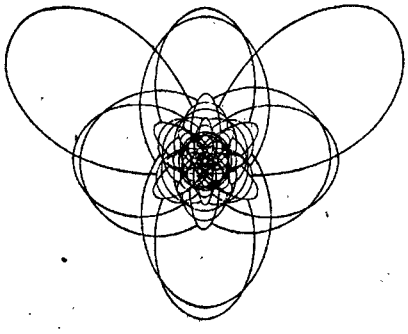
```
37010: 5004 5037 56 4737 37200 5004
; режим и первое обращение к внутренней подпрограмме
37024: 104006 10003
; прием кода с клавиатуры, выделение кодов
;  перемещения и модификация координат
37030: 22703 31 1003 5237 37300 421
       22703 10 1003 5337 37300 413
```

```
        22703 33 1003 5237 37302 405
        22703 32 1013 5337 37302
        37106: 5204 4737 37200
; повторный вызов подпрограммы
        32737 100 177716 1742
; анализ нажатия клавиши и возврат на повторный цикл
        137 37022
; возврат для другого перемещения
37130: 10315 5001 5002 4737 37210 207
; перенос курсора в угол (0,0) и выход из функции USR
. . .
37200: 13701 37300 13702 37302
; внутренняя подпрограмма: загрузка текущих координат
;  в регистры
37210: 12737 377 50 104024 5037 50
; вызов EMT 24 в режиме "ГРАФ"
        12701 40000 22704 7 100002
        12701 400 77101
; регулируемая задержка времени (константы можно менять)
        207
; конец подпрограммы
```

В БЕЙСИК-программе надо обеспечить вызов функции USR с адреса 37010, предварительно записав в ячейки 37300 и 37302 начальные значения координат графического курсора (например, центра экрана). После возврата из функции в приведенном ниже простейшем фрагменте БЕЙСИК-программы нажатие клавиши «0» означает высвечивание точки, «1» — проведение отрезка от предыдущей точки до текущей, «2»—«5» — задание цвета линий, в том числе цвета фона (черный), «6» — закраска текущим цветом замкнутой области с границей того же цвета. Цвет курсора при этом не изменяется. В верхних строках экрана после выхода из USRподпрограммы печатаются значения координат курсора.

Перед запуском БЕЙСИК-программы нужно освободить память для подпрограммы с помощью оператора CLEAR ,&O37000.

```
10 DEF USR=&O37010
20 POKE &O37300,120
30 POKE &O37302,120
40 C%=1%
50 I%=USR(0%)
60 XP=PEEK(&O37300)
70 YP=PEEK(&O37302)
80 LOCATE 1,1
90 ? XP;YP;"     " '4 пробела
100 T%=I%-48%
110 IF T%<6% AND T%>1% THEN C%=T%-2%
120 IF T%=0% THEN PSET (XP,YP),C%
130 IF T%=1% THEN LINE -(XP,YP),C%
140 IF T%=6% THEN PAINT (XP,YP),C%,C%
...
...
1000 GOTO 50
```



Примечание. Перед работой с графическим курсором желательно нормализовать рулонное смещение экрана. Из-за какой-то ошибки в прошитом в ПЗУ БК программном обеспечении может произойти самопроизвольное снятие режима «ГРАФ», если адрес ячейки экранного ОЗУ, соответствующей графическому курсору (содержимое ячейки 170), изменяется с максимального значения (77776) на минимальное (40000) или наоборот. При ненормализованном рулонном смещении эта граница адресов может находиться посередине экрана, что делает ее труднообнаружимой.

НАЧИНАЮЩИМ ПОЛЬЗОВАТЕЛЯМ

Машинные коды

Большинство пользователей БК-0010(.01) и БК-0011(М) начинают освоение программирования с ФОКАЛа или БЕЙСИКа. О том, как работать с этими языками, достаточно подробно рассказано в прилагаемых к БК руководствах, да и книг по ним тоже написано немало. (Следует еще учесть и то, что большинство так называемых языков программирования высокого уровня, к которым относятся ФОКАЛ, БЕЙСИК, недавно появившиеся на БК ПАСКАЛЬ, ЛОГО, ФОРТ и другие, соответствуют некоему общепринятому стандарту. Поэтому обладатели БК могут подчерпнуть весьма полезные сведения, например, по БЕЙСИКу практически из любой книги об этом языке, даже если речь в ней идет о другом типе компьютера.) Но большая часть программ для БК, среди которых — самые интересные и красиво оформленные игры, наиболее мощные и использующие максимум возможностей компьютера прикладные и системные разработки (базы данных, электронные таблицы, «записные книжки», дисковые операционные системы, текстовые и графические редакторы, да и трансляторы с самих языков высокого уровня) — все это программы в кодах. Для игр разница между БЕЙСИКом (или ФОКАЛом) и машинными кодами становится очевидной с первых же дней после покупки БК — достаточно поработать с программами, прилагаемыми к компьютеру на кассетах. И по этой причине умение работать с кодами становится главной мечтой любого начинающего БКмана. И вот здесь его поджидают значительные трудности — не потому, что сложна изучаемая тема, а потому, что почти невозможно найти подходящие источники информации. Не будем утверждать, что их нет вообще: в тех же прилагаемых к БК руководствах имеются многие необходимые сведения: описания системных регистров и EMT-функций, таблицы команд ассемблера и т. д. Но способ изложения этого материала подходит лишь для уже более или менее подготовленных пользователей. А чтобы стать «подготовленным», сначала нужно пройти стадию «начинающего». Получается замкнутый круг, из которого не так-то легко вырваться. Книги же и журналы тоже не всегда оказывают должную поддержку: во-первых, специальной литературы о БК очень мало, а во-вторых, и там сведения рассчитаны тоже на «подготовленных». (Автору этих строк еще памятны его первые месяцы общения с БК, когда перед ним лежал один из выпусков журнала «Наука и жизнь» с готовой программой в кодах, превращающей БК-0010 в «электронный музыкальный инструмент», а он, то есть автор, не знал даже, как эти коды ввести в БК! Да и сегодня многие наши читатели испытывают те же трудности.)

Надеемся, что эта статья поможет начинающим БКманам в освоении их «электронного друга».

Что такое машинные коды

Любой компьютер, начиная с самых первых, еще неуклюжих моделей МИНСКов или ЭНИАКов и кончая суперсовременной IBM AT–586, может понимать только один язык команд — цифры. Это заложено в самих принципах цифровой вычислительной техники. А из цифр ЭВМ понимает только две: нуль и единицу. Поэтому для компьютера любое числовое данное записывается как последовательность нулей и единиц — в так называемой двоичной системе.

А что такое двоичная система? Вспомним, как мы привыкли записывать числа (для начала рассмотрим целые неотрицательные). Например, число «триста сорок восемь»: запись 348 означает «три, умноженное на сто, плюс четыре, умноженное на десять, плюс восемь». Любое число можно разложить таким способом на единицы, десятки, сотни, тысячи и т. д., этому учат еще в первом классе. Но десять, сто, тысяча и т. п. — это десять в степени 1, 2, 3..., а единица — десять в нулевой степени. А если пронумеровать позиции в числе от нуля справа налево, то получается простая и удобная формула разложения любого целого десятичного числа без знака: число есть сумма произведений цифр на 10 в степени, равной позиции цифры. Для человека, с детства привыкшего работать «просто с числами», такая формула кажется излишним усложнением и ненужной формальностью, но для компьютера она — настоящий «букварь первоклассника». Ведь точно так же, как мы раскладывали только что число по степеням десяти (поэтому наша система счисления и называется десятичной, а 10 именуется «основанием системы счисления» или просто «основанием»), мы можем разложить его и по основанию два, так что оно станет понятно любой ЭВМ! Например, 17 можно представить как 2^4+1 или, если использовать умножение на 1 и на 0 (что в обычной математике тоже кажется лишенным

всякого смысла), можно получить «полную формулу разложения»: $1*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 1*2^0$. А отсюда уже недалеко и до двоичного числа — перепишем в том же порядке только цифры 1 и 0 и получим двоичное представление числа 17: **10001**.

Несколько сложнее переводятся в двоичный вид целые числа со знаком и числа с плавающей запятой. Эти преобразования мы здесь рассматривать не будем — все они достаточно подробно описываются в статье Ю. А. Зальцмана (№2 за 1994 г., с. 9) и во многих книгах.

Однако на компьютерах приходится иметь дело и с большими числами (на БК — от 0 до 65535 без знака или от −32767 до +32767 со знаком), для которых нужно писать длинные строки из нулей и единиц. Для программиста это большое неудобство, поэтому при записи чисел в программах для ЭВМ часто используются другие системы счисления, являющиеся компромиссом между «пожеланиями» человека и компьютера: восьмеричная (по основанию 8) и шестнадцатеричная (по основанию 16; в качестве «цифр» для 10, 11, 12, 13, 14 и 15 используются латинские буквы A, B, C, D, E и F). Как выполняется перевод десятичных чисел в восьмеричные или шестнадцатеричные и обратно, думаем, читатели после всех предшествующих объяснений догадаются сами, а вот почему это компромисс между человеком и машиной, поговорим. Взгляните повнимательнее: ведь и 8, и 16 — это два в третьей и в четвертой степени соответственно. Уже этот факт заставляет надеяться на какие-то преимущества при переводе из восьмеричной и шестнадцатеричной систем в двоичную и обратно. Верно, эти операции намного более просты, чем для десятичных чисел. Каждая восьмеричная цифра просто заменяется на триаду двоичных в соответствии с табл. 1, а шестнадцатеричная — на четыре двоичные цифры (тетраду) по табл. 2. Аналогично двоичное число может быть разбито на триады или тетрады, каждая из которых заменяется восьмеричной или шестнадцатеричной цифрой.

Таблица 1

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

Таблица 2

Шестнадцатеричная цифра	0	1	2	3	4	5	6	7
Двоичная тетрада	0000	0001	0010	0011	0100	0101	0110	0111
Шестнадцатеричная цифра	8	9	A	B	C	D	E	F
Двоичная тетрада	1000	1001	1010	1011	1100	1101	1110	1111

Как легко подсчитать, число 65535, требующее в двоичной системе записи шестнадцати нулей и единиц, в восьмеричной будет состоять из шести цифр, а в шестнадцатеричной — всего из четырех. Компьютер же (а точнее, его системная программа — монитор), получая на входе запись из восьмеричных или шестнадцатеричных цифр, легко выполняет их замену на триады или тетрады по записанной в его память таблице, не выполняя умножений и возведений в степень.

Воздав должное системам счисления, пойдем дальше. Говоря о числах, мы как-то забыли, что кроме них нам могут понадобиться также буквы (для записи текстов или проверки, какая клавиша нажата) и команды операций над числами и буквами. Вначале разберемся с последними (правильнее, впрочем, будет говорить не о буквах, а о символах, так как кроме русских и латинских заглавных и строчных букв в БК есть еще полуграфика, символы математических операций и пунктуации и т. д.). Еще на заре вычислительной техники программисты договорились перенумеровать все используемые на компьютере символы по порядку, расположив их в виде таблицы, а число, являющееся порядковым номером символа в этой таблице, передавать компьютеру в качестве кода данного символа. Заметим, что в этих таблицах некоторые коды соответствуют не отображаемым символам, а командам управления выводом на дисплей, перемещения курсора и т. п. либо просто не задействованы. (Некоторые из них также используются по отдельной договоренности для управления печатью на принтере.) Таблицы символов (называемые также таблицами кодировок или просто кодировками) — стандартные, правда, стандартов имеется несколько для различных стран и моделей ЭВМ. На БК кодировка соответствует стандарту КОИ8, принятому на многих отечественных компьютерах, а ее первая часть (коды от нуля до 127 включительно) — международному стандарту ASCII.

Разобравшись с тем, как компьютер «понимает» текст, поговорим о командах. Любая из команд, задаваемых программистом и указывающих процессору БК, что нужно делать с данными (мы уже знаем, что данные — это всегда числа), как многие уже, наверное, догадались, — тоже числа! По некоторой изначальной договоренности (или, скорее, по воле разработчиков конкретной модели процессора) каждому из выполняемых процессором действий, так же как и символам, присвоен свой «порядковый номер». А значит, и здесь мы имеем дело с кодами. На первых компьютерах коды команд действительно были всего лишь порядковыми номерами в некоей условной таблице. Например, так (табл. 3):

Таблица 3

Действие	Код
сложение	00
вычитание	01
умножение	10
деление	11

Тогда любое математическое выражение, скажем (5+3)*8−(3+7)/2, записывалось в программе в виде последовательности таких строк (исходные числа заранее заносились в ячейки оперативной памяти, как и результат в процессе выполнения программы):

```
00  0101  0011 @1000   ; сложить 5 и 3 и поместить сумму в
                       ;  ячейку с адресом 1000
00  0011  0111 @1002   ; 3 + 7 -> в ячейку 1002
10 @1000  1000 @1000   ; число из ячейки 1000 умножить на 8
                       ;  и поместить результат снова
                       ;  в ячейку 1000
11 @1002  0010 @1002   ; число из ячейки 1002 разделить на 2
                       ;  и поместить результат снова в 1002
01 @1000 @1002 @1000   ; вычислить разность промежуточных
                       ;  результатов и поместить в ячейку 1000
```

Как видим, запись довольно громоздкая, поэтому позже во многих ЭВМ (и в том числе на БК) в коды команд стали включаться не только их «порядковые номера», но и указание, где машине искать обрабатываемые данные, например:

```
012737
000123
005432
```

Первое число здесь — код команды, предписывающей процессору занести следующее по порядку число (123) в ячейку памяти с адресом, указанным по порядку третьим (5432). При этом в команде 012737 первая часть («01») — это и есть «порядковый номер» в условной таблице, о чем мы говорили выше, а пары цифр «27» и «37» указывают на расположение исходного числа и отведенного под результат места. (Заметим, что все три рассмотренных кода — восьмеричные числа. Кроме того, нужно сказать, что на самом деле кодирование команд и способов доступа к операндам и результату гораздо более сложно, чем показано здесь. Для его подробного обсуждения понадобилось бы немало места на страницах журнала, времени, а также потребовалась бы определенная подготовка читателя. Да и, кроме того, этот вопрос достаточно хорошо рассмотрен в статьях Ю. А. Зальцмана — см. выпуски №1,94—№3,95.)

А теперь, когда мы поняли, как выглядят «с точки зрения» компьютера (а точнее, его процессора) все данные и команды, можно поговорить о том, что представляет собой собственно программа. На БК коды команд, а также все необходимые данные и отведенные под константы и переменные ячейки памяти размещаются по порядку друг за другом (а иногда отдельными фрагментами) в одном общем для всех них объеме памяти. Такая последовательность кодов команд процессора, числовых констант, кодов символов и адресов ячеек памяти, содержащих исходные операнды или отведенных под результат, и называется программой в машинных кодах. Для начинающего пользователя не требуется умение составлять такие программы или расшифровывать их, нужно лишь суметь ввести готовую последовательность кодов в БК, записать на магнитофон или диск и запускать на выполнение.

И наконец, сообщим читателям вещь совсем уж «феноменальную»: машинные коды способны понимать (расшифровывать вручную) лишь немногие, а программированием в машинных кодах сегодня практически никто уже не занимается.

«А как же все новые программы в кодах? — спросите вы. — Откуда же они берутся?» Ответ на этот вопрос содержится в одном-единственном слове: «ассемблер». Оно настолько часто повторяется в книгах и журналах (и не только о БК), что сегодня его, наверное, знают все.

Ассемблер как язык программирования представляет собой набор сокращенных до трех или четырех букв названий команд процессора, а также перечень обозначений, указывающих на местонахождение констант и значений переменных в памяти. Для программиста такой способ записи удобен как при написании программ, так и для запоминания команд и выполняемых ими действий. А для машины? Специальная программа, называемая ассемблер-транслятором или просто ассемблером (второе значение этого «волшебного» слова), переводит последовательность записей на ассемблере в последовательность соответствующих им машинных кодов. Таким образом, язык ассемблера — это как бы «второе лицо» машинных кодов, более удобное для программиста и вместе с тем понятное (после обработки транслятором) для процессора. Так вот откуда берутся программы в кодах! Все они (за очень редким исключением) изначально пишутся на ассемблере, а потом транслируются в готовые к работе кодовые модули, с которыми пользователи и имеют дело.

Отсылая тех читателей, кто хотел бы научиться программировать на ассемблере (а значит, и в кодах), к циклу статей Ю. А. Зальцмана, продолжим разговор обсуждением следующей темы: как ввести в компьютер готовую программу в кодах. В качестве примера возьмем небольшую подпрограмму красивой очистки экрана, которую можно использовать в качестве USR в БЕЙСИКе, ввести в состав программы на ассемблере или запускать независимо. Начальный адрес при ее размещении в памяти (в данном случае указан 1000) может быть любым — подпрограмма перемещаема.

```
НАЧ. АДРЕС: 001000          ДЛИНА : 000120

010546  010446  004437  110346  012705  040000  012700  000003
005004  012703  100256  041315  062705  000072  020527  100000
103020  062703  000002  005713  001002  012703  100256  077415
005300  001402  005004  000757  004437  110362  012604  012605
000207  162705  040000  000755  000000  000000  000000  000000

          КОНТРОЛЬНАЯ СУММА : 024340
```

Работа с кодами в режиме ТС

Режим ТС, называемый в документации к БК-0010(.01) «отладочным монитором» или «монитором тестов самодиагностики», — это одна из системных программ, прошитых в ПЗУ. На старой модели БК-0010 с ФОКАЛом режим ТС доступен всегда, а на БК-0010(.01) необходимо пристыковать блок МСТД к разъему системной магистрали. Выход в режим ТС из ФОКАЛа производится следующими командами (жирным шрифтом выделены символы, вводимые пользователем, остальные — подсказки, выдаваемые БК):

```
* <ЛАТ> <ЗАГЛ> Р <ПРОБЕЛ> Т <ВВОД>
+ <РУС> ТС
¤
```

Знак «солнышко» (¤) — это признак того, что вы находитесь в среде режима ТС, и одновременно приглашение к вводу команд.

«Но зачем нам отладочный монитор, — спросит удивленный читатель. — Что же в нем отлаживать?» Этот вопрос следовало бы задать тем, кто писал документацию к БК, только в несколько иной формулировке: почему в ней не сказано ни слова о том, что режим ТС на практике чаще всего используется не для отладки или самодиагностики, а как простейшее средство для ввода в БК программ в машинных кодах? (На БК-0011(М) функции режима ТС включены в состав системного монитора.)

Узнав же, что режим ТС и есть искомое средство для ввода кодов, постараемся научиться с ним работать.

Все команды, кроме запуска на выполнение, в режиме ТС подаются заглавными русскими буквами. Их полный список приведен в прилагаемом к БК руководстве, нам же понадобятся лишь несколько. Рассмотрим их на приведенном выше примере. (При работе в ТС желательно установить режим вывода 32 символа в строке, нажав клавиши «АР2»+«;».)

Вначале нам нужно указать компьютеру начальный адрес вводимой программы. (Если она, как и в нашем примере, перемещаема, все равно нужно вводить ее с какого-либо конкретного адреса, в данном случае — с 1000, после чего записать на магнитофон. А вот потом ее можно будет считывать и запускать с любого начального адреса (в свободной зоне ОЗУ). Неперемещаемые же программы можно загружать и запускать только с указанного начального адреса.) Делается это с помощью команды **<начальный адрес>А** (в нашем примере — **1000А**). Затем нужно ввести команду просмотра текущей ячейки **И**, после чего на экран выдается некое восьмеричное число — прежнее содержимое ячейки ОЗУ по этому адресу.

Чтобы ввести в нее требуемый восьмеричный код (в данном случае первый по порядку), нужно набрать его и нажать клавишу **«запятая»**. Набранное восьмеричное число переписывается в данную ячейку, а на экран выдается содержимое следующей ячейки. Вводим очередной код (второй по счету) и вновь нажимаем на «запятую» и т. д. Заметим, что вводимый восьмеричный код должен состоять из не более чем шести цифр, каждая из которых может быть от 0 до 7.

Вот как это будет выглядеть на экране для приведенного выше листинга подпрограммы очистки экрана (жирным шрифтом выделены символы, вводимые пользователем):

¤1000АИ 000000 **10546,** 177777 **10446,** 000000 **4437,** ...

И так до тех пор, пока не будет введен последний код. После него надо еще раз ввести «запятую» и нажать на «СТОП» — появится приглашение «¤». Программа в машинных кодах введена!

(Кстати, если перед входом в режим ТС, когда на экран выведено приглашение «+», запустить тест 1 («Тест ОЗУ–ПЗУ»), а после его завершения войти в ТС и начать набирать программу, то в качестве прежнего содержимого каждой ячейки будет выводиться ее восьмеричный адрес, что удобно для контроля правильности ввода.)

А что делать, если при вводе допущена ошибка? Нажимая клавишу **«минус»**, нужно вернуться назад (компьютер при этом выводит содержимое ячеек в обратном порядке — в сторону уменьшения адресов), пролистав коды до ошибки и еще немного ближе к началу. А затем нажимаем на «запятую» до тех пор, пока в качестве прежнего значения компьютер не выведет ошибочный код. Теперь набираем правильный, нажимаем «запятую» и продолжаем ввод. Если же ошибка совершена где-то в начале программы (или замечена, когда программа уже вся набрана, лучше всего вновь набрать **<начальный адрес>А**, а затем нажимать на «запятую», пропуская правильные коды, пока не появится ошибочный. После этого надо набрать правильное значение и подтвердить его нажатием «запятой». (Очевидно, читатель уже понял, что во всех случаях ввод «запятой» без набора числового кода означает переход к следующей ячейке с сохранением прежнего содержимого текущей.)

Когда программа набрана, нужно проверить правильность ее ввода. Для этого служит контрольная сумма и команда ее вычисления. Набираем **<начальный адрес>А<длина>ДХ**. Компьютер тут же выводит знак равенства и вычисленное для данного массива значение контрольной суммы. При правильном вводе оно должно совпадать с указанным для данной программы, в противном случае нужно искать ошибку. («Хороший тон» предполагает указание контрольной суммы для каждой из публикуемых программ в кодах, как это и делается в нашем журнале. Если же она не указана, для контроля правильности нужно проверить все введенные коды.) Для нашего примера вычисление контрольной суммы будет выглядеть так:

¤1000А120ДХ=024340

Для проверки правильности ввода кодов можно «пролистывать» их по одному с помощью клавиши «запятая» или вывести листинг с помощью команды **Л**:

¤1000А120Л

На экран выводится кодовый листинг указанной длины, выглядящий почти так же, как публикуемый в журнале, за исключением сведений о начальном адресе, длине и контрольной сумме. В режиме 32 символа в каждой строке содержится по четыре кода, в режиме 64 — по восемь. Если же длина всей программы велика и пролистывание строк на экране не дает их рассмотреть, можно приостановить движение клавишами **«СУ»+«@»** (продолжение вывода — любая клавиша) или выводить листинг по частям длиной 200 (для режима 32 символа в строке). Например:

```
¤1000A200Л
. . . . .      (первая порция кодов)
¤1200A200Л
. . . . .      (вторая порция)
¤1400A200Л
. . . . .      (третья порция)
. . . . .
¤2200A134Л
. . . . .      (последняя порция, неполная)
```

А теперь, если все введено правильно, запишем программу на магнитофон, для чего служит команда **МЗ**. После ввода этой пары символов БК последовательно запрашивает начальный адрес и длину и имя записываемой программы, а затем записывает на ленту одну копию файла (магнитофон должен быть заранее включен на запись, а кассета перемотана к началу требуемого свободного участка ленты), например:

```
¤МЗ
 АДРЕС= 1000 <ВВОД>
 ДЛИНА= 120 <ВВОД>
 ИМЯ= OCHSCR.BIN <ВВОД>
```

(Имя может быть любым, в нашем же примере программа названа OCHSCR.BIN в расчете на последующее использование в БЕЙСИКе. Здесь для набора имени в ответ на приглашение команды МЗ нужно перейти на латинский регистр, а для продолжения работы в режиме ТС — вновь на русский.)

Готово. Программа введена и записана на магнитофон. Теперь ее можно загружать и запускать в мониторе с помощью команд M и S:

```
? M                (или M1000)
  ИМЯ=OCHSCR.BIN <ВВОД>
  OCHSCR.BIN          (если не было ошибки при чтении)
? S                (или S1000)
```

То же можно сделать и в ТС:

```
¤МЧ
 АДРЕС= 1000 <ВВОД>
 ИМЯ= OCHSCR.BIN <ВВОД>
```

Компьютер загружает файл с указанного начального адреса (в данном случае 1000), после чего можно запустить его командой **<начальный адрес>G**:

```
¤1000G
```

(Обратите внимание: это единственная команда в ТС, вводимая латинской буквой!)

При запуске данного примера (кстати, команду G можно было дать и сразу же после набора кодов на клавиатуре, если подпрограмму не предполагается сохранять на магнитной ленте) экран очищается нестандартным способом.

Ввод кодовой программы в отладчике

Отладчик — это специальное программное средство, служащее для отслеживания процесса выполнения программ в кодах (или на ассемблере, оттранслированных в коды) и их отладки, как это и следует из его названия. Однако можно использовать отладчик и для ввода программ в машинных кодах. Сведения о том, как работать с конкретной версией отладчика, нужно искать в прилагаемом к нему текстовом файле с документацией, здесь же мы рассмотрим только основные моменты.

Большинство отладчиков выводят информацию на экран построчно в следующем формате: адрес ячейки памяти, кодировка RADIX (впрочем, ее может и не быть), содержащееся в ячейке восьмеричное значение, его представление в виде пары символов и в виде двух байтов, а справа —

ассемблерная расшифровка последовательности кодов (на рис. 1 показан вид экрана для отладчика-дезассемблера DEBU10 А. Г. Прудковского).

Для ввода кодовой программы в DEBU10 нужно установить курсор на строку, соответствующую ячейке с адресом, равным начальному адресу программы. (Строки можно пролистывать, смещая курсор за верхний или нижний край экрана. Для перехода же на любой адрес нужно нажать «А» и ввести его восьмеричное значение.) Установив курсор в третьем столбце («содержимое»), вводим код и подтверждаем нажатием клавиши «ВВОД» — курсор переносится на строку ниже. Аналогично вводим следующий код и т. д. (Расчет контрольной суммы в отладчике не предусмотрен, поэтому при вводе кодов нужно быть внимательным.)

Когда вся программа набрана, можно записать ее на магнитную ленту (или диск, если DEBU10 запущен из дисковой системы). Для этого служит команда «З». Предварительно нужно рассчитать длину программы (если она не была указана в листинге) по формуле D=AK–AN+2, где D — длина, а AK и AN — адреса последней и первой занятой ячейки (не забудьте, что вычисления производятся по правилам восьмеричной арифметики). Для уверенности можно увеличить длину на несколько байт или округлить ее до ближайшего большего значения. Будьте внимательны: в отличие от режима ТС, в DEBU10 вначале запрашивается длина, а затем адрес и имя.

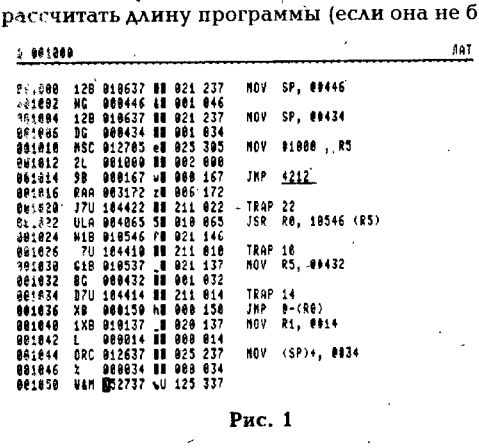


Рис. 1

(Во всех случаях нужно обязательно вводить все шесть восьмеричных цифр числа, дополняя его при необходимости нулями слева. Ошибочно введенную цифру можно удалить с помощью клавиши «ЗАБОЙ». Цифры 8 и 9, а также нецифровые символы в данном случае игнорируются.)

Для запуска набранной программы нужно установить курсор на строку, соответствующую ячейке с начальным адресом, и нажать клавишу «G».

Ввод кодов в БЕЙСИКе

Пользователям БК, работающим с БЕЙСИКом, из всех кодовых программ чаще всего приходится иметь дело с USR-подпрограммами, поэтому попутно мы рассмотрим и основные правила работы с последними. Все, что будет сказано далее, относится к вильнюсской версии БЕЙСИКа, прошитой в ПЗУ БК-0010.01.

Для работы с любой USR-функцией прежде всего нужно освободить для нее место в памяти. По умолчанию БЕЙСИК использует для своих нужд весь массив ОЗУ с восьмеричными адресами от 1000 до 40000. Область от 1000 до 2000 — дополнительный стек БЕЙСИКа, далее следуют служебные ячейки БЕЙСИК-транслятора и сама программа. При трансляции и работе область ОЗУ после текста исходной программы заполняется шитыми кодами и ячейками, резервируемыми под переменные. С противоположной стороны пользовательской области ОЗУ — от адреса 40000 в сторону уменьшения адресов располагается генерируемый БЕЙСИКом список номеров строк программы и адресов начала исходного текста каждой строки и шитого кода для нее. Таким образом, БЕЙСИК заполняет память одновременно снизу и сверху, а посередине, как правило, остается неиспользуемая область.

Чтобы высвободить место для USR-подпрограммы, нужно выполнить оператор **CLEAR**, указав в его втором аргументе значение адреса начала высвобождаемой области. Указанный адрес БЕЙСИК в дальнейшем (до выключения питания) будет использовать вместо значения 40000 как начало своего «растущего вниз» списка номеров строк, а область ОЗУ от данного адреса до 40000 окажется полностью в распоряжении пользователя (рис. 2).

(Первый аргумент оператора CLEAR определяет количество байтов, отводимых в программе для символьных переменных и символьных массивов, — по умолчанию 200. Если в вашей программе символьные переменные используются не очень активно, а дефицит памяти не очень ощущается, так что изменять объем области символьных переменных вы не собираетесь, первый аргумент CLEAR можно опустить, но следующую за ним запятую пропускать нельзя, например можно записать: «**CLEAR ,&O37000**» , но не «**CLEAR &O37000**».)

В освобожденную с помощью CLEAR область можно загрузить готовую кодовую подпрограмму (а также спрайт, закодированную мелодию или другие данные) с магнитной ленты (или диска, если ваш контроллер оборудован «дисковым БЕЙСИКом») с помощью оператора **BLOAD "<имя>",<адрес>** , например **BLOAD "OCHSCR", &O37000** . (Параметр <адрес> нужно вводить

только в том случае, если на магнитную ленту или диск требуемый кодовый фрагмент записывался с другим значением начального адреса. Если же после имени файла (в кавычках) набрать букву **R** (например, **BLOAD "OCHSCR",R,&O37000**), программа в кодах будет сразу же после окончания ее загрузки запущена на исполнение.) Немного позже мы рассмотрим и то, как ввести небольшую кодовую подпрограмму в самом БЕЙСИКе.

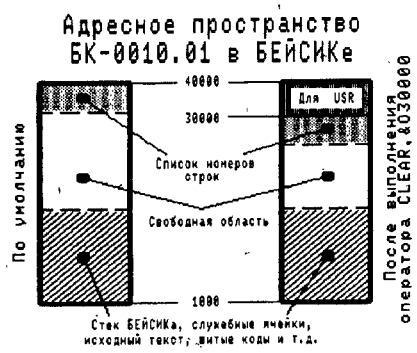


Рис. 2

После этого считанная в память (или набранная в БЕЙСИКе) подпрограмма должна быть подключена к одной из USR-функций оператором **DEF USR<номер> = <адрес начала подпрограммы>** (для USR0 номер «0» можно не указывать). Отметим, что USR-функции БЕЙСИКа изначально представляют собой лишь список независимых друг от друга «условных имен» от USR0 до USR9. Оператор DEF USR обеспечивает «привязку» конкретной кодовой подпрограммы к одному из этих имен, по которому позже можно производить ее вызов из БЕЙСИКа. Независимость же имен друг от друга позволяет указывать в DEF USR любой свободный пока номер. Если же USR с этим номером уже задействована, оператор DEF USR переопределяет данный USR на новую подпрограмму.

После того как USR-функция определена, ее можно вызвать на выполнение с помощью оператора присваивания, скажем **B%=USR(A%)** (для целых чисел) или с помощью **PRINT USR(A%)** . (Перед вызовом не забудьте занести в требуемые ячейки с помощью POKE аргументы, если они требуются.) В первом случае портится содержимое переменной B% (если, конечно, речь не идет о возврате в нее результата работы USR-подпрограммы), а во втором возвращаемое подпрограммой USR значение выводится на экран, что не всегда желательно. На практике приходится выбирать одно из этих двух зол.

Сама USR-подпрограмма представляет собой обычную подпрограмму на ассемблере, оттранслированную в коды. (Единственное существенное требование — подпрограмма должна быть перемещаемой или рассчитанной на тот начальный адрес, который будет ей выделен в БЕЙСИКе. Если же нужно одновременно использовать несколько кодовых подпрограмм, следует проследить, чтобы при загрузке с требуемых адресов они не затирали друг друга.) Последним оператором подпрограммы, обеспечивающим возврат в БЕЙСИК, должен быть **RTS PC** (код **207**), а содержимое используемых в ней регистров процессора желательно сохранять в стеке.

Единственная специфика USR-подпрограмм, в отличие от используемых в ассемблере, состоит в передаче параметров. БЕЙСИК позволяет передать в USR-функцию один аргумент и вернуть одно значение, а кодовая подпрограмма должна считывать его из ячейки, адрес которой записан в регистре R5, и туда же заносить результат, если он нужен (имеется в виду передача и возврат целых чисел, для текстовых строк и чисел с плавающей запятой все несколько сложнее). Если же нужно передать более одного аргумента (также в случае целых чисел), можно отвести под них ячейки ОЗУ с заранее оговоренными адресами, к которым обращается USR-подпрограмма и куда данные заносятся операторами **POKE <адрес>,<значение>** . Аналогично можно возвращать несколько результатов, извлекая их в БЕЙСИК-программе с помощью **PEEK(<адрес>)**.

Кроме PEEK и POKE, при работе с кодами в БЕЙСИКе могут пригодиться и функции чтения/записи кода по маске INP(<адрес>,<маска>) и OUT <адрес>,<маска>,<код> . Маска — это некоторое число (обычно двоичное — &B или восьмеричное — &O), единичные биты которого указывают, к каким битам ячейки с заданным адресом относится данное действие. При чтении по маске каждый бит результата, которому соответствует единичный бит маски, копируется из «одноименного» бита ячейки, а биты, соответствующие нулевым битам маски, в числе-результате обнуляются. При записи в биты ячейки, соответствующие единичным битам маски, переписывается содержимое «одноименных» битов аргумента <код>, а соответствующие нулевым битам маски остаются без изменения. Примеры:

1. A=PEEK(&O30)

@#30: 100112_8

Тогда A= 100112_8

2. POKE &O40000,&B1100110011001100

@#40000 (было): 0000000011000011_2 (например),

@#40000 (стало): 1100110011001100_2

3. A=INP(&O1000,&B1100110010100101)

```
@#1000:   10100101111110000 (например)
Маска:    11001100101001O1
Тогда A=  10000100010100000
```

(фактически данный оператор аналогичен записи PEEK(&O1000) AND &B1100110010100101)

4. OUT &O10000,&B1100110010100101,&B1010010111110000

```
@#10000 (было):  1111000011000011 (например)
Маска:           1100110010100101
Код:             1010010111110000
@#10000 (стало): 1011010011100010
```

А теперь расскажем, как вводить небольшие кодовые подпрограммы в самом БЕЙСИКе. Сделать это можно двумя способами (есть еще и третий — воспользоваться БЕЙСИК-программой PATCH из прилагаемой к БК при продаже кассеты, но на практике ею почти никто не пользуется: проще и быстрее ввести коды вручную, чем искать PATCH на кассете и грузить с магнитофона). Во-первых, можно заносить коды в память непосредственно, с помощью ряда операторов POKE, например:

```
POKE &O37000,&O10546
POKE &O37002,&O11446
POKE &O37004,&O4437
```

и т. д.

Это можно сделать в диалоговом режиме или (если длина кодового модуля небольшая) записав операторы POKE в виде программных строк.

Другой способ, чисто программный, состоит в записи всех кодов (не забудьте дополнять каждый из них обозначением числа восьмеричного типа «&O») в составе одного или нескольких операторов **DATA** с последующим считыванием в цикле с помощью **READ** и записью в память через все те же **POKE**, например:

```
10 DATA &O10546, &O10446, &O4437, ... ' и т. д.
20 FOR I%=<адрес начала> TO <адрес начала + длина> STEP 2%
30 READ A%
40 POKE I%,A%
50 NEXT I%
60 DEF USR=<адрес начала> ' определяем новую USR-функцию
```

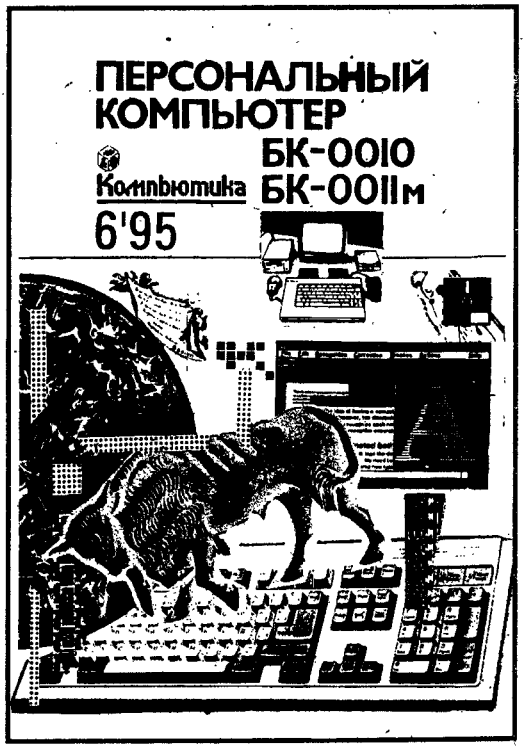
В любом случае введенную кодовую подпрограмму можно записать на магнитофон оператором **BSAVE"<имя>",<адрес начала>,<адрес начала + длина>**, чтобы позже не вводить ее повторно.

* * *

Возможно, приведенные в данной статье советы и рекомендации кому-то покажутся черезмерно простыми, а кому-то — слишком сложными, в зависимости от индивидуального уровня подготовленности. Кроме того, все сказанное выше — это не «учебник», а скорее «методичка»: повторяя описанные здесь действия, можно ввести и запустить любую кодовую программу, но, если вы захотите получить более подробную информацию или научиться разрабатывать кодовые (а точнее, ассемблерные) программы самостоятельно, следует обратиться к другим книгам и статьям нашего журнала (и иных изданий, публиковавших ранее материалы о БК). Но не забудем, что любая, самая долгая дорога всегда начинается с первого шага, и надеемся, что эта статья поможет вам сделать первый шаг в освоении кодов и ассемблера.

Планируемое содержание выпуска 6'95:

- Практикум на ассемблере *(продолжение)*
- Руководство системного программиста БК-0011(М) *(окончание)*
- СПРАВОЧНЫЙ ЛИСТОК: термоструйные печатающие головки для принтеров 6312
- Самодельный фотопистолет для БК
- Электронный диск
- Полнооконный интерфейс для БЕЙСИКа и ассемблера
- Работа с дисковым БЕЙСИКом БК-0010.01
- NORD для винчестера — мнение автора
- Vortex 4.0: отзывы пользователей
- Начинающему пользователю: драйверы принтеров



Этот список статей отражает перечень готовых к печати в №6/95 материалов из «редакционного портфеля». Реальное содержание выпуска ограничено его объемом и может быть изменено.

Впервые на БК!

Белорусскими программистами разработана полноэкранная издательская система БК—PageMaker, работающая на БК-0011М в ОС ANDOS 3.1 с EPSON-совместимыми принтерами (в том числе Robotron и D100M) и обладающая следующими возможностями:

- **верстка страниц формата А4 (до 960 x 768 графических точек);**
- **произвольное размещение на листе любого количества графических элементов и фрагментов текста;**
- **использование графических шрифтов из программ «Журналист» (для IBM) и Comfort;**
- **загрузка текста в формате VorteX и EDASP с автоматическим преобразованием шрифта в графический вид;**
- **импорт/экспорт графики в формате TechnoArt и графических копий экрана;**
- **просмотр уменьшенной копии листа (Preview)**

и многое другое.

Подробное описание программы БК—PageMaker читайте в следующем выпуске журнала.

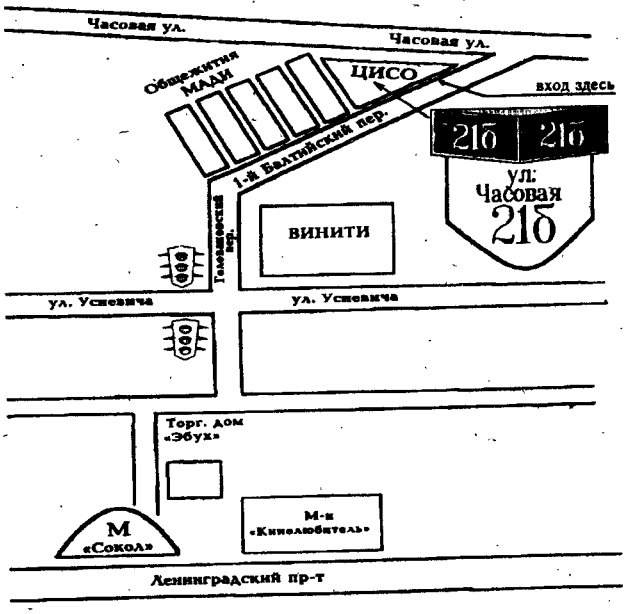
По вопросам приобретения программы обращайтесь в редакцию.

СОДЕРЖАНИЕ

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
Усенков Д. Ю.

Подписано в печать с оригинал-макета издательства «Компьютика» 1.11.95. Тираж 1500 экз. Формат 70×100 1/16. Бумага офсетная. Усл.печ.л. 6,5. Заказ № 273
Цена 5000 руб. (по подписке). В розничной продаже цена договорная.
Отпечатано в типографии "Принт-Сервис" (г.Москва)

Федеральное Управление почтовой связи
при Министерстве связи России

ГАЗЕТЫ ЖУРНАЛЫ

Агентство «Роспечать»