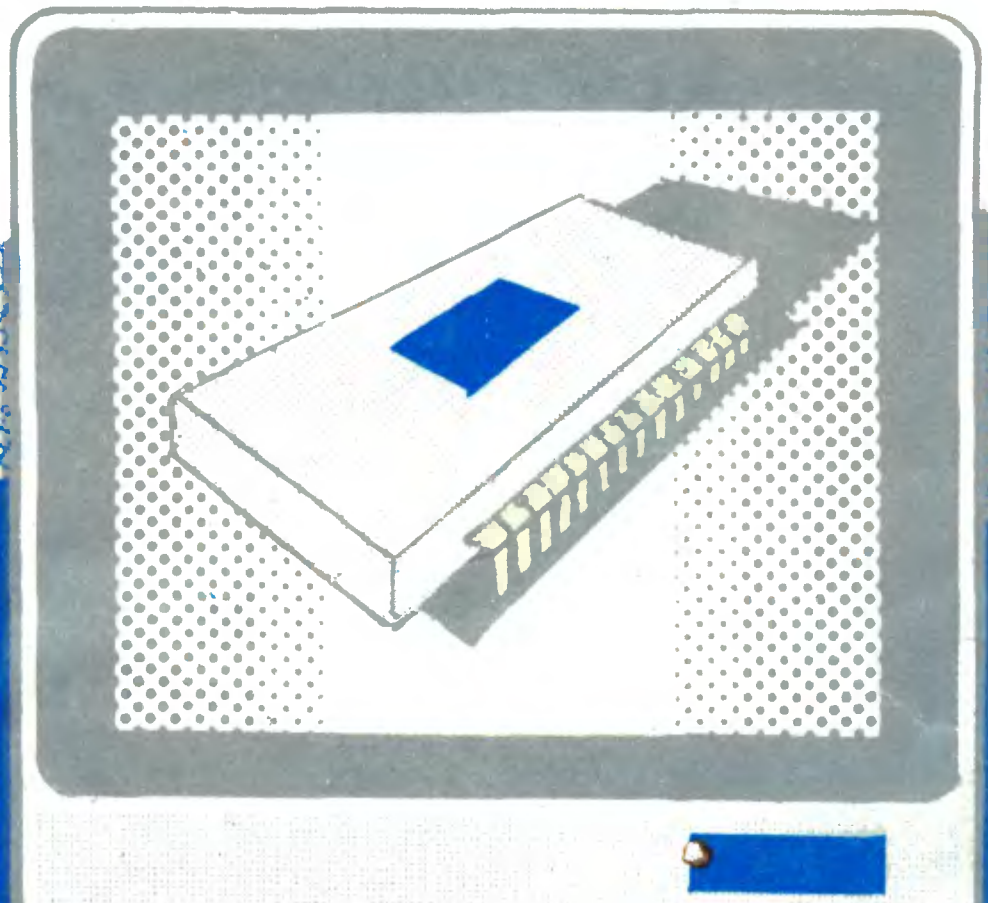
# THE SPECTPUM OPERATIHG SYSTEM

STEVE KRAMER



INTEL
SEEQ
FILIPS

2764 K
27128 K
27256 K
27512 K

РУКОВОДСТВО ХАККЕРА

# ОПЕРАЦИОННАЯ СИСТЕМА

zx spectrum

Psland

# Операционная система СПЕКТРУМ

Стив Кремер

Mikro PRESS

# THE SPECTRUM OPERATING SYSTEM

Steve Kramer

MICRO PRESS FIRST PUBLISHED 1984 BY

*MICRO PRESS CASTLE HOUSE, 27 LONDON ROAD, TUNBRIDGE WELLS, KENT*

Отсканировано
и упаковано 11.08.2007
savelij

ISBN 90346—03—X

**Стив Кремер**

РУКОВОДСТВО ХАККЕРА
ОПЕРАЦИОННАЯ СИСТЕМА «СПЕКТРУМ»

Подписано к печати 22.12.93. Формат 60×84/16
Усл. печ. л. 8,0. Печать офсетная. Бумага офсетная
Тираж 3000. Заказ 989. Цена договорная

Отпечатано на фабрике офсетной печати
г. Обнинск, ул. Королева, 6

# ГЛАВА 1
# ВВЕДЕНИЕ ДЛЯ НАЧИНАЮЩИХ

Так как эта книга первоначально предназначалась для тех, кто уже имеет определенное представление о программировании на языках низкого уровня, мы предварим ее информацией, которая будет полезна не имеющим опыта в программировании, а так же желающим освоить машинные коды без заучивания правил их написания. Если вы из этих, я надеюсь, что эта книга вызовет у вас аппетит и заставит начать изучать что-нибудь о программировании на языках низкого уровня. Некоторые программы, предлагаемые в следующих главах, могут быть легко написаны и использованы при минимальном количестве знаний и опыта. Ленивым, как я, т.е. предпочитающим использовать процедуры содержащиеся в ПЗУ или уже существующие тому, чтобы заново изобретать колесо при необходимости написать программу, я предлагаю обсудить возможность обращения как к 16К ПЗУ, которая есть в СПЕКТРУМЕ, так и 8К ПЗУ микродрайверного интерфейса. В некоторых случаях я дам примеры и пояснения по использованию программ из этих ПЗУ. Мной будут рассмотрены системные переменные и показано наивыгоднейшее их использование. Кроме вышеперечисленного будет объяснено применение процедур прерывания. Если вы хотите самостоятельно преобразовать программы с языков низкого уровня в числовые коды и завести их в память своего компьютера, вам необходимо освоить АССЕМБЛЕР. Могу порекомендовать книгу "PICTURESQUE EDITOR ASSEMBLER" с программой "MONITOR/DISASSEMBLER" из нее. Она проста и быстродействующа, и, кроме того, программу "HIGHSOFT'S DEVPACK 3", которая несколько лучше. Более опытным пользователям подходит "DEVPACK" (который содержит как "MONITOR/DISASSEMBLER" так и "EDITOR/ASSEMBLER"). Я не собирался обучать языкам низкого уровня, так как есть огромное количество книг на эту тему. Две из них я могу порекомендовать: «МАШИННЫЙ ЯЗЫК "SPECTRUM" ДЛЯ НАЧИНАЮЩИХ», автор - RODNAY ZAKS и его же «ПРОГРАММИРОВАНИЕ Z80", которая не связана с использованием "SPECTRUM", но подробна и содержит описание всех доступных операционных кодов. Для того, чтобы вызывать описываемые мной машинные коды из БЕЙСИКА необходимо использовать команды "RANDOMIZE USR NN", "PRINT USR NN", либо "LET V=USR NN", где NN - входная точка в программу в машинных кодах, а V - любая числовая переменная. После выхода из процедуры в машинных кодах, а V - любая числовая переменная. После выхода из процедуры в машинных кодах переменная команды "LET" будет иметь значение пары регистров "BC", а команда "PRINT" выведет значение пары регистров "BC" в текущий выходной

поток. При входе в программу во всех случаях пара регистров "BC" должна содержать "NN", т.е. адрес вызова. Независимо от того, где и когда вызывалась программа в машинных кодах, рекомендуется сохранять содержимое пары регистров "H'L'", так как это необходимо для успешного возвращения в БЕЙСИК. Пара регистров "IY" обычно не применяется, так как ПЗУ использует ее для индексирования системных переменных, но, если сняты точки прерывания и процедуры ПЗУ не использовались до восстановления значения 23610 (5C3Ah), ее можно применять. Пока в вашем распоряжении "SPECTRUM" вы можете пользоваться этой книгой как справочником. Я старался не быть болтливым или ограничивать вас в чем-либо, но надеюсь, что в основном книга содержит необходимую для вас информацию. Очевидно невозможно детализировать все процедуры ПЗУ и способы их применения, поэтому я выбрал те, о которых меня часто спрашивают, и наиболее важные на мой взгляд. Серьезным же программистам в дальнейшем предстоит прочитать «ПОЛНОЕ ДИЗАССЕМБЛИРОВАНИЕ ПЗУ СПЕКТРУМА» (DR. IAN LOGAN).

# ГЛАВА 2

# КАК ВЫЗЫВАТЬ ПОДПРОГРАММЫ 16К ПЗУ

Перед использованием процедур ПЗУ всегда очень важно сохранять "H'" и "L'" регистры и восстанавливать их перед возвратом в БЕЙСИК, при котором "IY" регистр должен содержать адрес системной переменной "ERR NR" 23610(5C3Ah).

---

## ☐ Сообщения: RST 16 (10h)

Строчное значение, код которого содержится в регистре "A", будет напечатан в любом открытом в настоящий момент потоке. Это может быть использовано для вывода управляющих кодов (т.е. TAB, INK, OVER и т.д.; детали см. в Справочнике СПЕКТРУМа).

---

## ☐ Открытие и закрытие потоков (для RST 16 (10h)): CALL 5633 (1601h)

CALL 5633 (1601h) при вызове устанавливает вывод для RST 16 (10h) в поток, указанный в регистре "A". Обычно при:

A=2 - печать идет на основной экран,

A=3 - на принтер, а если

A=1 или A=0 - на нижний экран.

Другие потоки при использовании INTERFACE 1, могут быть распределены для вывода на накопитель или другие элементы. Дета-

ли распределения потоков при решении конкретных задач (например контроля интерфейса для других нужд, кемпстона или интерфейса принтера CENTRONICS) даны ниже в этой главе в разделе «РАСШИРЕНИЕ СИМВОЛОВ ДЛЯ ВЫВОДА».

## ☐ Контроль нажатия "BREAK": CALL 8020 (1F54h)

Эта подпрограмма тестирует ввод "CAPS SHIFT" и "SPACE". При необходимости протестировать только "SPACE" см. далее раздел «ВВОД СИМВОЛА С КЛАВИАТУРЫ», там все изложено в деталях. Сейчас же можно использовать:

```
LD    A,7Fh
IN    A(0FEh)
RRA
JP    NC,PRESSED
```

## ☐ Установка позиции печати при использовании RST 16 (10h): CALL 3545 (DD9h)

Координаты позиции печати перед этим вызовом помещаются в "BC". При этом необходимо, чтобы регистр "B" содержал номер строки экрана в форме:

B = 24 - N (N - номер строки)

(т.е. если B=24, то это верхняя строка экрана; если B=1 - нижняя строка). К сожалению из-за оплошности, допущенной при написании ПЗУ, вы не сможете использовать 23-ю и 24-ю строки основного экрана, поэтому нужно использовать нижний экран, устанавливая выходной поток для RST 16 (10h) равным 1 и используя две верхние строки нижнего экрана.

Регистр C - номер колонки с учетом:

C = 33 - N (N - номер колонки)

(т.е. если C=33, то это левая колонка; если C=2 - правая). Какой бы поток вы не использовали, вызов 5633 (1601h) автоматически дает сообщение о системных переменных для позиции печати. Некоторая осторожность необходима при работе с последней строкой основного экрана, так как при печати в последнюю позицию нижней строки будет выдаваться предупреждающее сообщение "SCROLL?", а это в свою очередь приведет к возвращению в БЕЙСИК, если в ответ будет введено "N" или "SPACE". Кроме этого, любая попытка печати в поток 1 приведет к "SCROLL" нижнего экрана, при заполненном месте для печати, определяемом системной переменной "DFSZ (23659)". Это может привести к непредсказуемым результатам. Очевидно, что при печати на принтер невозможно установить номер строки, т.к. регистр "B" не используется.

### ☐ Стирание всего экрана: CALL 3435 (D6Bh)

Этот вызов очищает весь экран, применяя атрибуты, хранящиеся в системной переменной "ATTRP" - для основного экрана и "BORDCR" для нижнего экрана (23693 и 23624, соответственно; см. СПРАВОЧНИК СПЕКТРУМа).

---

### ☐ Стирание нижнего экрана: CALL 3438 (D6Eh)

Этот вызов очищает нижний экран и сбрасывает атрибуты. Примечание: и 3435 и 3438 устанавливают "DFSZ" в 2 и могут тем самым изменить текущий поток, используемый RST 16, поэтому он должен быть соответственно изменен. При этом текущая позиция печати устанавливается в верхнее левое положение для обоих экранов

---

### ☐ "SCROLL" всего экрана: CALL 3582 (DFEh)

Этим вызовом производится "SCROLL" на одну строку вверх с сохранением позиции печати. Если вы продолжаете печатать в той же строке, то по окончании произойдет как на печатной машинке (печать, начинающаяся в конце страницы переходит вверх после каждого перевода каретки).

---

### ☐ Рисование на экране: CALL 8933 (22E5h)

Точка X, Y указанная в регистре "B" (Y 0-175) и в регистре "C" (X 0-255) будет размещена на экране. Требующийся цвет задается загрузкой системных переменных атрибутов "INK" и "PAPER", или используются текущие атрибуты. Командой

"SET 0,(IY+87)"

может быть установлен "OVER1", отключен же командой

"RES 0,(IY+87)".

Стирается точка командой "SET 2,(IY+87)". Примечание: при этом вызове точки печатаются и на нижний экран, но, если этого не нужно, то требуемый поток определяется вами с помощью команды вызова 5633 (1601h).

---

### ☐ Вывод числа в поток.

Это не простой вызов. В ПЗУ содержится процедура, берущая 16-битное число из двух адресов и выдающая его в десятичной форме, но она неудобна. Байты для нее должны быть размещены в памяти так, как это делает Z80, т.е. в ячейке памяти с меньшим адресом хранится старшая половина 16-битного числа. Вот небольшая программа, что бы делать это:

---

```
LD    DE,ADRES    ; адрес, где размещено число
LD    HL,SPARE    WORD
LD    (HL),D
INC   HL
LD    (HL),E
```

Теперь можно использовать подпрограмму ПЗУ 6696 (1A28h), при этом в "HL" необходимо помещать адрес "SPARE WORD", выводимого в коде "ASCI" в текущий поток. При этом:

Вывод будет корректным, если число целое и меньше 10 000. Пятизначные числа необходимо выводить с пробелом перед числом во избежание их слияния в строке, для четырехзначных этого делать не нужно.

Те, кто читал об этих вызовах в "SPECTRUM POCKET BOOK", но не использовал их практически, назовут меня идиотом. В этой книге автор пишет, что регистр "E" управляет способом представления числа, но применив эту процедуру вы поймете, что я описываю не тот случай, потому что у меня регистр "E" игнорируется. Это довольно двояко: 1) Для обхода первой части ПЗУ, касающейся "E" можно написать свою маленькую программу:

```
PUSH  DE
LD    D,(HL)
INC   HL
LD    E,(HL)
PUSH  HL
EX    DE,HL
LD    E,20h
JP    1A30h
```

Примечание: последней в этом тексте должна быть "JP", лишь после этого вызывается процедура, которую вы написали. В противном случае стек будет испорчен, т.к. на его вершине размещен адрес возврата (командой "PUSH"), а команда "POP" использует именно эти значения. Теперь видно, что регистр "E" уже загружен и автор "SPECTRUM POCKET BOOK" не прав, т.к. это двойной старт процедуры вследствие двойной загрузки. Теперь можно изменить загрузку "LD E" в этой процедуре, но загружаемое число должно быть либо 48 (DEC), в этом случае первыми печатаются нули, либо 25 (DEC), когда при печати игнорируется незначимая часть числа. «Минуточку!», скажете вы, «теперь я могу видеть, почему изменился порядок байта!». Совершенно верно, если вы поменяете порядок загрузки "DE" или просто загрузите "HL" числом, которое не нужно переворачивать этой программкой, то это будет стоящая подпрограмма, которую можно использовать для выведения в поток символов. 2) Есть еще одна часть процедуры ПЗУ (начинающаяся с точки 1A1Bh), которая не будет детализирована. Она просто берет число из пары

"BC" и выводит его без нулей слева или пробелов и, как мне кажется, во многих случаях будет весьма полезна.

Я думаю читатель найдет возможным применять эти процедуры для вывода чисел любых размеров, однако может оказаться проще использовать процедуры, которые берут значения из стека калькулятора (см. Главу 8).

## ☐ Ввод символа с клавиатуры.

Команда RST 56 (38h) используется компьютером для сканирования клавиатуры и присваивания новых значений системной переменной "FRAMES". Вызывается она маскируемым прерыванием. Если требуется лишь подтверждение нажатия, то можно использовать пятый бит "FLAGS" 23611 (5C3Bh); если бит активен, нажатие происходило с момента последнего обнуления этого бита, но бит должен сбрасываться специально.

Код последней нажатой клавиши можно найти по адресу 23560 (5C08h) "LASTK" (см. Главу 4), этого обычно достаточно для ввода с клавиатуры, но есть некоторые неудобства.

Во-первых, смена значений производится 0 раз в секунду, поэтому нельзя использовать команду "RES 5" сразу после просмотра бита, т.к. очевидно, что даже если клавиша была нажата, клавиатура не успеет быть опрошена при прерывании.

Во-вторых, если прерывания отключены, клавиатура никогда не изменит своего значения. Поэтому следующие строки положат в "A" код ключа "NO KEY", или 0, если ничего не было нажато.

```
LD   HL,23611   ;системная переменная
                ;"FLAGS"
RES  5,(HL)
LD   A,FFh
LD   (23552),A  ;23552 часть системной
                ;переменой"KSTATE"
PUSH HL
RST  56
POP  HL
XOR  A
BIT  5,(HL)
JP   Z,NO KEY
LD   A,(23560)  ;"LAST K" системная
                ;переменная, где всегда код
                ;последнейвведенной клавиши
```

В вышеуказанной программе "LD HL,23611" можно опустить и вместо этого может быть проверена (IY+1), ведь "IY" содержит адрес "ERR NR" (23610) (5C3Ah) и ПЗУ использует его для адресации

системной переменной. Вот почему, если вы используете в вашей программе "IY", то вы должны уяснить, отключены ли прерывания или направлены на ваши собственные процедуры и нужно возвратить "IY" к корректному адресу перед использованием ПЗУ или возвращением к режиму нормальных прерываний.

Нельзя определить, было ли нажато сразу несколько клавиш одновременно потому, что "SPECTRUM" игнорирует комбинацию ключей, не имеющую смысла. Для преодоления этого можно написать свою программу сканирования клавиатуры. Для начала нужно просто определить факт нажатия. Что-то подобное делает следующая программка:

```
XOR   A
IN    A,(FEh)
LD    D,31
AND   D
XOR   D
JR    Z,NO          ;клавиши
```

Клавиатура может быть сканирована и более подробно, для этого в "A" загружается значение линии, которая должна быть сканирована перед командой "IN A,(FEh)". В главе 23 справочника "SPECTRUM" описано подробно, каким образом устроена клавиатура, кратко - в приложении "D". Первая строка в каждом случае - это нулевой бит, вторая - бит четыре. Для определения бита опрошенной строки ниже дано шестнадцатиричное значение, загружаемое в регистр "A":

| | | |
|---|---|---|
| CAPS SHIFT | V | FEh |
| A | G | FDh |
| Q | T | FBh |
| 1 | 5 | F7h |
| 0 | 6 | EFh |
| P | Y | DFh |
| ENTER | H | BFh |
| SPACE | B | 7Fh |

Так, например, чтобы проверить, была ли нажата "ENTER", нужно написать следующее:

```
LD    A,0BFh
IN    A,(0FEh)
AND   1
JR    Z,ENTER PRESSED
```

Чтобы узнать, была ли нажата более чем одна клавиша на одной и той же строке, можно использовать логические операторы "AND", "OR" и т.п., или программу тестирования битов (см. CALL 5598 (15DEh)) при вводе обычных символов.

## ☐ Ожидание ввода: CALL 5598 (15DEh)

Это очень полезная программа, позволяющая делать ввод с любого потока и имеющая адрес ввода. Командой 5633, описанной выше, поток открывается для ввода. При работе эта подпрограмма в свою очередь вызывает процедуру ввода текущего канала. При возврате из нее проверяется флаг переноса и, если он установлен, осуществляется возврат в основную программу. Если флаг при возврате не был установлен, то проверяется нуль-флаг и процесс повторяется заново, если он активизирован. Обычно при текущем вводе "SPECTRUM" использует программу ввода с клавиатуры, но если в "CURHL" установлен адрес канала, указывающего на адрес вашей подпрограммы, то ввод будет осуществляется из нее (см. главу 5). При использовании потока 1 (клавиатура и нижний экран), процедура будет ждать, и по нажатии клавиши положит в "A" ее код. Это продемонстрировано в программе "DEBASE" (см. приложение G). Однако есть сложности: при каждом вызове этой процедуры проверяется "TVFLAG 23612 (5C3Ch) IY+2" и если бит 3 активен, то входной буфер копируется в редактируемую часть экрана. Это может быть преодолено использованием процедуры «ВВОД КЛАВИШИ» 4264 (10A8h), а не через программу ожидания ввода. Это применено в "DEBASE" (см. приложение G с метки "INPUT"). Эти строки аналогичны программе «ОЖИДАНИЕ ВВОДА», но всегда контролируется сброс флага "MODE CHANGE". Программа «ОЖИДАНИЕ ВВОДА» используется с метки "INPUTF" и если введена она, или ее часть, то суть проблемы может быть продемонстрирована нажатием "CAPS SHIFT" и "SIMBOL SHIFT" для выхода в "EXTENDED MODE" (расширенный режим). При этом последний веденный с клавиатуры символ появится в нижней части экрана.

При использовании этой процедуры должны быть включены и прерывания, и вызываемая в цикле процедура нормальных прерываний 56 (38h), иначе ввод не произойдет.

## ☐ Копирование экрана на принтер: CALL 3756 (0EACh)

Процедура не требует предварительных установок и прямой ее вызов распечатает содержимое экрана на "ZX" принтере.

### ☐ Печать графики на принтере: CALL 3789 (0ECDh)

Эта процедура похожа на копирование экрана тем, что она использует буфер принтера и выводит его содержимое на бумагу. При этом используется процедура "RST 16", работающая с буфером, как с одной строкой экрана высотой 8 пиксель. Если вы разместите в одну линию буфера ваш рисунок и затем сделаете вызов по этому адресу, то буфер распечатается принтером. Примечание: буфер выводится построчно линиями пиксель (по 32 байт), а не побуквенно, как на экран. По завершении вывода буфер обнуляется.

### ☐ Очистка буфера принтера: CALL 3807 (EDFh)

Этот вызов обнуляет буфер принтера.

### ☐ Использование "BEEP" CALL 949 (3B5h)

Для этой программы необходимо поместить в: пару регистров "DE" - время звучания, "HL" - частоту.

Здесь: 0 - высокий звук, FFFFh - низкий.

Чем выше тон, тем короче звук, и эта линейная зависимость должна быть учтена. Необходимые значения рассчитываются следующим образом:

HL = 437500 / частота - 30.125 (частота в Гц)

DE = время * частота (время в сек.)

30.125 вычитается потому, что сама процедура 120.5 периодов генерирует ноту, потом очищает регистры и т.д. Нота «до» первой октавы примерно 261 Гц, следовательно, в "HL" должно быть число 1646 (DEC), а в "DE" для одной секунды 261 (DEC). Помните, прерывание осуществляется 50 раз в секунду и если вы запишите вашу программу в нижнюю часть 16K ОЗУ, звук будет промодулирован сигналами прерываний.

### ☐ Печать сообщений: CALL 3082 (C0Ah)

Для этого вызова пара "DE" должна содержать начальный адрес таблицы сообщений, в которой бит 7 должен быть в 1, а регистр "A" - содержать смещение начала сообщения в таблице, седьмой бит последнего байта сообщения должен быть активизирован. Первое сообщение имеет смещение 0. Если вы хотите напечатать сообщение "I AM" вы можете написать строку вроде: 'MESSAGE DEFB 80h : DEFM "I AM" : DEFB " "+80h'. Программа теперь прибавляет 80h, корректно выводя последний пробел и зная, что достигнут конец, произведет возврат к вызывающей программе. Таким образом целиком программа выглядит примерно так:

```
LD    A,0          ;для печати первого
CALL  PR_MES       ;списка сообщения
...
      ;здесь идет остальная часть программы
...
PR_MES LD DE,MESSAGE
CALL 3082
RET
MESSAGE DEFD 80
DEFM "ENTRY"
DEFB " "+80
DEFM "ENTRY 2"
DEFB " "+80h
```

Этим способом в текущем потоке, в текущей позиции будет напечатано сообщение "ENTRY", но, чтобы было напечатано "ENTRY2" регистр «А» должен содержать 1. Выведено будет только то, что между апострофов, сами апострофы не печатаются. Эта программа использует ПЗУ для расширения символов и генерации сообщения об ошибках при вызове из программы 2898 (B52h). Программа "DEBASE" в приложении G широко используют эту процедуру.

## ☐ Расширение символов для вывода: CALL 2898 (B52h)

Как только "SPECTRUM" находит код символа (нечто с активизированным битом 7), он должен решить, что с ним делать, т.к. им может быть или определенный пользователем рисунок, часть рисунка или слово из словаря БЕЙСИКа. Обычно за этим следят автоматически, когда используют RST 16 (10h) посредством этой процедуры. Если вы заменили адрес вывода в поток (например, выводите в интерфейс принтера), регистр «А» будет содержать код символа независимо от того, когда использовалась ваша процедура, и если вы хотите его расширить, вам придется сделать это самостоятельно. Но, если вы оставите его таким как есть, на принтере можно получить несколько необычные результаты. Эта программа может быть использована для расширения ключевых слов, но вам придется самому работать с графикой и делать так, чтобы графические коды не засылались в программу, иначе у вашего компьютера «крыша поедет». Примечание: при расширении процедура сама повторяет вызовы адреса, указанного используемым потоком, и осуществляет возврат после выдачи всех букв. Это означает, что вам предпочтительнее перейти к процедуре изменив код в регистре «А», тогда при последнем возврате код выведен не будет. Это показано в следующей типовой программе вывода на интерфейс принтера:

```
INIT LD HL,(23631);"CHANS" (адрес канала
                   ;данных)
LD   BC,15        ;установка потока 3 (принтер)
ADD  HL,BC        ;в HL ячейка, где находится
                  ;адрес, вызываемый
                  ;при выводе в этот поток
LD   BC,START     ;START = адрес вашей
LD   (HL),C       ;программы ввода/вывода
INC  HL
LD   (HL),B
RET  ;сейчас третий поток установлен на
     ;вывод вашей программы
START LD B,A      ;сохранить ASCII код в «B»
CP   165
JP   NC,B52h      ;программа расширения в ПЗУ
CP   13
JR   Z,CRLF       ;возврат каретки после
                  ;перевода строки
CP   32
RET  C            ;все, что меньше 32
                  ;не печатается
CP   128
JR   C,PRINT      ;должен быть обычный символ,
                  ;остальное здесь - графика,
     ;можно делать с ней что хотите
PRINT             ;сюда придет ваша программа
                  ;вывода/печати
RET               ;возврат за следующим
                  ;символом, если его нет,
     ;то возврат к вызывающей программе
```

Это стандартная программа вывода, но управляющие коды принтера ею выведены не будут. Чтобы преодолеть это, необходимо сначала однократно запросить первую часть программы, т.к. в ней предварительно определяется, куда распределен вывод потока 3. Единственная ситуация, когда необходимо инициализировать ее еще раз, это после исполнения команды "NEW" или изменения вывода другой частью программы.

## ☐ Расширение блока графики: CALL 2878 (B3Eh)

Если вы хотите применить блок графики из программы, то это может сделать процедура с точки 2878 (B3Eh). Для нее в пару "HL" нужно поместить базовый адрес резервных восьми байтов, в которых вы хотите изобразить сконструированный рисунок, а в регистр «B» - адрес блока графики. Затем делаются два вызова, второй непосредст-

венно после первого, т.к. каждый вызов создает лишь четыре байта изображения блока 8 * 8 пикселей. Здесь первый байт вашего 8-ми байтного блока будет вершиной, а "HL" укажет точку, следующую за последним байтом графики. "AF", "HL" и "BC" изменяются этой программой, никакие другие не используются.

## ☐ Рисование окружностей: CALL 9005 (232Dh)

Процедура рисования окружностей требует, чтобы параметры окружности были размещены на стеке калькулятора, поэтому предварительно необходимо загрузить стек. Программа, размещенная в 11560 (2D28h) выполнит эту задачу, если мы положим нужный нам код из регистра «A» на стек (подробности использования калькулятора в главе 8). После этого необходимо знать, что программа сбрасывает регистр "IY", идентифицируя "ERR NR" и портит содержимое большей части других регистров, поэтому нужно сохранять его значение перед использованием этой программы. На стеке калькулятора вами в указанном порядке устанавливаются следующие параметры: X, Y, Z, где X и Y - координаты центра, а Z - радиус. Программа рисования окружностей задает системные переменные COORDS и, если менять их не нужно, следует запомнить их перед рисованием окружности, разместив затем заново. Таким образом, программа рисования окружности выглядит примерно так:

```
LD    HL,(23677) ;COORDS
PUSH  HL         ;запоминаем координаты
LD    A,X        ;где X=0-255
CALL  2D28h
LD    A,Y        ;Y=0-175
CALL  2D28h
LD    A,Z        ;Z=РАДИУС (убедитесь, что на
      ;экране есть достаточное окно, в
      ;противном случае вы получите
      ;сообщение об ошибке
CALL  2D28h
CALL  232Dh      ; рисуем окружность
POP   HL
LD    (23677),HL ; заново размещаем COORDS
```

Если нужно нарисовать окружность вокруг текущих координат "COORDS", необходимо разместить их на стеке калькулятора, предварительно запомнив на машинном стеке, чтобы при желании иметь их в неизменном виде.

## ☐ Рисование линии: CALL 9146 (23BAh)

Процедура рисования линии лежит в ПЗУ с 9399 (24B7h). Как и программа рисования окружностей берет значения со стека калькулятора, однако в ней проще обойти место, где этот стек используется. Процедура стартует из "COORDS" и, если вам нужно начать в каком-то другом месте, то необходимо загрузить в "COORDS" эту стартовую точку, предварительно ее запомнив для использования в дальнейшем. В противном случае она укажет на конец вашей линии. На входе регистр "DE" содержит знаки параметров "DRAW", содержащихся в "BC": -1 (FFh) для отрицательных, +1 (01h) для положительных. Регистры "C" и "E" содержат "X", а "B" и "D" - "Y". Машинные коды, эквивалентные команде "DRAW 0,175" БЕЙСИКа выглядят примерно так:

```
LD    BC,AF00h    ;175,0
LD    DE,0101h    ;+
CALL  24BAh
```

или, для DRAW -255,0

```
LD    BC,00FFh    ;0,255
LD    DE,01FFh    ;+,
```

Отметьте, что эти команды рисуют из текущих "COORDS" и, если они не были изменены, то после окончания процедуры они укажут последнюю нарисованную точку и не учитывая этого не стоит рисовать что-нибудь вновь.

---

## ☐ Поиск адреса пикселя ("PIXEL"): CALL 8874 (22AAh)

Эта процедура может быть использована для нахождения адреса байта, содержащего пиксель, при этом регистры "BC" должны содержать координаты X и Y ("B" Y 0-175, "C" X 0-255). После возврата в паре "HL" будет содержаться адрес, а в "A" - позиция бита.

---

## ☐ Стирание части экрана: CALL 3652 (E44h)

Эта программа очистит строки нижней части всего экрана. Число их указано в регистре «B» (т.е. если «B» содержит 1, то сотрется только нижняя строка, если 10 - 10 нижних строк). Низ экрана - это всегда 24-я строка, а не последняя строка основного экрана.

---

## ☐ "SCROLL" части экрана: CALL 3584 (E00h)

Для того, чтобы произвести "SCROLL" части экрана, в регистре «B» необходимо предварительно разместить число, на единицу меньшее количества строк, для которых выполняется "SCROLL". Затем осуществляется вызов. Нижняя строка после каждого использования

программы будет стерта, а две последние строки подняты. Линии вновь считаются с нижней части экрана.

### ☐ Ввод в текущий поток: CALL 5606 (15E6h)

Эта программа берет адрес текущего потока из системной переменной "CURHL", ищет вход подпрограммы ввода в области информации о каналах и вызывает требуемую процедуру.

### ☐ Очистка стека калькулятора и рабочей области памяти: CALL 5823 (16BFh)

Эта программа может быть использована для освобождения стека калькулятора и определения его размеров. Она использует "HL" и по завершении размещает там содержимое "STKEND".

### ☐ "SAVE", "LOAD" и "VERIFY"

Эти программы очень просты и непритязательны, если в одной упряжке используются "SAVE" и "LOAD", известна точная длина данных и не волнует возврат в БЕЙСИК вследствие ошибки или нажатия "BREAK". Но если длина не известна и используется еще что-либо, то все усложняется. Обычно при загрузке "SPECTRUM" полагает, что заголовок, говорящий компьютеру как работать, будет получен перед основным блоком, и лишь затем последует сам блок. Но гораздо проще делать "SAVE" и "LOAD" без заголовка. Это возможно только тогда, когда точно известны все параметры загружаемого блока. Длина заголовка 19 байт (а не 17, как написано в большинстве книг), но только 17 должны быть активны, так как "SAVE" и "LOAD" первый и последний байты определяют сами.

Байт 1 - для заголовка всегда 00.

Последний - "PARITY BYTE", генерируется процедурой и он нас не волнует.

Байт 2 - содержит число, характеризующее запись;

если 0 - то это БЕЙСИК-программа,

1 - числовой массив,

2 - массив символов,

3 - блок кодов.

Байты 3--12 - имя.

Байты 13 и 14 - длина основного блока; (для БЕЙСИК-программы это соответственно переменные "ELINE - PROG").

Байты 15 и 16 - начальный адрес загрузки блока кодов или номер строки автономного старта для БЕЙСИК-программы.

Байт 16, для массива - имя в следующей форме:

биты 0-4 - имя (от A=1, до Z=26)

бит 5 - сброшен, если массив числовой;

бит 6 - активен, если массив строковый;

бит 7 - активен всегда.

Байты 17 и 18 - длина БЕЙСИК-программы (т.е. "VARS-PROG").

Байт 19 - активизируется в ходе работы программ "SAVE" и "LOAD".

**Схема заголовка:**

| BYTES | 1 | 2 | 3..12 | 13-14 | 15-16 | 17-18 | 19 |
|---|---|---|---|---|---|---|---|
| | флаг | тип | имя | длина | старт | БЕЙСИК-длина | паритет |
| IX+ | | 0 | 1..10 | 11-12 | 13-14 | 15-16 | 17 |

Для "SAVE" на ленте нужно либо создать заголовок (см. выше) и сразу же за ним набор данных, либо может быть записан только набор, если известны его параметры. В программу "SAVE" есть несколько входов, у каждого свои достоинства и недостатки.

1. Наиболее простой. Для него: в "IX" помещается точка старта заголовка (байт 2 см. выше), в "HL" - точка старта записываемого основного блока, после всего этого - CALL 2416 (0970h). Этим будет сохранен как заголовок, так и сам блок, но:

а) Загорится "START TAPE THEN PRESS ANY KEY", и все будет хорошо, если вы нажмете все, кроме "BREAK" на клавиатуре. Иначе будет возврат в БЕЙСИК.

б) Клавиша "BREAK" периодически опрашивается во время процедуры "SAVE" и происходит возврат в БЕЙСИК, если она нажата.

в) Заголовок сохраняется в форме, гарантирующей, что сохраненная информация будет загружена БЕЙСИКОМ. Это иногда более полезно, чем кажется на первый взгляд.

2. Эта точка входа аналогична предыдущей, за исключением того, что она не ждет и не опрашивает "BREAK". Она управляется через подпрограмму вызывающей процедуры, потому что нормальная ее работа зависит от правильной загрузки машинного стека. Как и выше, сначала устанавливаются "HL" и "IX", затем вызывается следующая подпрограмма:

```
SAVE PUSH HL
JP   2436(0984h)
```

Возвращение после вызова будет к точке, следующей после "SAVE". Это удобно для записи на ленту нескольких блоков подряд, без нажатия каждый раз на клавишу по сообщению "...THEN PRESS ANY KEY...".

Для того, чтобы исключить влияние "BREAK", должен быть сделан нормальный старт, но заголовок и данные записаны отдельно, как блоки. Программа "SA_BYTES" с точки 1218 (04C2h) делает любые записи. Для нее нужно в регистр «A» поместить: 00 - для заголовка, а FFh - для блока данных. Поначалу она загружает машинный стек программой "SAVE/LOAD RETURN", которая разрешает прерывания и проверяет "BREAK". По нажатии этой клавиши вызывается программа обработки ошибок из RST 8 и происходит возврат в БЕЙСИК. В противном случае возврат осуществляется к точке "RETURN", по адресу, положенному на стек вызывающей программой. По прохождении программы "SAVE/LOAD RETURN", на стеке останется адрес вашей вызывающей программы, а управление к ней будет передано при выходе из процедуры "SAVE". Если в процессе нее нажималась "BREAK", то флаг переноса сбрасывается. В противном случае флаг останется, но прерывания при этом будут отключены и для использования их необходимо восстанавливать.

Возможно вам придется организовывать какую-либо печать для того, чтобы подтверждать старт вывода, например путем использования детально изложенных в этой главе программ печати сообщений и ожидания ввода.

Чтобы записать этим методом блок кодов, необходимо, чтобы "IX" пара содержала точку старта, а "IY" - его длину. При этом в «A» положен 00 для заголовка, либо FFh для блока данных. Прямой вызов осуществляется по адресу 1222 (04C6h).

Если вы хотите сохранить стандартный заголовок, пара "DE" должна содержать 17 (11h). Так можно записать блок данных и без заголовка, но загрузить его обратно будет возможным лишь когда известна его длина.

Если не волнует возврат в БЕЙСИК, можно использовать коды с точки 1218 (04C2h), здесь, при нажатии "BREAK" вы вернетесь в БЕЙСИК и придется вновь включать запись.

## ☐ Выполнение "LOAD" и "VERIFY".

Данные с ленты могут быть загружены в память вашего компьютера в двух видах: с заголовком или без. Если есть заголовок, то он может быть использован для:

а) задания всех параметров блока данных, следующего за заголовком;

б) для задания тех деталей, которые не известны (как в БЕЙСИКе), или известны не до конца;

в) чтобы гарантировать правильность вводимых данных.

Там, где нет заголовка, все обычно размещающиеся в разделе заголовка детали должны быть определены до загрузки.

Из обычного заголовка можно сконструировать несколько типов разных заголовков следующим путем: как основной блок данных записывается ваш собственный заголовок, который устанавливает детали в настоящем основном блоке.

Часто это полезно тем, что предотвращается потеря блока при неизвестности его длины или местоположения на ленте. Это предотвращает введение блока кодов непосвященным пользователем, которому придется умудриться написать свою программу, проверяющую, что вы такое сделали, и лишь после этого ввести блок.

Процедура, которая дает такие специальные заголовки, дана в программе "DEBASE" приложения "G".

При загрузке блока данных с нормальным заголовком, сначала резервируются 34 байта в ОЗУ. В первых 17 байтах должен быть создан макетный заголовок. Он определяет параметры, требуемые для соответствия загружаемому последнему блоку. Все не прошедшее проверки на наличие и соответствие частей заголовка повторно тестируется уже относительно реального заголовка. Предполагая, что допустимы изменения, детали для загрузки будут взяты из макетного заголовка. Если изменения недопустимы, то командой RST 8 генерируется сообщение БЕЙСИКа об ошибке. Вторые 17 байт заполняются заголовком с ленты для сравнения, и при наличии соответствия эта область освобождается для использования.

Макетный заголовок делается также, как и записываемый на ленту. Как и раньше первый и последний из 19 байтов генерируются автоматически и не включены в 17-байтовую спецификацию. Первый байт должен быть аналогичен первому байту заголовка загружаемых данных (это типы данных, для успешной загрузки необходимо их соответствие). При входе в подпрограмму "LOAD BYTES" он со-

держится в "A". Эта загрузка должна выполнятся только прямым вызовом. При несоответствии типа, будет ожидаться начало следующего блока информации до тех пор, пока типы не станут равны: 0 - для БЕЙСИКа, 1 - для числового массива и т.д.

Следующие 10 байт - имя. Если оно безразлично, то первый байт из десяти должен быть FFh, при этом всегда считается, что соответствие есть.

Еще 2 байта - длина. Если это 0000, то длина будет взята из заголовка на ленте. В противном случае - эти байты здесь и на ленте должны быть равны.

Байты 1 и 16 содержат стартовые адреса для загрузки. Для БЕЙСИКа байт 15 игнорируется, а байт 16 задает спецификацию набора в той же форме, что и при выполнении команды "SAVE". Для БЕЙСИК-программы байт 1 - 0, а байт 16 - 80h. Последующая часть безразлична. Перед использованием контрольной части заголовка, в системной переменной "TADDR" в младшем байте должно быть 01 для команды "LOAD", или 2 для выполнения "VERIFY".

В паре "HL" должен хранится адрес, по которому разместится основной блок, или 0, если будет использоваться информация заголовка с ленты. Для существующего БЕЙСИК набора здесь начало данных, идущих вслед за байтами имени и длины в области переменных БЕЙСИК-программы.

"IX" адресует первый байт макетного заголовка, и, наконец, вызов программы ПЗУ с точки 1889 (761h), выполнит команды "LOAD" и "VERIFY" заголовка и данных с ленты.

Выполнение команд "LOAD" и "VERIFY" без заголовка возможно, если все основные параметры блока известны. Делается это очень просто. Сначала в «A» кладется FFh. Это сигнализирует о том, что будет вводится собственно программа. Затем в "DE" полная длина блока, после чего в "IX" помещается адрес начала "LOAD" или "VERIFY". Это сопровождается установкой флага переноса, если выполняется "LOAD", или его сбросом при "VERIFY". И наконец, CALL 1366 (556h).

В начале работы этой процедуры как и при "SAVE" загружается машинный стек и происходит возврат по ошибке, при нажатии "BREAK". "BREAK" можно исключить вызовом следующей короткой подпрограммы:

```
LOAD INC D      ;сброс флага в 0
EX   AF,A,'F'
DEC  D
```

```
DI                      ;отключаются прерывания
LD    A,0Fh
OUT   (FEh),A           ;установка "BORDER" и
                        ;порта "EAR"
JP    562h              ;перейти к главной
                        ;LOAD программе
```

Команда "OUT" устанавливает "BORDER" в белый цвет с помощью трех младших битов по адресу (FEh). Можно установить и другой цвет, но пятый бит остается 1, т.к. он определяет порт "EAR", сигнализируя, что с ленты идет загрузка.

Ошибка загрузки диагностируется по сбросу флага переноса при возвращении к вызывающей программе. Так как попытка сделать "BREAK" приводит к возврату, то он может произойти, если процедура проверки "BREAK" вызывается после проверки ошибок. Примечание: прерывания после возврата из загрузки будут отключены. Эта процедура может быть также использована для ввода заголовка вместо основного блока данных, для этого перед вызовом в «A» помещается 0. Это полезно при создании вами оригинальных заголовков, читаемых только вашей программой (см. "DEBASE" приложения G).

# ГЛАВА 3

# 8К ПЗУ ИНТЕРФЕЙСА

С добавлением в "SPECTRUM" 8К ПЗУ микродрайверного интерфейса, открылась возможность расширения форм использования ПЗУ (как в «BBC» микрокомпьютерах) и простой путь для расширения БЕЙСИКа СИНКЛЕРА. Для этого в первую очередь необходимо понять, как "Z80" может адресовать память более 64К.

Процессор имеет два множества информационных линий. Это A1-16 (которыми он указывает памяти, какой байт он использует (адресная шина)), и D0-7, которые он использует для считывания и записи в память позиций, указанных на этой адресной шине.

Обычно невозможно выйти за пределы 64К объема, поэтому для большего адресного пространства производится некая операция, формирующая чередование двух банков памяти в одном адресном пространстве. Это легко сделать выбрав байт памяти, сравнив его адрес с заранее определенным значением на адресной шине, и при достижении соответствия переключиться на альтернативную память. Выполнение затем будет продолжено со следующего адреса, т.к. программный счетчик инкрементируется, но данные будут уже из нового адресного пространства. В СПЕКТРУМе это сделано путем слежения за адресной шиной программного счетчика при адресации ячейки 8. Это программа обработки ошибок. Так как этот адрес достигается только RST 8, вершина стека всегда содержит адрес воз-

врата (т.е. адрес команды, следующей за командой вызова). Вершина стека достигается программой, хранящейся в теневой ПЗУ, и ей проверяется содержимое этой ячейки, называемое далее «обходной код». При обращении к RST 8 его значения в интервале от 00 до 1Ah ведут к возврату в 16K ПЗУ, т.к. это обычные коды ошибок. Но значения между 1Bh и 32h используются в качестве ловушки. После этого вызывается следящая процедура, что сейчас будет детализировано. Для использования этих обходных кодов применяется RST 8 непосредственно после которой находится "DEFB" обходной код. Вся адресация после этого относится уже к теневой ПЗУ.

## ВВОД

### ☐ Ожидание ввода ключа: обходной код 1Bh адрес 6616 (19D9h)

Это похоже на команду ввода некоторых разновидностей БЕЙСИКа, но БЕЙСИК СИНКЛЕРа ее не содержит. Процедура ожидает нажатия и затем помещает код нажатой клавиши в «A». Маскируемые прерывания должны быть включены, т.к. 16K ПЗУ используется для сканирования клавиатуры.

### ☐ Ввод RS232: обходной код 1Dh адрес 2945 (B81h)

Перед использованием этой программы необходимо предварительно установить коэффициент "BAUD", используя системную переменную "BAUD 23747/8 (5CC3/4h)", рассчитываемую как 3 500 000/(26 * коэффициент "BAUD") - 2, где 3 500 000 - тактовая частота СПЕКТРУМа. После этого можно вызывать процедуру ввода, установив "SET_FL 23751 (5CC7h)" в 0. По окончании код полученного символа будет в «A», а флаг переноса активизирован. Для введения кода программа ждет лишь некоторое время. Если ожидание затягивается или нажат пробел, произойдет возврат, но флаг переноса установлен не будет.

### ☐ Ввод сети: обходной код 2Fh адрес 6705 (1A31h)

Перед использованием необходимо открыть канал сети, сделав его текущим с помощью программы, описанной ниже (см. раздел «Вывод сети»). Эта программа будет читать набор сигналов сети. Перед вызовом в "IX" необходимо поместить точку входа области сети и при этом "IX"+11, "IX"+12, "IX"+13, "IX"+14 нужно корректно установить для вводимого блока (детали см. в «ЗАГОЛОВОК СЕТИ», раздела «ВЫВОД СЕТИ», обходной код 30h). "IX"+13 и "IX+14 - номер блока, инкрементируемый после каждого успешного ввода блока.

Флаг переноса на первый взгляд помогает распознать, был ли введен набор сигналов, или произошла ошибка с возвратом в вызывающую программу. Но флаг переноса может испортится и по другим причинам, например сбросом цвета бордюра при выходе из программы.

Возврат из этой процедуры будет сделан после успешного ввода, флаг переноса сброшен, но, по истечении допустимого времени ожидания ввода набора сигналов, выдается ошибка контрольной суммы. В другом случае, если нажат "BREAK", активизируется флаг переноса.

По некоторым причинам проще использовать процедуру 5606 (15E6h) из 16K ПЗУ. Перед ее вызовом "IX" нужно сохранять, если регистр «A» не получил вводимого кода.

ВЫВОД

---

### ☐ Печать на экран: обходной код 1Ch, адрес 6636 (19ECh)

Перед вызовом в регистр «A» помещается код символа. Эта подпрограмма - точная копия программы 16K ПЗУ, устанавливающей вывод на поток 2 (основной экран) и RST 16, печатающей текущий поток, также используемая программой 16K ПЗУ. Если применить это в сочетании с «Ожидание ввода ключа» (см. выше) получится т.н. автономный терминал. Программа для этого выглядит примерно так:

```
RST   8
DEFB  1Bh
RST   8
DEFB  1Ch
```

RST 8 в первом случае ждет нажатия ключа и делает возврат с его кодом в регистре «A», вторая RST 8 как эхо выводит эту строку на основной экран (или в другое место, определяемое потоком 2).

---

### ☐ Печать на принтер: обходной код 1Fh, адрес 6652 (19FCh)

Совершенно такая же программа как и для печати на экран, но здесь используется поток 3 (обычно это принтер), а не поток 2.

---

### ☐ Вывод RS232: обходной код 1Eh, адрес 3162 (C5Ah)

Вновь в регистр «A» помещается выводимый код, но используются RS232 порты вывода информации. Коэффициент "BAUD" берется из системной переменной "BAUD", цвет бордюра из системной перемен-

ной "IOBAUD" (как их установить см. в главе 4). Никакая информация не пересылается до активизации линии "DTR" ("DTR" - Data Terminal Ready).

Основная точка входа, вызываемая этим обходным кодом, допускает вывод любого значения регистра «A». При этом необходима осторожность, дабы не переслать контрольные коды. Другая полезная точка 3132 (C3Ch), она ищет непечатаемые "ASCII" коды и реагирует на них. Любые коды ниже 32 (20h) повлекут за собой немедленный возврат. Но код 13 (0Dh) (перевод каретки) будет выведен, он сопровождается кодом заполнения строки 10 (0Ah). Коды начиная с 128 (80h) интерпретируются исходя из их типа. Здесь графика выводится в виде "?", а символы будут расширены путем обращения к программе по адресу 3088 (C10h) 16K ПЗУ, требующей, чтобы перед вызовом из кода было вычтено 165 (A5h).

## ВЫВОД СЕТИ

### ☐ Открытие канала: обходной код 2Dh, адрес 3753 (EA9h)

Перед получением или пересылкой данных посредством сети нужно открыть канал сети. Это делается данным вызовом после инициализации системных переменных "D_STR 23766 (5CD6h)" и "NSTAT 23749 (5CC5h)". Ими задаются номера пункта назначения и пункта, из которого производится пересылка, соответственно. Канал сети будет создан в конце области "CHANS". Вся область от "PROG 23635/6 (5C53/4h)" и до "STKEND 23653/4 (5C65/6h)" переместится вверх на 276 байт. Значения всех соответствующих переменных будут сброшены, полагая, что в ОЗУ, ниже "RAMTOP" есть окно, и если места в ОЗУ мало, то это повлечет ошибку.

По завершении работы этой подпрограммы регистр "IX" будет указывать стартовый адрес нового канала. Новый созданный канал будет временным, что определяется установкой в 1 седьмого бита "IX"+4. Чтобы сделать канал постоянным, необходимо сбросить этот бит. Теперь канал можно использовать для пересылки загружаемого в "CURCHL 23633/4 (5C51/2h)" байта в адрес из регистра "IX". Для вывода данных побайтно используется RST 16 (10h) 16K ПЗУ.

По завершении пересылки данных через RST 16 канал должен быть закрыт с помощью обходного кода 2Eh (см. ниже «Закрытие каналов сети»), который перешлет все оставшееся из буферов, таким образом область буферов будет очищена. Но это не закроет остальные потоки, соединенные с каналами сети, и необходимо внимание, чтобы другие каналы и потоки не были испорчены.

### ☐ Пересылка пакета: обходной код 30h, адрес 3530

**(DB2h)**

Этот вызов позволяет переслать пакет по сети. Перед использованием канал сети должен быть открыт, заголовок и системные переменные заданы, а данные для пересылки помещены в буфер. Для открывания канала сети используется обходной код 2Dh (детали см. выше).

Чтобы при вызове этой подпрограммы удостоверится, что сеть свободна, посылается некий «разведчик». После этого следует заголовок и уже потом блок данных. Заголовок содержится перед каналом сети и адресуется через регистры "IX", указывающие адрес первой ячейки канала. Байты от "IX+00" до "IX+10" устанавливаются программой обработки каналов, а байты от "IX+11" до "IX+18" являются заголовком. Схема заголовка:

IX+11 пункт назначения

IX+12 пункт, из которого производится пересылка

IX+13 и 14 номер блока

IX+15 1 для последнего блока, иначе 0

IX+16 длина буфера (0-255)

IX+17 контрольная сумма для буфера данных

IX+18 контрольная сумма для заголовка

IX+19 и 20 используется только при получении

IX+19 позиция последнего взятого из буфера кода

IX+20 количество доступных байтов в буфере

IX+21 до IX+275 данные для пересылки, вплоть до 255 байт.

Контрольные суммы задавать не нужно, они устанавливаются самой программой, "IX+15" загружается содержимым регистра «A» при входе в подпрограмму. Все остальное определяется пользователем. В дополнение к созданию заголовка системные переменные "DSTR1 23766 (5CD6h)" и "NSTAT 23749 (5CC5h)" должны содержать номера пунктов назначения и отправления соответственно. При каждом обращении к этой процедуре номер блока инкрементируется. При выходе из программы открывания каналов базовый адрес канала сети возвращается в регистре "IX".

## ☐ Закрытие каналов сети: обходной код 2Eh, адрес 6692 (1A24h)

Если эта подпрограмма вызывается после процедуры пересылки, она перешлет все оставшиеся в буфере сети данные, отмечая их как конец блока, но после получения подтверждения очистит 270 байтов

области буфера, используя коды 16K ПЗУ по адресу 6632 (19E8h), и канал будет закрыт.

## ВЫВОД МИКРОДРАЙВА

☐ **Открывание канала/открывание файла:**

обходной код 22h, адрес 6953 (1B28h)

Перед пересылкой данных на МИКРОДРАЙВ задается канал МИКРОДРАЙВа и область памяти для его описания. Обходной код 28h был задуман для решения этой задачи, но в программе есть ошибка. Этот код может быть использован для выполнения команды, если сначала в "DSTR1 23766/7 (5CD6/7h)" установить номер драйва (1-8), в "NSTR1 23770/1 (5CD4/h)" длину имени файла, а в "TSTR1 23772/3 (5CD6/7h)" адрес начала имени файла в памяти. Пару регистров H'L' нужно сохранить и после этого использовать обходной код.

Канал открывается для ввода, если есть имя файла. В противном случае он откроется как канал вывода. Чтобы сделать этот канал постоянным необходимо ввести нужные данные в информацию о потоке. Это достигается с помощью следующей программы:

```
EXX   ;сохраним адрес возврата в БЕЙСИК
PUSH HL
EXX
LD    A,S          ;S-номер потока для канала
RLA                ;продублируем его
LD    HL,5C16h     ;базовый адрес для потоков
LD    D,0          ;ответвим для "DE" поток S
LD    E,A
ADD   HL,DE        ;теперь "HL"=адрес потока
PUSH  HL           ;сохраним его
RST   8            ;вызов обходного кода
DEFB  22h          ;22(открывание канала/файла)
PUSH  HL           ;по возвращении из этой
      ;программы   "HL"=РАЗМЕЩЕНИЕ потока
XOR   A            ;программа открывания канала
RST   8            ;не прекращает работу
DEFB  21h          ;МИКРОДРАЙВа, поэтому он
                   ;должен быть отключен
POP   DE           ;"DE"=РАЗМЕЩЕНИЕ потока
POP   HL           ;"HL"=АДРЕС потока
LD    (HL),E       ;установим поток "S"
                   ;в корректное значение
INC   HL
LD    HL),D
RES   7,(IX+4)     ; по возвращении из обход-
```

```
        ;ного кода 22, "IX"=НАЧАЛО области ка-
        ;нала. Сброс ("IX"+4) делает этот
        ;канал постоянным
EXX     ; восстановим адрес возврата в БЕЙСИК
POP     HL
EXX
RET     ;конец программы
```

Для считывания/записи с МИКРОДРАЙВа задаваемый канал должен быть сделан текущим путем загрузки в системную переменную "CURHL 23633/4 (5C51/2h)" базового адреса, помещаемого в "IX" после открывания канала. Затем набор может быть записан с помощью 16K ПЗУ RST 16 или считан 5606 (15E6h), передавая каждый символ через регистр «A». Если потребуется исполнить команды "SAVE", "VERIFY" или "LOAD" с помощью МИКРОДРАЙВа, вместо использования набора как файла, см. далее в этой главе.

### ☐ Вывод записи: обходной код 26h, адрес 4607 (11FFh)

Этот обходной код запишет содержимое буфера МИКРОДРАЙВа в очередной свободный сектор на микродрайверном наборе. Перед вызовом буфер должен содержать информацию, которая должна быть сохранена, а регистр "IX" - указывать начало канала МИКРОДРАЙВа, "IX"+11 - содержать длину данных, "IX"+13 - номер сектора (0 для первого сектора записи). Он автоматически инкрементируется каждый раз, когда запись пересылается или получается. От "IX"+14 до "IX"+23 располагается имя записи, "IX"+25 - используемый драйв. Ячейки от "IX"+82 до "IX"+593 используются для пересылаемых данных.

### ☐ Вывод сектора: обходной код 2Ah, адрес 6801 (1A91h)

Эта команда отличается от предыдущей тем, что ищет сектор записи с номером из "IX"+13, и если такой существует, то переписывает его в буфер. Если же такого сектора не существует, то выдается ошибка "FILE NOT FOUND". Содержимое сектора не проверяется, поэтому нужно быть очень осторожным и точно знать, что в секторе нет ничего полезного.

### ☐ Закрытие канала МИКРОДРАЙВа: обходной код 23h, адрес 4777 (12A9h)

Этот обходной код такой же, как и «Закрытие канала сети», но для МИКРОДРАЙВа регистр "IX" на входе должен содержать базо-

вый адрес файла. Если канал был использован для пересылки, то вместо оставшейся в буфере информации будет послано сообщение "END OF FILE" (конец файла) и буфер сброшен, или просто сброшен, если это был канал считывания.

---

### ☐ Уничтожение файла: обходной код 24h, адрес 7534 (1D6Eh)

Этой командой вычеркивается файл МИКРОДРАЙВа, имя которого записано в памяти и указано в "T_STR1", длина имени в "N_STR1", а адрес МИКРОДРАЙВа в "D_STR1".

Ввод МИКРОДРАЙВа.

Возврат из всех последующих программ "READ" делается с работающим МИКРОДРАЙВом и отключенными маскируемыми прерываниями.

---

### ☐ Считывание печатаемой записи: обходной код 27h, адрес 6679 (1A17h)

Номер записи должен содержаться в "IX"+13, "IX" указывает на начало микродрайверного канала. "IX"+25 - номер драйва, от "IX"+14 до "IX"+23 - имя записи. Если сектор записи существует, то он считывается в буфер, иначе - ошибка.

---

### ☐ Считывание следующей печатаемой записи: обходной код 25h, адрес 6665 (1A09h)

Команда аналогична предыдущей, но используется после нее. Она автоматически считывает запись с файла печати, если запись существует. В противном случае выдается сообщение об ошибке.

---

### ☐ Считывание сектора записи: обходной код 28h, адрес 6731 (1A4Bh)

Вызов считывает запись из файла. В регистре "IX" стартовая точка канала МИКРОДРАЙВа, "IX"+13 должен быть активизирован, номер записи внутри файла "IX"+25 должен содержать номер драйва и имя файла - от "IX"+14 до "IX"+23. Если файл существует, будет сделано считывание в буфер. Если это файл печати, возврат произойдет со сброшенным флагом переноса, а запись сделана в область буфера МИКРОДРАЙВа. Если это не файл печати, флаг переноса будет сброшен, а запись стерта из буфера.

---

### ☐ Считывание следующего сектора записи: обходной код 29h, адрес 6790 (1A86h)

То же, что и выше, но для следующего сектора файла. Это также похоже на обходной код 25h.

---

### ☐ Включение/выключение работы МИКРОДРАЙВа: обходной код 21h, адрес 6135 (17F7h)

Регистр «А» должен содержать либо 0, тогда отключается работа всех работающих драйверов, либо номер драйва, работа которого требуется. Если драйв не существует, то это приведет к ошибке. Если драйв существует, то начнется его работа. Возврат при этом делается с отключенными прерываниями.

---

### ☐ Сброс канала МИКРОДРАЙВа: обходной код 2Ch, адрес 4292 (10C4h)

Область канала МИКРОДРАЙВа, указанная в регистрах "IX" будет сброшена. Все текущие потоки, использующие этот канал, закроются, сбросится также и область схемы МИКРОДРАЙВа, если она не используется другими каналами. Вся память от сброшенного канала до "RAMTOP" передвигается вниз на 627 байт, освободивших-ся при сбросе.

---

### ☐ Сканирование клавиатуры: обходной код 20h, адрес 6657 (1A01h)

При возврате активизируется флаг переноса после любого ключа.

---

### ☐ Добавление переменных: обходной код 31h, адрес 6568 (19A8h)

Этим вызовом устанавливаются внешние системные переменные, необходимые для интерфейса 1. На самом деле это просто команда возврата, т.к. переменные создаются автоматически при появлении первой ошибки, выданной программой 16К ПЗУ.

---

### ☐ ПЗУ 2: обходной код 32h, адрес 6564 (19A4h)

Возможно это самая полезная программа, т.к. позволяет подключать 8К ПЗУ по желанию и использовать любые программы из нее. Адрес программы, которая вам необходима, должен быть помещен в системную переменную "HD_11", после этого применяется обходной код. К сожалению единственный регистр, который при вызове может быть непосредственно использован - регистр «А», но это не беда. При

---

использовании обходного кода адрес из "HD_11" 23789/90 (5CED/Eh) помещается на машинный стек, за ним помещается адрес возврата для переключателя 1792 (700h) 8K ПЗУ на 16K. Если вы делаете адрес в "HD_11" указывающим обратно на вашу собственную программу, то затем можно сделать "POP" с двумя адресами возврата на стеке, при этом 8K ПЗУ будет подключено, а управление сохранится в вашей программе. Теперь свободно могут быть использованы любые регистры для работы программ 8K ПЗУ, а 8K ПЗУ отключится. Примечание: многие программы следят за флагом синтаксис/выполнение в "FLAGS 23611 (5C3Bh) IY + 1" и производят возврат к интерпретатору БЕЙСИКа, если установлен синтаксис. Некоторые программы МИКРОДРАЙВа также производят возврат в БЕЙСИК через "ERR SP 23613/4 (5C3D/Eh)", если завершение было нормальным, иначе через ошибку 8K ПЗУ. Ошибка 16K теперь используется для указания вызывающей программы путем замены адреса в переменной "ERR SP" на адрес возврата вашей программы на машинном стеке (см. программу "DEBASE" в приложении 6).

После загрузки адреса в "HD_11" обходным кодом "RST 8 DEFB 32h" вызываются следующие программы.

---

### ☐ Каталог набора: обходной код 32h, адрес 7256 (1C58h)

Перед использованием сохраните регистры "H'L'". Переменная "D_STR1" должна содержать номер МИКРОДРАЙВа, а "S_STR1" - номер потока, для которого печатается каталог. Затем делается вызов. По окончании вывода каталога восстановите содержимое "H'L'".

---

### ☐ Формат набора: обходной код 32h, адрес 7022 (1B6Eh)

Вновь нужно сохранить регистры "H'L'". "N_STR1" должна содержать длину имени, которое будет дано набору, "T_STR1" - адрес первого символа имени при вызове программы. Регистры "H'L'" должны быть заново размещены после успешного возврата в БЕЙСИК.

---

### ☐ Выполнение (RUN): обходной код 32h, адрес 2709 (A95h)

Это самая простая из программ. Она загружает программу "RUN" с МИКРОДРАЙВа 1. Все, что нужно сделать - это перейти к программе в ПЗУ. Это делается как описано выше после отключения 16K ПЗУ.

Следующие программы для выполнения команд "SAVE", "LOAD" и "VERIFY", "MERGE" требуют задания заголовка перед их использованием. Подробно задание канала МИКРОДРАЙВа см. в справоч-

никах МИКРОДРАЙВа и интерфейса 1. Как делать канал МИКРОДРАЙВа, сейчас объяснено не будет (см. ниже). Я покажу, как формируется заголовок, чтобы сделать возможным использование МИКРОДРАЙВ из программы в кодах.

## ☐ Формирование заголовка МИКРОДРАЙВа.

Заголовок МИКРОДРАЙВа делается подобно тому, как описано в главе 2, но не нужно создавать заголовок в канале МИКРОДРАЙВа, т.к. для этого есть процедура ПЗУ. Теперь достаточно лишь правильно изменить системные переменные. Процедура, которая делает это внутри вашей программы выглядит так:

```
LD    HL,23782    ;HD_00
LD    (HL),T      ;T=0 для программы,
      ;1 для числового массива, 2 - массив
      ;строк, 3 - коды
INC   HL
LD    DE,LEN ; длина основного блока данных
LD    (HL),E
INC   HL
LD    (HL),D
INC   HL
LD    DE,START    ;начало данных, которые
будут сохранены или запрошены, если выполняется "LOAD". 0, если информация набора будет использоваться.
LD    (HL),E
INC   HL
LD    (HL),D
```

Все последующее нужно выполнять для программ на БЕЙСИКе или массивов, в противном случае переходите к "GO_ON"

```
INC   HL
LD    DE,DATA     ;длина программы или имя
      ;переменной как описано в главе 2 для
      ;"SAVE" или "LOAD"
LD    (HL),E ;
INC   HL ;
LD    (HL),D
INC   HL
LD    DE,AUTO     ;номер строки автономного
                  ;старта
LD    (HL),E      ;если нет, то число больше
INC   HL          ;10000
LD    (HL),D
```

```
GO_ON LD HL,23769 ;делаем это командой
                  ;МИКРОДРАЙВа
LD    (HL),"M"    ;это должно быть заглавной
      ;буквой SET BIT,(IY+124)
      ;бит 4 - "LOAD", 5 - "SAVE",
      ;6 - "MERGE", 7 - "VERIFY"
LD    HL,NLEN     ;длина имени файла
                  ;(до 10 букв или символов)
LD    (23770),HL  ;это "NSTR1"
LD    HL,NAME     ;адрес начала имени файла
LD    (23772),HL  ;T_STR1
LD    HL,DRIVE    ;номер драйва (1-8)
LD    (23766),HL  ;D_STR1
BIT   5,(IY+124)  ; это команда "SAVE" ?
JP    NZ,1E7Fh    ;если да, то идем к "SAVE"
                  ;в ПЗУ
JP    0BAF        ;иначе программы
                  ;LOAD/VERIFY/MERGE
NAME  DEFS 10     ;10 свободных байтов для
                  ;имени файла
```

Вышеприведенная программа должна быть вызвана из головной программы. Возврат по окончании работы программы ПЗУ к вашей программе осуществляется к адресу на машинном стеке, указанному системной переменной "ERR SP". Подробнее смотрите программу DEBASE, приложение G. Программы "SAVE" или "LOAD" подключают 16K ПЗУ.

Любая неточность в задании заголовка приведет либо к ошибке в БЕЙСИКе, либо к зависанию машины, поэтому требуется особое внимание. Как и в большинстве 8K ПЗУ программ необходимо сохранять регистры "H'L'", а потом их восстанавливать.

При написании программ в машинных кодах, использующих программы ПЗУ, очень важно помнить, когда и какое ПЗУ подключено. Меняются не только программы, но и RST, которые различны у 8K и 16K ПЗУ.

Возможен свободный обмен между ПЗУ с помощью обходного кода 32h из 16K ПЗУ и RST 10h из 8K ПЗУ. Использование кода 32h уже показано, а детальное описание RST 10h и других RST 8K ПЗУ дано ниже.

```
RST   0
POP   HL            ;перемещает адрес возврата
                    ;с машинного стека
LD    (IY+124),00   ;это "FLAGS 3"
JP    700h          ;700h - возврат к 16K ПЗУ
```

Адрес на вершине стека на этом шаге будет адресом возврата к программе, вызывающей 8K ПЗУ.

```
RST   8
LD    HL,(CH_ADD); не работает, если
                  ;вызывается из 16K
POP   HL          ;адрес кода ошибки
PUSH  HL          ;снова его сохраним
JP    009Ah ;проверяет, что выполняется и
      ;либо делает возврат к 16K ошибке,
      ;используя обходной код,
      ;либо работает канал
```

Это не нужно, если подключено 8K ПЗУ. RST 16 в 8K ПЗУ вызывает программу 16K ПЗУ, адрес которой содержится в "DEFW" после команды рестарта. По выполнении программы 16K управление передается по адресу, указанному после "DEFW" в вызывающей программе.

```
RST   16
LD    (5CBAh),HL ;запомним "HL" для восстанов-
                 ;ления после возврата
(10h) POP HL ;поместим в "HL" адрес "DEFW"
PUSH  DE          ;запишем "DE" на машинный
                  ;стек
0081h LD E,(HL)   ;поместим вызываемый адрес
                  ;16K ПЗУ в "DE"
INC   HL
LD    D,(HL)
INC   HL          ;теперь "HL" - адрес воз-
                  ;врата после "DEFW"
EX    (SP),HL     ;зададим новый адрес
                  ;возврата
EX    DE,HL
LD    HL,0        ;это для процедуры RST 8
PUSH  HL          ;возврат из 8K к вызову 16K
LD    HL,8        ;рестарт по ошибке
PUSH  HL          ;на стек для последующего
                  ;возврата
LD    HL,5CB9h    ;начало подпрограммы
PUSH  HL          ;на стек для возврата
JP    0700h       ;команда возврата,
                  ;подключающая 16K ПЗУ
0700h RET         ; возврат в 16K ПЗУ
```

Я дал полный листинг, т.к. это интересный метод переключения управления. После возврата по адресу 0700h подключена 16K ПЗУ

и 5CB9h - адрес со стека. Он содержит 21h, т.е. команду "LD HL,NN". Как мы видим, NN - величина, которая была в "HL" при рестарте. Следующий адрес 5CBCh содержит CDh, т.е. "CALL NN", опять-таки этот адрес был загружен раньше из "DEFW" после команды RST 16 (10h). Возврат же из 16K ПЗУ теперь к адресу 5CBFh, содержащему 22h, т.е. команду "LD (NN),HL", где NN - 5CBAh, таким образом в регистре "HL" сохраняется новое значение, команда "RET" - возврат к адресу на вершине машинного стека там теперь 8 (рестарт по ошибке). Если адрес для "DEFB" был снят со стека, то это будет 0, т.о. для 8K ПЗУ будет сказано, загрузить "HL" из 5CBAh. После этого еще один возврат, на этот раз по адресу после двух "DEFB", хотя RST 16 выполнялась так давно.

Следующие рестарты могут быть использованы для генерации сообщений об ошибках теневой ПЗУ, так же, как и RST 8 у основной.

RST 24 BIT 7,(IY+1) ; это "FLAGS"

(18h) RET ; флаг нуль, если на синтаксис проверяется строка на БЕЙСИКе

RST 32 RST 24 ; проверка: синтаксис или выполнение программы

(20h) JR Z,0068h ; сбросит указатель стека на адрес в "ERR_SP", загрузит "X_PRT" в "CH_ADD" и сделает возврат к подпрограмме ввода строк из основной ПЗУ через программу очистки калькулятора и укажет место ошибки.

JR 003Ah ; генерирует сообщение об ошибке, код ошибки теневой ПЗУ будет взят из байта после рестарта.

Этот рестарт может быть использован для генерации ошибок 16K ПЗУ загрузкой в "ERR_NR" кода ошибки перед вызовом.

(28h) JR 0040h ; TV флаг проверка, произошла ошибка во время выполнения, или при проверке на синтаксис. Если при выполнении, то возврат будет сделан к программе ошибок 16K, иначе - переход к 0068h, как показано выше.

RST 48 JP 01F7h ; проверяем, есть ли переменные (30h) интерфейса, если нет, то вводим их.

RST 56 EI ; это обычное маскируемое (38h) прерывание, которое сканирует клавиатуру, если подключено 16K ПЗУ.

RET ; отметьте, этим клавиатура не сканируется.

Программа немаскируемых прерываний в 102 (66h) состоит из одной команды возврата "RETN".

Другие программы 8K ПЗУ обсуждаются в главе 7, но из них мало используемых для других целей. Их использование и адреса были показаны в разделе обходных кодов.

# ГЛАВА 4
# СИСТЕМНЫЕ ПЕРЕМЕННЫЕ

Интерпретатор БЕЙСИКа должен запоминать то, что он собирается делать. Он находится в ПЗУ, но должен использовать ОЗУ для размещения информации. Т.к. ПЗУ неизменно, адреса системных переменных предварительно должны быть определены так, чтобы быть доступными в одном и том же месте. В СПЕКТРУМе это достигается как фиксацией их адресов в определенной области памяти, так и индексацией их относительно регистра "IY", всегда показывающего адрес 5C3Ah (ERRNR).

## СИСТЕМНЫЕ ПЕРЕМЕННЫЕ 16K

### ☐ KSTATE

### ☐ адреса 23552 - 23559 (5C00h - 5C07h)

Эти адреса используются при считывании с клавиатуры. Их можно разбить на два множества из четырех, одинаковых в использовании. То, какое множество используется в данный момент зависит от состояния другого.

В первой позиции каждого множества располагается значение "CAPS SHIFT" нажатого ключа или 255 (FFh), если ключ не был нажат, сигнализируя о том, что множество свободно для использования. Второй байт - счетчик, уменьшающий значение на 1 при каждом освобождении множества. Он первоначально установлен в 5. Но если множество свободно, то в 0. Задержка для повтора при длительном нажатии здесь в третьем байте, первоначально загруженном из "REPPER". ASCII код нажатого ключа находится в последней позиции множества до тех пор, пока второй байт не придет в 0 (множество свободно). Чтобы разъяснить это, я покажу программу, использующую эти переменные. Если ключ нажат, то сканирующая подпрограмма смотрит, свободно ли первое множество, и если да, то используется оно, иначе проверяется второе. В свободном множестве "CAPS SHIFT" код нажатого ключа помещается в первую ячейку, вторая ячейка устанавливается в 5, а в третью загружают задержку для повтора "REPDEL". Код нажатого ключа декодируется в ASCII, помещается в последнюю ячейку и копируется в "LAST K". Бит 5 "FLAGS" активизируется, сигнализируя, что был нажат новый ключ и происходит возврат к вызывающей программе. Если оба множества заняты, число во втором байте каждого множества декрементируется и производится возврат к вызывающей программе.

Как только один из счетчиков достигает 0, код нажатого ключа запоминается и производится сравнение кодов ключей двух множеств. Их совпадение означает, что ключ все еще нажат, и программа разрешает автоповтор. Не путайте эту программу с процедурой разделения ключей. Задержка для повтора содержится в третьем байте и первоначальна загружается из "REPDEL". По истечении времени до первого повтора (до повтора программа вызывается столько раз, сколько позволяет счетчик задержки см. выше) и не нарушенном равенстве кодов в множествах, третий байт загружается из системной переменной "REPPER". Код вновь помещается в системную переменную "LAST K", бит 5 "FLAGS" активизируется и происходит возврат к вызывающей программе. При последующих вызовах программы сканирования клавиатуры процесс повторяется, но с более короткой задержкой из "REPPER", т.к. коды больше не совпадают. Если нажали другую клавишу, то выполнение произойдет, как если бы множество было свободно.

Вкратце результат всего этого - наличие двух пунктов хранения и обработки ключей, допускающих нажатие второго ключа до того, как истекает задержка повтора первого множества, что исключает потерю второго ключа, если он нажат недостаточно долго для освобождения исходного множества.

---

### ☐ LAST K: адрес 23560 IY - 50 (5C08h)

Содержит код последней нажатой клавиши.

---

### ☐ REPDEL: адрес 23561 IY - 49 (5C09h)

В этой ячейке содержится количество вызовов программы сканирования клавиатуры до повтора при длительном нажатий.

---

### ☐ REPPER: адрес 23562 IY - 48 (5C0Ah)

Содержит число, равное количеству опросов клавиатуры между автоповторами.

---

### ☐ DEFADD: адреса 23563/4 IY - 47 (5C0B/Ch)

В этой ячейке содержится либо адрес контролируемой в данный момент определяемой пользователем функции, либо 0.

---

### ☐ K DATA: адрес 23565 IY - 45 (5C0Dh)

Используется для временного размещения информации о цвете при вводе параметров цвета (т.е. второй байт - номер цвета после "E" курсора).

---

### ☐ TVDATA: адреса 23566/7 IY - 44 (5C0E/Fh)

Аналогична предыдущей, но используется для вывода (применяется также с "AT" и "TAB", поэтому в двух байтах).

---

### ☐ STRMS: адреса 23568 - 23605 IY - 42 (5C10h - 5C35h)

Содержит адреса каналов, связанных с потоками. Изначально потоки от -3 до 3 в первых 14 байтах. Информация о внешних потоках (вплоть до 19) дополняется по мере их открывания в соответствующем месте.

---

### ☐ CHARS: адреса 23606/7 IY - 4 (5C36/7h)

Содержит адрес теоретического старта таблицы литер. Для каждой литеры резервируется 8 байт. Он на 256 меньше, чем его местоположение в памяти, т.к. все литеры адресуются относительно этой ячейки умножением их ASCII кода на восемь, ASCII коды 0-31 непечатаемые, поэтому обрабатываются не этой таблицей. Последний интерпретируемый ASCII код 127 (7Fh), но СПЕКТРУМ использует оставшиеся 128 значений полного диапазона восьми бит для символов-слов. Они обрабатываются отдельно и не затрагивают таблицу литер.

---

### ☐ RASP: адрес 23608 IY - 2 (5C38h)

Содержит продолжительность звука, генерируемого при печати ниже 22 строки или выходе за пределы доступной памяти.

---

### ☐ PIP: адрес 23609 IY - 1 (5C39h)

Содержится продолжительность звучания сигнала при нажатии клавиши.

---

### ☐ ERR NR: адрес 23610 IY + 0 (5C3Ah)

Содержит число, на единицу меньшее, чем код сообщения, генерируемый при ошибке. Может быть использована для ваших собственных сообщений или применения стандартных сообщений для ваших нужд (см. главу 2).

---

### ☐ FLAGS: адрес 23611 IY + 1 (5C3Bh)

Бит 0 установлен при отмене пробела перед выводимым символом или если пробел непосредственно предшествовал символу. Бит сброшен, если пробел должен быть напечатан.

---

Бит 1 установлен, если поток 3 (обычно принтер) должен использовать подпрограммой RST 16, и сброшен для потока 2 (обычно основной экран).

Бит 2 установлен, если печать производится в режиме "L". Сброшен в режиме «K».

Бит 3 установлен, сигнализируя при ожидании ввода о режиме "L", сброшен при режиме «K».

Бит 5 установлен, если после последнего сброса был введен новый ключ (см. «Сканирование клавиатуры» в главе 2 и комментарии к системной переменной "KSTATE").

Бит 6 определяет тип выражения. Установлен для численного и сброшен для строкового, используется интерпретатором БЕЙСИКа.

Бит 7 сброшен при проверке интерпретатором БЕЙСИКа синтаксиса вводимой строки и установлен при выполнении программы.

---

### ☐ TV FLAG: адрес 23612 IY + 2 (5C3Ch)

Бит 0 установлен, если контролируется нижний экран, сброшен - основной.

Бит 3 сигнализирует, что текущий режим мог изменится и нуждается в проверке.

Бит 4 активен, если печатается листинг, в противном случае сброшен.

Бит 5 сигнализирует, что нужно очистить нижний экран (например когда ожидается печать сообщения).

---

### ☐ ERR SP: адреса 23613/4 IY + 3 (5C3D/Eh)

В этих ячейках содержится адрес на машинном стеке, к которому осуществится возврат при ошибке выполнения команды БЕЙСИКа. Он часто меняется БЕЙСИКом и при ошибке берется рестартом. Его можно заменить машинным кодом и при ошибке сделать переход в вашу собственную программу, как показано в программе "DEBASE" (приложение G, раздел "SAVE/LOAD" МИКРОДРАЙВа). При ошибке переменная должна быть сброшена. Ошибка удаляется, если "ERR NR" содержит 255 (FFh). Заметьте, что сообщение "0 OK" считается ошибкой.

---

### ☐ LIST SP: адреса 23615/6 IY + 5 (5C3F/40h)

"LIST SP" используется для сохранения значения указателя стека с тем, чтобы стек мог быть восстановлен по окончании листинга. Это необходимо, т.к. листинг мог быть прекращен в различных местах, с различными величинами на машинном стеке (например путем "N" в ответ на "SCROLL?").

---

### ☐ MODE: адрес 23617 IY + 7 (5C41h)

Ей определяется курсор, используемый при вводе. Влияет лишь на первый нажатый ключ. Исключение - переход к инициализации графического режима. Однако может быть полезна при получении различных курсоров для ввода (например если вы сделаете "POKE MODE,32", вы получите ввод мерцающей графики). Можете поэкспериментировать, т.к. система не испортится.

---

### ☐ NEWPPC: адреса 23618/9 IY + 8 (5C42/3h)

Содержит номер БЕЙСИК-строки следующего интерпретируемого оператора.

---

### ☐ NSPPS: адрес 23620 IY + 10 (5C44h)

Номер следующего интерпретируемого оператора в БЕЙСИК-строке. Запись в ячейки номера строки и номера оператора повлечет за собой переход к указанной точке программы при выполнении.

---

### ☐ PPC: адреса 23621/2 IY + 11 (5C45/6h)

Содержит номер строки обрабатываемого оператора. Используется также программой автономного запуска в соответствии с данными из заголовка.

---

### ☐ SUBPPC: адрес 23623 IY + 13 (5C47h)

Содержит номер обрабатываемого оператора.

---

### ☐ BORDCR: адрес 23624 IY + 14 (5C48h)

Содержит значение цвета бордюра умноженное на восемь и атрибуты для нижнего экрана. Битами 7 и 6 можно сделать нижний экран мерцающим или подсвеченным.

---

### ☐ EPPC: адреса 23625/6 IY + 15 (5C49/Ah)

"EPPC" содержит номер текущей строки (т.е. отмеченной курсором, она будет переведена в область редактора, адресуясь из "E LINE" при вводе команды "EDIT".

---

### ☐ VARS: адреса 23627/8 IY + 17 (5C4B/Ch)

Содержит адрес начала области переменных. Используя вместе с "E LINE" можно рассчитать полную длину переменных БЕЙСИК-программы.

---

### ☐ DEST: адреса 23629/30 IY + 19 (5C4D/Eh)

Содержит адрес первой буквы имени используемой в данное время переменной. Если будет использоваться новая переменная, то 80h загружается непосредственно до того, как в "E LINE" помещается адрес начала новой переменной.

---

### ☐ CHANS: адреса 23631/2 IY + 21 (5C4F/50h)

Здесь содержится адрес начала области информации о каналах. В этой области содержатся параметры открытых каналов.

---

### ☐ CURCHL: адреса 23633/4 IY + 23 (5C51/2h)

Ячейки содержат адрес начала информации об используемом канале в области информации о каналах.

---

### ☐ PROG: адреса 23635/6 IY + 25 (5C53/4h)

Адрес начала области БЕЙСИК-программы. Является следующим байтом после области информации о каналах или любых используемых интерфейсом 1 буферов ввода-вывода.

---

### ☐ NXTLIN: адреса 23637/8 IY + 27 (5C55/6h)

Содержит адрес начала следующей БЕЙСИК-строки в программе.

---

### ☐ DATADD: адрес 23639/40A IY + 29 (5C57/8h)

Адрес конца последнего использованного пункта данных, или начала строки, даваемой командой "RESTORE" или следующей после нее, если команды не было. Это сохраняет порядок, в котором используются элементы данных и, если данных после этой точки нет, то генерируется ошибка "OUT OF DATA".

---

### ☐ E LINE: адреса 23641/2 IY + 31 (5C59/Ah)

"E LINE" - адрес начала редактируемой области, начало любой строки в области редактора.

---

### ☐ K CUR: адреса 23643/4 IY + 33 (5C5B/Ch)

Содержит адрес курсора в текущей строке. Используется при редактировании или написании новой строки в области редактора.

---

### ☐ CH ADD: адреса 23645/6 IY + 35 (5C5D/Eh)

Содержит адрес следующей команды при интерпретации БЕЙСИКом (все числа помечаются "CHRS 14" и опускаются).

---

### ☐ X PTR: адреса 23647/8 IY + 37 (5C5F/60h)

Эти ячейки содержат адрес места, где нарушен синтаксис БЕЙСИКа при вводе новой строки программы, отмеченного "?". Используется также интерпретатором для временного размещения информации.

---

### ☐ WORKSP: адреса 23649/50 IY + 39 (5C61/2h)

Адрес текущей рабочей области, первый байт свободной области, создаваемый вызовом "CALL 5717 (1655h)" подпрограммы "MAKE BC SPACE" см. главу 5, раздел «Стандартные потоки»).

---

### ☐ STKBOT: адреса 23651/2 IY + 41 (6C63/4h)

Адрес начала стека калькулятора. Подробнее разъяснено в главе 8.

---

### ☐ STKEND: адреса 23653/4 IY + 43 (5C65/6h)

Адрес вершины (конец) стека калькулятора. Опять-таки подробнее см. в главе 8.

---

### ☐ BREG: адрес 23655 IY + 45 (5C67h)

Регистр «B» для использования калькулятором (см. главу 8).

---

### ☐ MEM: адреса 23656/7 IY + 46 (5C68/9h)

Содержит адрес области памяти калькулятора (см. главу 8).

---

### ☐ FLAGS2: адрес 23658 IY + 48 (5C6Ah)

Бит 0 установлен, если надо очищать основной экран при расположении строки в основной области.

Бит 1 активен, если буфер принтера использовался программой 16K ПЗУ и сброшен после очистки буфера программой "CLEARING PRINTER BUFFER", описанной в главе 2.

Бит 2 установлен, если экран чист.

Бит 3 установлен после использования "CAPS LOCK".

Бит 4 установлен, если используется канал «K».

---

### ☐ DF SZ: адрес 23659 IY + 49 (5C6Bh)

Количество строк в нижнем экране. Возможен сбой системы БЕЙСИКа если это число меньше трех, т.к. всегда оставляется одна резервная строка между основным экраном и любыми сообщениями в нижнем. Если для сообщений места нет, то интерпретатор переводит для них строки вверх.

---

### ☐ S TOP: адреса 23660/1 IY + 50 (5C6C/Dh)

Номер первой строки программы, выводимой по команде БЕЙСИКа "LIST".

---

### ☐ OLDPPC: адреса 23662/3 IY + 52 (5C6E/Fh)

Содержит номер строки следующей за той, при исполнении которой были выполнены команды "BREAK" или "STOP" (т.е. строки, интерпретированной после ввода команды "CONTINUE").

---

### ☐ OSPPC: адрес 23664 IY + 54 (5C70h)

То же, что и предыдущее, но указывает номер оператора в строке.

---

### ☐ FLAGX: адрес 23665 IY + 55 (5C71h)

Системная переменная "FLAGX" - грубый эквивалент "FLAGS" (адрес 23611 (5C3Bh). Используется интерпретатором при выполнении команд "INPUT" вместо нее.

Бит 1 установлен, если интерпретатор БЕЙСИКа работает с новой переменной.

Бит 5 установлен, когда 16K ПЗУ работает в режиме ввода и сброшен в режиме редактирования.

Бит 6 установлен при обработке строки.

---

Бит 7 установлен, если интерпретатор БЕЙСИКа работает с командой "INPUT LINE".

### ☐ STRLEN: адреса 23666/7 IY + 56 (5C72/3h)

Содержит либо длину используемой в данный момент существующей строковой переменной, либо букву численной или новой строковой переменной в младшем байте (в форме, описанной в разделе выполнение команд "SAVE" и "LOAD", см. главу 2).

### ☐ TADDR: адреса 23668/9 IY + 58 (5C74/5h)

Обычно содержит адрес следующего элемента в таблицах синтаксиса, размещенных в основной ПЗУ начиная с 6782 (1A48h). Однако используется также и для других целей некоторыми подпрограммами.

### ☐ SEED: адреса 23670/1 IY + 60 (5C76/7h)

Является началом случайного числа. Если команда "RANDOMIZE" не имеет числа, то берется из двух младших байтов "FRAMES", в противном случае из числа в команде "RANDOMIZE".

### ☐ FRAMES: адреса 23672 - 23674 IY + 62 (5C78h - 5C7Ah)

По этому адресу содержится трехбайтовый счетчик системы. Установлен в 0 при первом подключении СПЕКТРУМа и инкрементируемый при каждом вызове обычной программы прерываний. Последний значимый байт счетчика 23672 (5C78h).

### ☐ UDG: адреса 23675/6 IY + 65 (5C7B/Ch) (User Defined Graphics)

Адрес начала определенной пользователем графики.

### ☐ COORDS: адреса 23677/8 IY + 67 (5C7D/Eh)

"COORDS" задают координаты X и Y последней выведенной точки. Может быть использована для определения стартовой позиции при выполнении команд "DRAW" и "CIRCLE" без команды "PLOT"; X - горизонталь - в 23677 (5C7Dh).

### ☐ P POSN: адрес 23679 IY + 69 (5C7Fh)

Содержит номер колонки следующей позиции, используемой буфером принтера (такая же как и "S POSN": адреса 23688/9 IY + 78 (5C88/9h) описанная ниже, но для принтера).

---

### ☐ PR CC: адрес 23680 IY + 70 (5C80h)

Это младший байт используемого в данный момент адреса буфера принтера. Идентична "DF CC: адреса 23684/5 IY + 48 (5C84/5h)" и для буфера принтера может быть смещена изменением так называемого неиспользованного байта, описанного ниже, который на самом деле является старшим байтом системной переменной "PR CC". К сожалению он сбрасывается по окончании печати каждой строки, чтобы указать начальный адрес. Поэтому необходима осторожность, если он перемещался.

---

### ☐ NOT USED: адрес 23681 IY + 71 (5C81h)

Обычно не используется, если буфер принтера на обычном месте (см. "PR CC"). Может быть использована, но только как указано выше.

---

### ☐ ECHO E: адреса 23682/3 IY + 72 (5C82/3h)

Как и "S POSN" это колонка и номер строки для следующей позиции печати, но для буфера ввода. Используется при вводе строки на БЕЙСИКе.

---

### ☐ DF CC: адреса 23684/5 IY + 74 (5C84/5h)

Содержит адрес верхней линии пикселей следующей позиции печати. "DF CC" может быть использована для изменения позиции обычных печатаемых символов путем перевода на одну строку вниз, но это может привести к неожиданным последствиям, т.к. программа RST 16 (10h) просто прибавляет 256 (100h) к каждой последующей строке печатаемого элемента. «СХЕМА ЭКРАНА СПЕКТРУМА» в приложении может быть использована для просмотра результатов перевода вниз более чем на одну строку пикселей.

---

### ☐ DFCCL: адреса 23686/7 IY + 76 (5C86/7h)

Это тоже самое, что и предыдущее, но для нижней строки экрана.

---

### ☐ S POSN: адреса 23688/9 IY + 78 (5C88/9h)

Ячейки содержат номера колонки и строки следующей позиции печати в основном экране и устанавливаются путем "CALL 3545 (DD9h)". Эта подпрограмма детально описана в главе 2 (33 - левая колонка, 24 - верхняя строка).

---

### ☐ SPOSNL: адреса 23690/1 IY + 80 (5C8A/Bh)

Как и "S POSN", эти ячейки содержат номера колонки и строки, но для нижнего экрана.

---

### ☐ SCR CT: адрес 23692 IY + 82 (5C8Ch)

Содержит число на единицу большее, чем число переводов экрана без запроса "SCROLL?". Это число должно всегда быть больше 0, т.к. программа в машинных кодах может вылететь, если при запросе получит отрицательный ответ.

---

### ☐ ATTR P: адрес 23693 IY + 83 (5C8Dh)

Содержит атрибуты для любой печати на экран ("PRINT", "DRAW", "PLOT" и т.д.). Устанавливается операторами управления цветом БЕЙСИКа и может быть загружена из программы в машинных кодах. В этом случае все последующие цвета будут изменены.

---

### ☐ MASK P: адрес 23694 IY + 84 (5C8Eh)

Используется в качестве маски для разделения между элементами с заранее определенными цветами и элементами, взятыми из "ATTR P". Для любого бита, установленного в 1, бит атрибута будет взят из атрибутов экрана для текущей позиции, в противном случае - из "ATTR P".

---

### ☐ ATTR T: адрес 23695 IY + 85 (5C8Fh)

Задает временные атрибуты для установки цвета элемента, выводимого командой RST 16 (10h).

---

### ☐ MASK T: адрес 23696 IY + 86 (5C90h)

Используется в качестве маски для разделения между элементами с заранее определенными цветами и элементами, взятыми из "ATTR T". Для любого бита, установленного в 1, бит атрибута будет взят из атрибутов экрана для текущей позиции, в противном случае - из "ATTR T".

---

### ☐ P FLAG: адрес 23697 IY + 87 (5C91h)

Это флаг, используемый для разделения параметров печати для любого вывода на экран через ПЗУ. Четные биты - временные биты, в то время как нечетные - постоянные, каждый относится к одному элементу.

Бит 0/1 установлен, если должна использоваться "OVER 1".

Бит 2/3 установлен, если должна использоваться "INVERSE".

Бит 4/5 установлен, если должна использоваться "INK 9".

Бит 6/7 установлен, если должна использоваться "PAPER 9".

---

### ☐ MEMBOT: адреса 23698 - 23727 IY + 88 (5C92h - 5CAFh)

Эта область используется калькулятором для размещения тех значений, которые нельзя легко разместить на стеке калькулятора (см. главу 8).

Ячейки 23728/9 IY + 118 (5CB0/1h) не используются.

---

### ☐ RAMTOP: адреса 23730/1 IY + 120 (5CB2/3h)

Основное использование этого адреса, содержащего адрес последнего байта области БЕЙСИКа, состоит в гарантировании того, что стек калькулятора имеет достаточный объем (см. главу 8). Адрес устанавливается БЕЙСИКовой командой стирания числа, которая также инициирует машинный стек с этого адреса.

---

### ☐ P-RAMT: адреса 23732/3 IY + 122 (5CB4/5h)

Содержит адрес последнего байта физической ОЗУ.

Далее следуют системные переменные 8K ПЗУ, которые можно использовать только после установки в компьютер интерфейса МИКРОДРАЙВа.

## СИСТЕМНЫЕ ПЕРЕМЕННЫЕ 8K

FLAGS3: адрес 23734 IY + 124 (5CB6h)

Бит 0 установлен, если выполняется расширенная команда.

Бит 1 установлен, если выполняется команда "CLEAR#".

Бит 2 установлен, если системная переменная "ERR SP" была изменена 8K ПЗУ.

Бит 3 установлен, для подпрограмм "NETWORK".

Бит 4 установлен, для подпрограмм "LOAD" 8K ПЗУ.

---

Бит 5 установлен, для подпрограмм "SAVE" 8K ПЗУ.

Бит 6 установлен, для подпрограмм "MERGE" 8K ПЗУ.

Бит 7 установлен, для подпрограмм "VERIFY" 8K ПЗУ.

---

### ☐ VECTOR: адреса 23735/6 IY + 125 (5CB7/8h)

Содержит адрес, по которому будет сделан переход, если обнаружится ошибка в синтаксисе интерпретаторами 16K и 8K ПЗУ. Обычно установлена в 496 (1F0h) - это ошибка 16K ПЗУ. Но может быть изменена для указания программ, проверяющих синтаксис. Позже это изложено в главе 7.

---

### ☐ SBRT: адреса 23737 - 23746 (5CB9h - 5CC2h)

Вообще-то это не системная переменная, а небольшая подпрограмма, используемая 8K ПЗУ для вызова подпрограмм 16K ПЗУ. Подробно описана в главе 3.

---

### ☐ BAUD: адреса 23747/8 (5CC3/4h)

Значение, используемое для определения скорости "BAUD" ввода/вывода RS232. Рассчитывается следующим образом:

3 500 000 / (26 * скорость "BAUD") - 2 = BAUD

2 500 000 - скорость часов СПЕКТРУМа. Может быть использована для получения нестандартных скоростей. Запрещенное значение - 12 (0Ch) - соответствует скорости 19 200.

---

### ☐ NTSTAT: адрес 23749 (5CC5h)

Номер пункта сети, соединенного со СПЕКТРУМом в данное время.

---

### ☐ IOBORD: адрес 23750 (5CC6h)

Содержит цвет бордюра во время операций ввода/вывода, загружается номером цвета. Обычно это 0 - черный, но может быть изменена.

---

### ☐ SER_FL: адреса 23751/2 (5CC7/8h)

Используется при вводе RS232. Первый байт - флаг, установленный в 0 на входе в программу ввода и в 1 при получении байта. Второй байт - полученный байт на выходе из подпрограммы ввода.

---

☐ **SECTOR: адреса 23753/4 (5CC9/Ah)**

Используется 8K ПЗУ для подсчета секторов МИКРОДРАЙВа.

---

☐ **CHADD_: адреса 23755/6 (5CCB/Ch)**

Это эквивалент "CH_ADD" 16K ПЗУ, но для 8K ПЗУ. Используется для размещения адреса "CH_ADD" при проверке расширенного синтаксиса и при необходимости перемещается.

---

☐ **NTRESP: адрес 23757 (5CCDh)**

Содержит код передаваемого в сеть ответа. Следующие 8 байт делают заголовок сети (как описано в главе 3).

---

☐ **NTDEST: адрес 23758 (5CCEh)**

Содержит пункт сети, на который работает вывод.

---

☐ **NTSRCE: адрес 23759 (5CCFh)**

Содержит адресуемый пункт сети.

---

☐ **NTNUMB: адреса 23760/1 (5CD0/1h)**

Содержит номер блока информации, пройденного в данное время по сети.

---

☐ **NTTYPE: адрес 23762 (5CD2h)**

Содержит идентификатор блока сети, 0 - для обычного блока, 1 - для последнего блока.

---

☐ **NTLEN: адрес 23763 (5CD3h)**

Содержит длину перемещаемого в сети блока.

---

☐ **NTDCS: адрес 23764 (5CD4h)**

Содержит контрольную сумму для следующего блока данных.

---

☐ **NTHCS: адрес 23765 (5CD5h)**

Содержит контрольную сумму для семи байтов заголовка.

Следующие восемь байтов создают спецификацию первого файла.

---

### ☐ D_STR1: адреса 23766/7 (5CD6/7h)

При работе МИКРОДРАЙВа задает номер драйва (из двух байтов).

При работе сети задает номер пункта назначения.

При работе RS232 скорость "BAUD".

Более полное описание "D_STR1" см. в главе 3, где каждый из способов использования обсуждается при объяснении работы программ, в которых эта переменная применяется.

---

### ☐ S_STR: адрес 23768 (5CD8h)

Содержит номера потока (0-15).

---

### ☐ L_STR: адрес 23769 (5CD9h)

В вышеприведенном случае содержит тип канала.

---

### ☐ N_STR: адреса 23770/1 (5CDA/Bh)

Содержит длину имени файла.

---

### ☐ T_STR1: адрес 23772/3 a(5CDC/Dh)

"T_STR1" содержит адрес начала имени файла.

Следующие восемь байтов используются командами "LOAD" и "MOVE".

---

### ☐ D_STR2: адреса 23774/5 (5CDE/Fh) до T_STR2: адреса 23780/1 (5CE4/5h)

Эти восемь байтов аналогичны предыдущим восьми, задающим спецификацию первого файла. Следующие байты в точности эквивалентны байтам заголовка подпрограмм 16К ПЗУ, но используются 8К ПЗУ (об объяснении см. в главе 3).

---

### ☐ ND_00: адрес 23782 (5CE8h)

Содержит тип файла: 0 - подпрограмма, 1 - массив чисел, 2 - массив строк, 3 - коды.

---

### ☐ HD_OB: адреса 23783/4 (5CE7/8h)

Содержит длину данных.

---

☐ **HD_0D: адреса 23785/6 (5CE9/Ah)**

Содержит начало данных.

---

☐ **HD_0F: адреса 23787/8 (5CEB/Ch)**

Содержит имя массива или длину программы.

---

☐ **HD_11: адреса 23789/90 (5CED/Eh)**

Содержит номер строки автономного старта. Может также использоваться обходным кодом 32h (см. главу 3).

---

☐ **COPIES: адрес 23791 (5CEFh)**

Задает число копий, выдаваемых при команде "SAVE".

# ГЛАВА 5

# ПОРТЫ И КАНАЛЫ ВВОДА ВЫВОДА

В стандартном СПЕКТРУМе адресная шина и шина данных выведены на разъем с обратной стороны корпуса и БЕЙСИК допускает связь с внешним миром с помощью команд "IN" и "OUT". Используя команды "IN A,(C)" и "OUT A,(C)" процессора Z80 вы распределяете всю адресную шину для однозначного определения, какое из периферийных устройств будет контролироваться.

---

☐ **Порт 254 (FEh).**

Используется как выводной для цвета бордюра, вывода на ленту и для управления внутренним громкоговорителем. Им также поддерживается клавиатура и ввод с ленты. Короткое резюме на этот счет дано в главе о 16K ПЗУ, и позднее в этой главе я приведу более полное его обсуждение.

---

☐ **Порт 251 (FBh) - ZX принтер.**

Используется как для ввода, так и для вывода.

---

☐ **Порт 247 (F7h).**

Передает данные как для сети, так и для RS232 связей ввода/вывода.

---

### ☐ Порт 239 (F0h).

Контролирует МИКРОДРАЙВ и подсоединение RS232 ввода-вывода интерфейса 1.

### ☐ Порт 231 (E7h).

Содержит данные МИКРОДРАЙВа при считывании и печати.

Последний порт является одной из причин несовместимости некоторых периферийных устройств с интерфейсом МИКРОДРАЙВа. В справочнике СПЕКТРУМа забыли указать, что он будет использоваться. Я поочередно рассмотрю детали работы наиболее полезных портов.

### ☐ ПОРТ 254 (FEh) 11111110 BIN

Через порт считывается клавиатура, биты 0-4. Каждая линия разбита на две секции по пять. Для каждой секции крайняя кнопка - бит 0, ближайшая к середине - бит 4. Каждый бит активизирован до тех пор, пока кнопка не будет нажата, в этом случае бит обнуляется. Адресная линия используется, чтобы определить, правая или левая половина считывается командой "IN". Бит, который должен быть обнулен для каждой линии указан ниже:

| | | | | |
|---|---|---|---|---|
| A0 | CAPS SHIFT | до V | 254 | (FEh) |
| A1 | A | до G | 253 | (FDh) |
| A2 | Q | до T | 251 | (FBh) |
| A3 | 1 | до 5 | 247 | (F7h) |
| A4 | 0 | до 6 | 239 | (EFh) |
| A5 | P | до Y | 223 | (DFh) |
| A6 | ENTER | до H | 191 | (BFh) |
| A7 | BREAK/SPACE | до B | 127 | (7Fh) |

К сожалению 5-7 биты считываемых данных устанавливаются непредсказуемым образом. Считывание клавиатуры было изменено в третьей версии СПЕКТРУМа, разрешив некоторые проблемы в программах, использующих «неочищенное» обращение к клавиатуре. Программа сканирования клавиатуры СПЕКТРУМа их игнорирует. Этот факт открывает возможность их использования для других нужд, например для определения функциональных клавиш, как на "BBC" и других компьютерах, и при сканировании программой опроса-прерывания. Можно одновременно считывать более чем один ряд клавиатуры, путем обнуления нужных битов перед считывание, но

это не позволяет сделать однозначное разделение между различными рядами кнопок. Примечание: особое внимание необходимо при считывании более чем одной одновременно нажатой кнопки. СИНКЛЕР не создал никакой защиты против обратной связи между рядами кнопок. Это означает, что если одновременно нажаты "A", "S" и "W" то линия от "Q" до "T" будет сканирована так, как будто "Q" нажата, даже если это не так. Это произойдет потому, что при нажатии двух кнопок на разных адресных линиях но с одинаковым битом данных ряды соединяются вместе и любой другой ключ, нажатый в любом ряду, сбросит указанный бит на обоих рядах, т.к. оба других ключа нажаты.

Бит 6 - ввод с магнитофонного разъема. Имеет тенденцию быть сброшенным, хотя сбрасывается командой "OUT".

Обратите внимание, что любой вывод в биты 0-2 установит цвет бордюра. Наиболее надежный способ установить все биты в 1 - вывести 248 (F8h), но это не нужно, если декодирование делается как положено и к тому же будет временной мерой.

При выводе биты 0, 1 и 2 контролируют цвет бордюра. Бит 0 контролирует синий, 1- красный, 2 - зеленый. Все прочие цвета могут быть получены сочетанием этих трех.

| Цвет | Номер | Двоичный код |
|---|---|---|
| Черный | 0 | 00000000 |
| Синий | 1 | 00000001 |
| Красный | 2 | 00000010 |
| Сиреневый | 3 | 00000011 |
| Зеленый | 4 | 00000100 |
| Бирюзовый | 5 | 00000101 |
| Желтый | 6 | 00000110 |
| Белый | 7 | 00000111 |

Бит 3 контролирует звуковой разъем. Помните, что для того, чтобы получить нечто большее, чем простое щелканье, его необходимо часто включать и выключать, чтобы получить тон. "MIC", напечатанное на стенке СПЕКТРУМа, несколько дезориентирует, так как он дает выход на магнитофон и втыкать туда микрофон будет пустой тратой времени.

Бит 4 контролирует динамик внутри СПЕКТРУМа и к нему применимы те же требования по использованию, что были ука-

заны выше. Программа, демонстрирующая возможности встроенного громкоговорителя и входа "EAR" для считывания речи или музыки, ее размещения в памяти и вывод через динамик, приведены в конце этой главы.

## ☐ ПОРТ 251 (FBh) 11111011 BIN

Бит 0 вход индикация занятости принтера. 0 - принтер занят.

Бит 1 выход Если активизирован - движение тормозится, сброшен (0) - ускоряется

Бит 2 выход сброшен (0) - начало движения

Бит 6 вход активизирован (1) - принтер не подсоединен

Бит 7 выход активизирован - производится печать (побитно)

## ☐ ПОРТ 247 (F7h) 11110111 BIN

Бит 0 выход очередные данные для сети и RS232 вход очередные данные из сети

бит 7 вход очередные данные из RS232

## ☐ ПОРТ 239 (EFh) 11101111 BIN

Возможна проверка работы установки защиты записи набора данных МИКРОДРАЙВа путем IN A,(239) AND 1; и флаг 0 будет установлен, если набор защищен.

Наличие драйва может быть проверено считыванием бита 2 порта 239 после выбора проверяемого драйва, он будет сброшен, если драйв есть.

В битах 3 и 4 порта 239 размещены линии RS232 DTR и CTS соответственно. Нежелательно использовать порты 247, 239 и 231 не из 8K ПЗУ, исключая проверку наличия подчиненных устройств.

Остальные порты доступны для использования другими подчиненными устройствами. Помните, что если подсоединяется принтер или другой интерфейс, то они используют порт или порты для передачи информации. Два из наиболее распространенных интерфейсов параллельных принтеров - "MOREX" (он также имеет двунаправленный RS232 встроенный интерфейс (могу его порекомендовать) и "KEMPSTON", которые используют следующие порты:

# ПОРТЫ 'MOREX' 251 (FBh) 11111011 BIN и 127 (7Fh) 01111111 BIN

## ☐ ПОРТ 251 (FBh) 11111011 BIN

Биты 0-7 выход данные "CENTRONICS OUTPUT"

Бит 0 вход занятость "CENTRONICS"

Бит 1 вход RS232 DTR

Бит 7 вход RS232 данные RX

## ☐ ПОРТ 127 (7Fh) 01111111 BIN

Бит 0 выход CENTRONICS стробирование

Бит 1 выход RS232 данные TX

Бит 2 выход RS232 CTS

## ☐ ПОРТЫ "KEMPSTON" 58047 (E2Fh), 57535 (EOBFh) и 58303 (E3BFh)

Интерфейс "KEMPSTON" может быть использован только командами "IN (C)" и " OUT (C)" т.к. он просматривает все 16 бит адресной шины. Программы, позволяющие переслать один символ в каждый из этих интерфейсов вывода "CENTRONICS", приведены в приложении «Полезные программы».

## ☐ ПОРТ 58047 (E28BFh)

Бит 0 вход линия занятости

## ☐ ПОРТ 57535 (E0BFh)

Биты 0-7 выходы данных "CENTRONICS"

☐ **ПОРТ 58303 выход "CENTRONICS" стробирование.**

Стандартные потоки

☐ **Поток K-3/253 (FDh) - аналог потоков 0 и 1**

☐ **Поток S-2/254 - аналог потока 2**

☐ **Поток R-1/255.**

Используется для записи в рабочую область памяти и разместит текущий код из регистра "A" в адрес, содержащийся в системной переменой "K CUR 23643/4 (5C5B/Ch)" и даст приращение адресу "K CUR". Он не так полезен, как это кажется на первый взгляд, т.к. Сначала вызывается подпрограмма организации окна, и при этом вверх перемещается вся память, находящаяся выше "K CUR", поскольку "RAMTOP" делает поток непригодным для записи чего-либо выше "RAMTOP" или в другое место без перемещения вверх. Подпрограмма стартует в 3969 (F81h). Поток может быть использован только для вывода. Попытки использования его для ввода приведут к ошибке. Потоки 0 и 1K обычно соединены с нижним экраном и клавиатурой. По их каналам также возможен ввод.

☐ **Поток S2 только для вывода, обычно на экран.**

☐ **Поток P3 только для вывода, обычно на принтер.**

Всего доступно 19 потоков. Каждый должен быть связан с каналом; каждый поток имеет два байта в области потоков системных переменных, начиная от 23569 (5C10h) для потока "-3", где содержится смещение присоединенного канала относительно базы области каналов. Помните, что первый поток "-3", поэтому для определения адреса канала потока "0" необходимо просмотреть 23574/5(5C16/7h). Канал состоит из не менее чем пяти байтов, первые два содержат адрес процедуры вывода, следующие два - адрес программы ввода, последний - букву кода канала (S, K, P и т.д. См. выше). Каналы, соединенные с интерфейсом МИКРОДРАЙВа, несколько длиннее, чем 5 байт. Их формат приводится в книге, прилагаемой к интерфейсу.

```
Программа записи и воспроизведения
10; эта небольшая программа позволяет записать
20; речь с магнитофона в память
30; через вход "EAR".
```

```
40; записанная речь затем может быть воспроизведена
50; вновь через динамик СПЕКТРУМа из памяти.
60; программа проста, но эффективна
70;
80;
90 LIMIT EQU 50000 ; низ доступной памяти.
100 TIME EQU 50 ; задержка между битами.
110 FIRST EQU 64998 ; первый используемый
120 ORG 65000 ; байт памяти.
130 START LD HL,FIRST ; подсчитаем количество
140 LD DE,LIMIT ; доступной памяти.
150 PUSH HL
160 AND A
170 SBC HL,DE
180 PUSH HL
190 POP BC
200 POP HL
210 DI ; отключим прерывания, гарантируя
220 LISTEN LD A,#7F ;  ожидание звука.
230 IN A,(#FE)
240 BIT 6,A
250 JR Z,LISTEN
260 PUSH BC
270 BYTEIN LD B,8 ; считаем 8 бит
280 DEC HL
290 HEAR IN A,(#FE) ; побитно
300 RLA
310 RLA
320 RL (HL) ; переведем в память, через CF
330 DJNZ PAUSE ; пауза и следующий бит
340 POP DE ; если 8 бит считано - проверяем
350 DEC DE ; место и изменяем счетчик.
360 LD A,D
370 OR E
380 PUSH DE
390 JR NZ,BYTEIN ; если место есть - идем на
400 POP BC ; считывание, иначе 410 EI ; заканчиваем
420 RET
430 PAUSE LD C,TIME
```



Поток P3 только для вывода, обычно на принтер.

```
440 DELAY DEC C
450 JR NZ,DELAY
460 JR HEAR
470 SPEAK LD HL,FIRST ; как для "LISTEN"
480 LD DE,LIMIT
490 PUSH HL
500 AND A
510 SBC HL,DE
520 PUSH HL
530 POP BC
540 POP HL
550 PUSH BC
560 DI
570 BYTEOT LD B,8 ; вывод 8 бит
580 DEC HL
590 LD A,(HL) ; отметьте, что выводятся все биты
600 RRCA ; и изменится цвет бордюра
610 RRCA
620 RRCA
630 BITOUT OUT (#FE),A ; побитно, как при вводе
640 RLCA
650 LD C,TIME
660 HOLD DEC C
670 JR NZ,HOLD
680 DJNZ BITOUT
690 POP BC
700 DEC BC
710 LD A,C
720 OR B
730 PUSH BC
740 JR NZ,BYTEOT
750 POP BC
760 EI
770 RET
```

Поток P3 только для вывода, обычно на принтер.

# ГЛАВА 6
# ИСПОЛЬЗОВАНИЕ ПРЕРЫВАНИЙ

## ПРЕРЫВАНИЯ

Стартовая последовательность СПЕКТРУМа, очищающая память и устанавливающая значения системных переменных, также инициализирует регистр прерываний, размещая в нем 63 (3Fh) и устанавливая режим прерываний в 1 (IM1). Задание регистра "I" необязательно, т.к. режим "IM1" его не использует, потому что любые прерывания выполняются через RST 56 (38h). В новой версии, в которой "ULA" СПЕКТРУМа содержит вывод, биты 6 и 7 регистра "I" задействованы.

При каждом цикле машинных команд Z80 обращается к ячейке памяти, адресуемой регистром "I", выводя его в старшие 8 битов адресной шины, а линия запроса памяти активизируется. "ULA" генерирует прерывание каждый раз, когда нужно изменить содержимое экрана. Это заставляет Z80 запустить программу обработки прерываний, при условии, что прерывания подключены. Обычно это сканирование клавиатуры и изменение счетчика системы, но если подключено ПЗУ интерфейса, то все, что делается при включении прерываний - немедленный возврат без сканирования клавиатуры или каких-либо других действий. Когда программа прерываний выполнена, ЦП возвращается в точку, в которой он был при прерывании. Если это команды считывания/записи в память между 16384 (4000h) и 32767 (7FFFh), которые "ULA" проверяет, просматривая две старшие линии адресной шины и линию "MREQ", "ULA" останавливает часы на ЦП до окончания изменения экрана. Если старший бит регистра "I" сброшен, а бит 6 активизирован, "ULA" может запутаться из-за регенерации динамической памяти во время T3 и T4 в M1 цикле. Активизируется линия "MREQ" и регистр "I" помещается в старшие восемь битов адресной шины. Далее "ULA" думает, что ЦП производит считывание или запись в эту область ОЗУ, даже при попытке это предотвратить и "ULA" пропускает свое собственное обращение для изменения дисплея, что приводит к развалу картинки. Поэтому в регистре "I" не должно содержаться любое число от 64 до 127 (40h до 7Fh) включительно, т.е. с двумя старшими битами установленными таким образом.

Путем установки "IM2" можно использовать прерывания для ваших собственных целей, до тех пор, пока вы выполняете RST 56 (38h) в конце вашей программы обработки прерываний, которая подключит прерывания перед возвратом в основную программу, если нужно, чтобы счетчик обнулялся, а клавиатура сканировалась и "RETI" в конце концов.



Поток P3 только для вывода, обычно на принтер.

### ☐ Если вы использовали RST 56

(38h) в прерывающей программе, необходимо выполнять команду "EI" до "RETI", если нужно подключить прерывания, чтобы снова вызвать программу обработки прерываний. Помните, что вам придется сбрасывать режим "IM1" и подключать прерывания до возвращения в БЕЙСИК, если вы не пользуетесь RST 56 (38h) в прерывающей программе.

Режим "IM2" сложен. По получении прерывания (50 раз в секунду), ЦП запоминает адрес следующей команды программы на машинном стеке. Отключает любые последующие прерывания. Затем просматривает ячейку, указанную шиной данных, + (256 * "I" регистр) и передает управление к адресу, содержащемуся в этой ячейке + (256 умноженное на содержимое следующей ячейки). Вообще-то считается дурным тоном иметь бит 0 на шине данных активизированным для использования в качестве указателя в "IM2", т.к. указатель всегда будет стартовать с адреса, пронумерованного четным числом, но к сожалению в СПЕКТРУМе нет выбора.

Пример. Регистр "I" содержит 10 (0Ah), и шина данных содержит 225 (FFh). При этом 10 * 256 = 2560 и 2560 + 255 = 2815, поэтому точка, к которой будет совершен переход, будет взята из содержимого адреса 2815 + (256 * на число по адресу 2816). Ячейка 2815 содержит 34, а 2816 - 128. В этом можно убедится с помощью команды "PEEK" вашего СПЕКТРУМА, т.к. она есть в ПЗУ. Таким образом адрес перехода 34 + (256 * 128), а это 32802. Другой пример : регистр "I" содержит 6: 6 * 256 = 1536 и 1536 + 255 = 1791. 1791 содержит 221, 1792 содержит 113. 221 + (113 * 256) = 29149, переход будет к 29149. Иначе, если у вас 48K СПЕКТРУМ и регистр "I" содержит 200: 200 * 256 = 51200 и 51200 + 255 = 51455, переход будет по адресу, который вы положите сюда и в следующую ячейку, в обычном Z80 сначала идут младшие байты.

Это может быть представлено как воображаемая невидимая команда прерывания в выполняемой программе. В момент прерывания эта команда выполняется так, как "DI", сопровождаемое командой "CALL" по адресу, предшествующему адресу, указанному регистром "I" и шиной данных. Вызываемый адрес в следующих двух байтах, (младший байт идет первым). Команда, будучи невидимой, не может разместить собственный адрес возврата на машинный стек, поэтому адрес, после последней выполненной команды кладется на стек и к нему будет осуществлен возврат после команды "RETI" в конце программы обработки прерываний.

Команде "RETI" должна предшествовать команда "EI". Причина, по которой "DI" включено в вызов, выполняемый прерыванием, состоит в том, чтобы гарантировать, что программа не зациклит петлю,

если процедура обработки прерываний выполняется продолжительнее, чем пауза между двумя прерываниями.

Довольно просто написать ряд команд, которые заменят адрес перехода по прерыванию с помощью загрузки байтов указателя адресом, а это две ячейки, за которыми мы следим для определения места перехода. Примечание: когда бы не использовались программы прерываний, очень важно, чтобы любые регистры, используемые программой прерывания, сохранялись на входе и восстанавливались перед возвратом в основную программу. Не должно быть попыток пересылки данных через регистры в или из программы прерывания. Из - за ограничения на величины, которые могут содержаться в регистре "I", есть лишь ограниченное количество адресов, к которым может быть осуществлен переход в 16K СПЕКТРУМе. Это диктуется содержимым ПЗУ. Дополнительная проблема при использовании прерываний из ПЗУ связана с интерфейсом МИКРОДРАЙВа, который изменяет указатель "I" во время работы. Список указателей для версии 2 СПЕКТРУМА и версии 1 интерфейса МИКРОДРАЙВа дан в приложении F, но если вы не уверены в том, какая у вас версия, или у вас различные версии, то необходима проверка. Для коммерческого матобеспечения опасно использовать адреса ПЗУ, т.к. любые изменения и любые дополнения в нем могут сделать ваше матобеспечение неработающим. Типичное использование процедур прерываний - управление спрайтами и сопровождающий звук программы. Зная, с какой частотой генерируются прерывания, легко рассчитать скорость перемещения спрайта, она будет независима от любых других операций внутри программы. Использование программ ПЗУ при прерывании затруднено возможностью подключения ПЗУ интерфейса во время прерывания, Это должно быть учтено при написании программы. Например, если программа "SPRITE" использует 16K ПЗУ при рисовании на экран с помощью вызова "PLOT" в 8933 (22E5h) при подключенном ПЗУ интерфейса, вызов 8933 произойдет в ПЗУ интерфейса. Т.к. такого адреса в нем нет, то это приведет к поломке программы.

Один из способов решения проблемы - включить в прерывающую программу проверку того, какая ПЗУ подключена. Самый простой путь проверки - посмотреть адрес ПЗУ, содержащий различные значения в разных ПЗУ. Рекомендую использовать адрес 20 (D5h) в ПЗУ интерфейса и 255 (FFh) в 16K ПЗУ и в зависимости от результата предпринять соответствующее действие. Если вызов к 16K ПЗУ и оно подключено, то вызов выполняется напрямую, а если подключено 8K ПЗУ, то через RST 16. Для 8K ПЗУ можно использовать обходной код 32h (см. Главу 3).

Я привел простую программу "SPRITE" в приложении G. Эта программа передвигает группу из четырех пикселей, отражая их от четырех углов экрана, выдает звуки и меняет цвет бордюра, демон-

стрируя тем самым способы обхода возникших проблем. Так как полное понимание использования прерываний очень важно (если, конечно, от них вообще есть польза), я полагаю, что вы войдете в программу используя ваш АССЕМБЛЕР. Листинг взят прямо из моего АССЕМБЛЕРа, чтобы гарантировать отсутствие ошибок; несколько необычно, что шестнадцатиричные числа предварены символом "#", а двоичные - "%". Программу можно и переместить, если рассчитать новое значение указателя и изменить регистр "I". Если вы ввели и ассемблировали программу, перед попыткой ее выполнить перечитайте еще раз эту страницу. Первая проблема, с которой вы столкнетесь, если у вас 16К ПЗУ СПЕКТРУМа в том, что указатель должен соответствовать ПЗУ. К сожалению, при написании я не нашел пути использования 16К ПЗУ СПЕКТРУМа с интерфейсом МИКРОДРАЙВа.

Перед тем, как что-либо предпринять, сохраните исходный и объектный коды на ленте или МИКРОДРАЙВе и, если у вас есть МИКРОДРАЙВ, поместите набор данных. Для инициализации спрайта нужно вызвать подпрограмму, помеченную в листинге "SETUP" и, если указатель не изменялся, то "RANDOMIZE USR 51457". Теперь можно увидеть одну черную точку, двигающуюся вокруг экрана. Если ее нет, то заново проверьте ваш код. Наличие спрайта не повлияет на другие действия компьютера, так что в программу можно войти и выполнять ее как обычно до тех пор, пока она не испортит память, используемую программой прерываний. Некоторые команды БЕЙСИКа, которые работают на прерываниях, могут быть продемонстрированы.

Введите строку на БЕЙСИКе:

```
10 BEEP 5,60 : FOR N=1 TO 100 : NEXT N : GO TO 10
```

При выполнении вы увидите, что "SPRITE" останавливается, если выполняется "BEEP". Есть другие команды, которые отключают прерывания, это те, что используются в "SAVE" и "LOAD". "SPRITE" движется со скоростью 50 пикселей в секунду и потребуется 3.5 секунды для перехода с нижней части экрана наверх.

# ГЛАВА 7
# РАСШИРЕННЫЙ БЕЙСИК С ИНТЕРФЕЙСОМ 1

## ВУТРИ БЭЙСИКа

При добавлении ИНТЕРФЕЙСА 1, любая команда БЕЙСИКа, не прошедшая проверки на синтаксис СИНКЛЕРа, обычно возвращается в программу обработки ошибок по адресу, содержащемуся в новой системной переменной "VECTOR". Она может быть изменена

для указания адреса в ОЗУ. Это позволяет написать программу, осуществляющую проверку и влияющую на команды, для обработки которых она была написана. Перед тем, как использовать эту возможность, необходимо понять метод проверки строк с тем, чтобы добавленная программа могла делать аналогичные действия. Как только нажата кнопка "ENTER", при вводе строки на БЕЙСИКе во входной буфер, вызывается интерпретатор ПЗУ. Он обрабатывает строку, но не выполняет команд, т.к. флаг синтаксиса (бит 7 в "FLAG") сброшен. Если есть какая-либо ошибка, то об этом сигнализируется символом "?" и ошибка должна быть исправлена до того, как строка включена в программу.

Этот же процесс осуществляется и во время выполнения, но т.к. флаг синтаксиса активен, команда выполняется. Помните, что когда бы ни использовалась программа ОЗУ как часть интерпретатора БЕЙСИКа, 8K ПЗУ будет подключена. В обеих ПЗУ есть программы проверки флага синтаксиса и возврата в вызывающую программу во время выполнения. Это очень полезно при добавлении любых команд (смотри примеры в конце этой главы). Основные критерии для проверки синтаксиса даны ниже, а процедуры ПЗУ использовались настолько, насколько это было возможно. Адрес строки, в которой синтаксис нарушен, будет размещен в системной переменной "CH_ADD" до тех пор, пока используется ПЗУ, а затем из ОЗУ вызывается расширенная программа проверки синтаксиса. Вследствие этого очень важно быть уверенным, что первая строка расширенной команды БЕЙСИКа не соблюдает нормальный синтаксис. Если это не так, интерпретатор начнет исполнение и будет практически невозможно восстановить контроль. По этой причине наиболее просто использовать неалфавитный символ в начале дополнительной строки, что выводит систему из режима «K» или режима команд, гарантируя нарушение синтаксиса. Символы "*" и "!" подходят для этого как нельзя лучше. Первое, что нужно проверить - строки команд. Это делается поочередной загрузкой строк и поиском конца рассматриваемой строки. Программа 16K ПЗУ в 24 (18h) загрузит строку в регистры. Программа с 32 (20h) запросит следующую строку и нарастит "CH_ADD" на единицу. Далее каждая строка может быть проверена и, если она некорректна, то вызывается обработчик ошибок переходом к исходному "VECTOR", адрес которого изначально был 496 (1F0h). Далее:

1) Если затем идет численное выражение, может быть применена 16K подпрограмма с 7298 (1C82h). Если нет, то это приведет к ошибке. При выполнении программа поместит значение на стек калькулятора.

2) Если далее идут два численных выражения, разделенных запятой, можно использовать программу 16K ПЗУ с 7290

(1C7Ah). Она работает как и предыдущая, но при выполнении помещает на стек оба значения.

3) Если далее идет строковое выражение (либо в кавычках, либо со знаком доллара), то может быть применена программа 16K с 7308 (1C8Ch). Опять-таки она работает как предыдущая, но генерирует ошибку, если это не строка. Во время выполнения на стек помещаются параметры строки, как описано в конце главы 8. Примечание: во всех трех приведенных выше программах первый код оцениваемого выражения должен быть в регистре «A» и перед вызовом "CH_ADD" должна содержать его адрес. После оценки регистр «A» будет содержать первый код, идущий после выражения, а "CH_ADD" его адрес. При использовании символов переменных в оцениваемых выражениях допустимы математические действия. Это можно использовать при нахождении параметров переменных. Например выражение A*(B+(C/(D+E))) будет вычислено, а результат помещен на стек во время выполнения.

Когда оценка завершена, "CALL 1463 (5B7h)" проверит наличие конца строки (код двоеточия или возврата каретки 13 (0Dh)) и осуществит либо возврат к интерпретатору (для проверки синтаксиса), либо к вашей программе (для выполнения), если конец строки найден. Если он не найден - генерируется ошибка. При выполнении ваша программа использует команды, берущие информацию со стека калькулятора, а затем возвращается к интерпретатору через команду "JP 1473 (5C1h)". "CH_ADD" указывает на следующую строку для интерпретации.

Для демонстрации расширения команд следующая программа создает !CALL NN и !FRE. !CALL NN вызывает программу в машинных кодах по адресу после "!CALL".

```
!FRE ВЫЧИСЛЯЕТ ОБЪЕМ СВОБОДНОЙ ПАМЯТИ.

!CALLRST 16
DEFW 24          ;ввод символа
CP   "!"         ;это !
JR   NZ,ERROR    ;если не разрешен - ошибка
CALL NEXT_CH     ;ввод следующего символа
CP   "C"         ;это C
JR   NZ,!FRE     ;если нет, переход к про-
                 ;верке, может быть это !FRE
CALL NEXT_CH
CP   "A"         ;каждый символ проверяется,
JR   NZ,ERROR    ;если нет совпадения, то
```

```
                        ;просмотр продолжается
CALL NEXT_CH
CP   "L"
JR   NZ,ERROR
CALL NEXT_CH
CP   L"
JR   NZ,ERROR
JR   ISCALL        ;если дошли досюда - все
                   ;слово совпадает
!FRE CP "F"        ;то же самое для !FRE
JR   NZ,ERROR
CALL NEXT_CH
CP   "R"
JR   NZ,ERROR
CALL NEXT_CH
CP   "E"
JR   NZ,ERROR
CALL NEXT_CH
JR   Z,ISFRE       ;это !FRE
ERROR JP 496       ;обычный указатель
ISCALL CALL NEXT_CH
RST  16
DEFW 7298          ;оценивает следующее
     ;выражение как численное. Если не так
     ; ошибка. При выполнении значение
     ;помещается на стек калькулятора.
CALL 1463          ;проверяет конец оператора
                   ;на  БЕЙСИКе
RST  16            ;делает возврат только
                   ;при выполнении
DEFW 11682         ;помещает в "BC",см.Главу 8
JR   C,ERROR       ;перенос устанавливается,
                   ;если  больше 65535
LD   (DEST),BC     ;адрес для вызова RST 16
DEST DEFS 2        ; он помещается сюда
JR   FINIS         ;16K ПЗУ подключена при
     ;выполнении вызванной процедуры
ISFRE CALL 1463    ;это при синтаксисе
LD   HL,00         ;очистим HL
ADD  HL,SP         ;добавим адрес в SP
LD   DE,(23653)    ; это STKEND
SBC  HL,DE         ;из адреса в SP, дает сво-
                   ;бодную память для БЕЙСИКа
PUSH HL
POP  BC            ;результат теперь в BC
```

```
RST  16
DEFW 11563        ;помещаем на стек BC,
                  ;см.Главу 8
RST  16
DEFW 11747        ;выводит значение на стек
FINIS JP 1473     ; возврат в БЕЙСИК
NEXT_CH RST 16
DEFW 32           ;программа NEXT_CH из ПЗУ
AND  223          ;обращение к вышеприведен-
                  ;ному случаю RET
```

Обычно команды или функции расширенного БЕЙСИКа завершаются переходом к основному интерпретатору как указано выше, что очень важно, т.к. это 8К ПЗУ, которая подключается, когда делается этот переход, в противном случае программа ломается.

## КАЛЬКУЛЯТОР

СПЕКТРУМ содержит в ПЗУ мощный калькулятор, который может быть применен для удобства при программировании в машинных кодах.

Всего он имеет 66 разных подпрограмм. Я объясню в деталях лишь некоторые наиболее употребимые. Для использования калькулятора необходимо уяснить, в какой форме СПЕКТРУМ содержит числа, как их размещать, чтобы они были доступны калькулятору и как получать ответ по окончании расчетов.

Все используемые калькулятором числа размещаются в пяти байтах, либо в двоичном виде с плавающей точкой, либо в виде целого числа.

### ☐ МАЛОЕ ЦЕЛОЕ ПРЕДСТАВЛЕНИЕ

Первый байт всегда 0.

Второй байт - знак, 255 (FFh) для отрицательных, 0 - для положительных чисел.

Третий и четвертый байты - само число в стандартном формате Z80 - первый байт младший.

Последний байт всегда 0. Число 0 рассматривается как положительное.

### ☐ ПРЕДСТАВЛЕНИЕ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Первый байт - показатель степени: это количество перемещений двоичной точки влево или вправо до активизации бита 7. Бит 7 показателя степени определяет направление движения. Установлен

- точка двигалась влево, сброшен - вправо. Рассмотрим пример: Десятичное число 126 (7Eh), в двоичном виде 01111110. Двоичная точка (как мы должны ее показать) будет справа. Она должна совершить семь перемещений влево, чтобы последний установленный бит был от нее справа. Картинка такая:

0 перемещений 0 1 1 1 1 1 1 0

1 перемещение 0 1 1 1 1 1 1.0

2 перемещения 0 1 1 1 1 1.1 0

3 перемещения 0 1 1 1 1.1 1 0

так до тех пор пока

7 перемещений 0.1 1 1 1 1 1 0

Любые биты слева от точки всегда должны быть сброшены и могут быть опущены, т.к. мы знаем ее местоположение. Обычно мы сами делаем это для десятичных чисел, хотя и неосознанно, не указывая десятичную точку справа от целого числа т.к. и так ясно, где она.

Этот процесс дает нам ту часть числа, которая известна как мантисса, в приведенном примере это 11111110 в двоичном виде и показатель степени (количество перемещений двоичной точки влево) равен семи. Самый важный бит - бит, определяющий знак числа, он всегда должен быть задан, активизирован для отрицательных и сброшен для положительных чисел. Показатель также выражен в двоичном виде, бит 7 установлен, если двоичная точка двигалась влево, сброшен - вправо. Поэтому в приведенном примере, когда мантисса равна 7 (в двоичном виде 00000111), но точка двигалась влево, бит 7 должен быть установлен, значит 100000111 или 135 в десятичном виде. Теперь число может быть представлено в полной пятибайтовой форме:

| | Показатель | Мантисса |
|---|---|---|
| Двоичное | 100000111 | 01111100 00000000 00000000 00000000 |
| Десятичное | 135 | 124 0 0 0 |
| 16-ричное | 87 | 7C 0 0 0 |

Обычно приведенное в примере число размещается в малом целом представлении, но здесь оно было использовано для простоты.

Калькулятор использует собственный стек, на котором держит любые числа, с которыми работает. Первое, что нужно сделать перед проведением расчетов - поместить числа, с которыми будете работать, на стек калькулятора. Это может быть достигнуто тремя основными способами:

1) Число может быть помещено на стек из регистра или пары регистров, используя подпрограмму ПЗУ для перевода в вид, требуемый калькулятором.

2) Число может быть приведено к виду, понимаемому калькулятором, и затем помещено на стек.

3) Число может быть записано в память в ASCII представлении и программа проверки синтаксиса БЕЙСИКа, используемая для его считывания и размещения на стек калькулятора, приведет его к корректной форме. Каждый из способов имеет свои преимущества и недостатки, и каждый применим к определенному типу чисел. Я сейчас по очереди рассмотрю их применение.

1) Для небольших целых чисел можно использовать две подпрограммы:

---

### ☐ CALL 11560 (2D28h)

Используется для помещения в регистр «A» числа, пересылаемого на стек калькулятора. Очевидно, что число ограничено 0-255 (0-FFh) и должно быть положительным.

---

### ☐ CALL 11563 (2D2Bh)

Заберет числа в интервале 0-65535 (0-FFFFh) из пары регистров "BC". Опять таки число может быть только положительным, в противном случае начало подпрограммы будет пропущено. Вызов тогда делается к 11569 (2D31h), с нулем в регистре «A», и 255 (FFh) в регистре "E", тогда число будет передано в калькулятор как отрицательное.

2) Для чисел в пятибайтовой форме, готовой к применению, можно использовать подпрограммы:

---

### ☐ CALL 10934 (2AB6h)

Пятибайтовое представление числа в форме, описанной выше, должно быть в регистрах "A", "E", "D", "C", "B"; показатель степени в регистре «A», мантисса в других четырех по порядку.

3) Представление ASCII:

---

### ☐ CALL 11419 (2C9Bh)

Найдя число во введенной строке, интерпретатор БЕЙСИКа помещает пятибайтовую двоичную форму числа, готовую для исполь-

зования при выполнении программы после версии ASCII. Программа, используемая для этого, может быть применена для конвертирования чисел программы в машинных кодах, написанной пользователем. Это сохраняет все проблемы конвертирования чисел пользователем или написания подпрограммы конвертирования внутри ваших программ, и достойно более детального рассмотрения.

При использовании этой программы для загрузки стека калькулятора системная переменная БЕЙСИКа "CH_ADD 23645 (5C5Dh)" должна содержать адрес первого кода числа, помещаемого на стек, регистр «A» - код цифры из "CH_ADD". Первая цифра обычно самый левый разряд десятичного числа, но т.к. эта программа используется для сканирования строк БЕЙСИКа, это может быть двоичный символ 196 (C4h), если следующие разряды - двоичные числа. Двоичное число может достигать 16 разрядов (число 65535 (FFFh)). Любые попытки превысить эти значения приведут к переходу в программу обработки ошибок БЕЙСИКа. Если нужно, можно использовать "E" формат, тогда число будет переделано сразу после того, как появится в строке на БЕЙСИКе.

После цифр, входящих в число, и показателя, если он используется, должен быть добавлен определенный байт, содержащий 13 (0Dh). Так программе дают понять, что конец числа достигнут и больше ничего просматривать не нужно.

Есть две подпрограммы, обратные подпрограммам "STACK A" и "STACK BC". Они берут последнее число со стека калькулятора и округляют до целого, если это возможно. Когда число слишком велико - флаг переноса при возврате активизируется, и, если число отрицательное, флаг 0 сбрасывается. Для положительного числа флаг 0 установлен. Число будет вычеркнуто из стека изменением указателя, но пара регистров "DE" по-прежнему укажет на него в памяти, допуская восстановление, если возврат не был успешным, хотя проще продублировать число перед попыткой возврата, а затем удалить копию при успешном выполнении. Вызываемые адреса:

---

### ☐ STACK TO A: CALL 11733 (2DD5h) STACK TO BC: CALL 11685 (2DA5h)

Есть процедура, обратная программе (2) (см. выше) с 11249 (2BF1h), которая возвращает последнее значение со стека в тот же регистр. Эта программа также удаляет число со стека, но т.к. она не может не возвратить число, флаги не устанавливаются. Если требуется, чтобы число осталось на стеке после возврата, должна быть сделана копия.

---

## ☐ STACK TO A, E, D, C, B: CALL 11249 (2BF1h)

Когда числа генерируются внутри программы, процедура, печатающая число со стека может быть использована для вывода числа в область памяти - процесс, обратный выводу на экран - в форме ASCII. Максимальное число свободных ячеек в памяти, требуемого для одного числа - 14.

Это достигается написанием программы, которая при каждом вызове помещает содержимое регистра «А» в следующую ячейку памяти, открывая канал, указывающий на вашу подпрограмму, и делая этот канал текущим, помещая его базовый адрес в CURCHL 23633 (5C51h). Для вывода числа может быть вызвана подпрограмма ПЗУ с 11747 (2DE3h). Затем подпрограмма вызывается с каждой цифрой числа по очереди, помещая числа в ячейки памяти готовыми для дальнейшего использования. Помните, что указание адреса, в котором цифры сохранены, не может содержаться в регистре между вызовами вашей программы и не может быть сохранено на машинном стеке, поэтому два байта памяти нужно зарезервировать для сохранения этого адреса. Программа, выполняющая это, выглядит примерно так:

```
SET_UP LD HL,SPACE
LD    (SP_WD),HL
LD    HL,(23633) ;CUR_HL
PUSH HL
LD    DE,START   ;текущий канал размещаем в
      ;вашей программе и сохраняем
      ;исходный адрес для
      ;последующего восстановления
LD    C,(HL)
LD    (HL),E
INC   HL
LD    B,(HL)
LD    (HL),D
PUSH BC
CALL 11747        ;программа вывода чисел
POP   BC          ;восстанавливаем исходный
                  ;адрес и текущий канал
POP   HL
LD    (HL),B      ;восстанавливаем место
                  ; назначения канала
INC   HL
LD    (HL),B
;далее идет остаток программы
STARTLD HL,(SP_WD)
LD    (HL),A
```

```
INC   HL
LD    (SP_WD),HL
RET
SPACE DEFS 16
SP_WD DEFW SPACE
```

Программа, используемая и упомянутая выше, при вызове берет верхнее значение со стека калькулятора и выводит его в ASCII как десятичное число в текущий поток. Число удаляется со стека. Когда бы ни использовался калькулятор, необходимо быть уверенным в сбалансированности его стека. Сам калькулятор при правильном его использовании всегда работает известным образом, но дисбаланс может привести к ошибочным результатам. Между "RAMTOP" и "STKBOT" должно быть оставлено окно для расширения стека калькулятора вверх и роста машинного стека вниз без их пересечения. Так как некоторые подпрограммы калькулятора используют калькулятор рекурсивно, необходимо оставить больше места, чем предположительно будет использоваться, лучше ошибиться в сторону завышения. Если есть сомнения в том, что размещено на стеке калькулятора после завершения выполнения набора команд, стек можно очистить путем "CALL 5823 (16BFh)", но помните, что таким образом сотрется все содержимое стека.

## ☐ ИСПОЛЬЗОВАНИЕ КАЛЬКУЛЯТОРА

Для лучшего понимания работы калькулятора полезно представить его в виде отдельного процессора со своим набором команд. Он подключается RST 40 (28h) и отключается "ENDCALC", код команды 56 (38h).

Пока калькулятор подключен, его операционные коды берутся из ячеек, следующих за RST 40. Это набор отдельных байтов в обычной программе Z80. Когда калькулятор получает команду "ENDCALC", контроль возвращается Z80 и продолжается выполнение программы, начиная с адреса, идущего за "ENDCALC". Некоторые операционные коды требуют операторов, а некоторые могут быть использованы только в начале, т.к. они требуют, чтобы регистры Z80 были установлены для работы определенным образом.

При каждом использовании стека его размер изменяется на пять байтов. Поэтому при каждом размещении числа на стек убедитесь в наличии свободного места. Если размер свободной области не соответствует выполняемой операции, генерируется ошибка БЕЙСИКа. Калькулятор не ограничен в выполнении числовых операций, он также используется для выполнения строк БЕЙСИКа и команд "VAL".

Об этом мы поговорим позже.

Операционные коды, доступные для понимания калькулятору, описаны ниже с указанием их функций. В каждом случае при изменении стека калькулятора используется пять байтов, по пять байтов на каждый выполненный операционный код.

X - значение ниже Y на стеке калькулятора, Y - последнее помещенное на стек число. Результат вычислений всегда находится на вершине стека калькулятор и этот результат равен Z. Например операционный код вычитания (03), X=5, Y=9, X будет размещено на стеке калькулятора после Y, RST 40 (28h) будет сопровождена байтом 03. После команды стек калькулятора содержит -5, ответ на вершине в Z, поэтому X и Y вычеркиваются.

Переходы делаются командой перехода обычным образом.

## ☐ ФУНКЦИИ, ВЫПОЛНЯЕМЫЕ ОПЕРАЦИОННЫМИ КОДАМИ

| | | | |
|---|---|---|---|
| 00 | JUMP TRUE | делает переход на расстояние, определенное "DEFB" сразу после кода, если Y не нуль | (STACK CH.-5) |
| 01 | EXCHANGE | меняет порядок расположения X и Y на стеке | (STACK CH. 0) |
| 02 | DELETE | убирает Y со стека. Z=X | (STACK CH.-5) |
| 03 | SUBTRACT | X - Y = Z | (STACK CH.-5) |
| 04 | MULTIPLY | X * Y = Z | (STACK CH.-5) |
| 05 | DIVIDE | X / Y = Z | (STACK CH.-5) |
| 06 | TO POWER | X \| Y = Z | (STACK CH.-5) |
| 07 | BINARY OR | X OR Y. Если=0, значит Z=X, иначе Z=1 | (STACK CH.-5) |
| 08 | BINARY AND X AND Y | И X и Y должны быть числами. Если Y=0, то Z=0, иначе Z=X. Для работы со строками есть отдельный операционный код 16 (10h) | (STACK CH.-5) |

Последовательность операционных кодов от 09 до 14 (0Eh) работает с численными значениями. Т.к. регистр «B» должен содержать код при непосредственном выполнении команды, код может быть использован только как первая команда после RST 40.

Рассмотрим второй набор операционных кодов для построчного сравнения; результат Z=1 - истина, Z=0 - ложь (STACK CH.-5).

09 Y<=X

10 (0Ah) Y>=X

11 (0Bh) Y<>X

12 (0Ch) Y>X

13 (0Dh) Y<X

14 (0Eh) Y=X

15 (0Fh) ADDITION X+Y=Z (STACK CH.-5)

В следующем наборе команд по крайней мере одно из значений X и Y должно содержать параметры строки. Подробности получения этих параметров и помещения их на стек даны ниже. Строковые параметры помечены знаком доллара.

| | | | | |
|---|---|---|---|---|
| 16 (10h) | $AND NO | X$ и Y | Если Y=, то Z$ будет пустой строкой, иначе Z$=X$ | (STACK CH.-5) |

Операционные коды 17-22 (11h-16h) - строковые эквиваленты кодов 09-14 (0Eh), вызываемые операционным кодом, содержащимся в регистре «B». Вновь Z=1 истина, Z=0 - ложь. (STACK CH.-5)

17 (11h) Y$<=X$

18 (12h) Y$>=X$

19 (13h) Y$<>X$

20 (14h) Y$>X$

21 (15h) Y$<X$

22 (16h) Y$=X$

**23 (17h) ADDITION**

X$+Y$=Z$. Две строки сцепляются в рабочей области и новые параметры засылаются в Z$. Помните, что если размер окна окажется недостаточным для копирования двух строк в расширенную рабочую область - генерируется ошибка БЕЙСИКа. (STACK CH.-5)

**24 (18h) VAL$**

VAL$ Y$=Z$. Так же как и в случае, приведенном выше, в рабочей области генерируется новая строка. Операционный код должен содержаться в регистре «B» во время выполнения.

Любые ошибки приведут к ошибке БЕЙСИКа. Эта программа используется интерпретатором БЕЙСИКа и обрабатывается всеми процедурами проверки синтаксиса. (STACK CH. 0)

**25 (19h) USR$**

Z=USR$. Опять таки используется интерпретатором БЕЙСИКа и является предметом для проверки на синтаксис. Y$ должна содержать параметр строки, состоящей из одной буквы от «A» до "U". Z будет по завершении содержать адрес определенной пользователем графики. (STACK CH. 0)

**26 (1Ah) READ IN**

Эта подпрограмма позволяет разместить в рабочей области через любой поток (0-15 отдельный байт, взятый из Y. Байт интерпретируется как строка, а Z$ - параметры строки. Если флаг переноса не устанавливается, то никаких действий не предпринимается и Z$ возвращается в виде пустой строки.

**27 (1Bh) NEGATE**

Z=Y, но с обратным знаком. (STACK CH. 0)

**28 (1Ch) CODE**

Z=CODE Y$. Как используемая интерпретатором БЕЙСИКа. (STACK CH. 0)

**29 (1Dh) VAL**

Z=VAL Y$. Как используемая интерпретатором БЕЙСИКа и на самом деле позволяет после сканирования строки разместить результат в Z. Возможны ошибки БЕЙСИКа. (STACK CH. 0)

**30 (1Eh) LEN**

Z=LEN Y$. Это проще сделать без использования калькулятора, все что делается - это на стек помещаются байты длины параметров строки. (STACK CH. 0)

Все нижеприведенные алгебраические функции помещают результат (Z) на вершину стека калькулятора. Размер стека не меняется.

**31 (1Fh) Z=SIN Y**

**32 (20h) Z=COS Y**

**33 (21h) Z=TAN Y**

**34 (22h) Z=ASN Y**

**35 (23h) Z=ACS Y**

**36 (24h) Z=ATN Y**

**37 (25h) Z=LN Y**

**38 (26h) Z=EXP Y**

**39 (27h) Z=INT Y**

**40 (28h) Z=SQR Y**

**41 (29h) Z=SGN Y**

**42 (2Ah) Z=ABS Y**

**43 (2Bh) PEEK**

Z=PEEK Y. Как используемая интерпретатором БЕЙСИКа. (STACK CH. 0)

**44 (2Ch) IN Z=IN Y.**

Выполняет команду "IN A,(C)" Z80 после того, как возьмет Y в пару регистров "BC" как целое число. (STACK CH. 0)

**45 (2Dh) USR NO**

Нужна осторожность, т.к. команда приводит к переходу по адресу в Y. Используется БЕЙСИКом для перехода в машинные коды. Адрес возврата 11563 (2D2Bh), программа помещения на стек содержимого "BC", которая загрузит число в регистр "BC" при возвращении на стек калькулятора. Это интересный операционный код, т.к. он открывает нам возможность использования программ ПЗУ и ОЗУ рекурсивно, из калькулятора. (STACK CH. 0)

**46 (2Eh) STR$**

Z$=Y. Верхнее значение со стека калькулятора выводится в "WORKSPACE" и расценивается как строка, параметры которой затем помещаются на стек калькулятора. (STACK CH. 0)

**47 (2Fh) CHR$**

Z$=CHR$ Y. Если 0 Y 255, то в "WORKSPACE" формируется один пробел и значение переводится там в однобайтовый

вид. Затем оно интерпретируется как строка и параметры возвращаются в Z$. (STACK CH. 0)

**48 (30h) NOT**

Z=1, если Y=0, иначе Z=0. (STACK CH. 0)

**49 (31h) DUPLICATE**

Z=0. Y дублируется. (STACK CH.+5)

**50 (32h) X MOD Y**

При возврате Z=INT(X/Y). Округление идет в меньшую сторону (где исходно было X) X-INT(X/Y).. (STACK CH. 0)

**51 (33h) JUMP**

Это эквивалентно команде JR Z80 (JUMP RELATIVE), но для калькулятора. Длина перехода берется из ячейки после операционного кода. Стек калькулятора не задействован. (STACK CH. 0)

**52 (34H) STK DATA**

Позволяет считать значение из ячеек, следующих за операционным кодом. Биты 6 и 7 первого байта (показатель) после операционного кода, определяют количество байтов, которые необходимы для формирования мантиссы, от 00 BIN для 1 и до 11 BIN для 4. Число с плавающей точкой затем считывается в стек калькулятора. (STACK CH.+5)

**53 (35h) DEC JR NZ**

Прямой эквивалент операционному коду "DJNZ" Z80, но для калькулятора. Регистр «B» берется как системная переменная "BREG". Команда может быть использована несамостоятельно, т.к. интенсивно используется калькулятором для своих целей, следовательно значение "BREG" сомнительно. (STACK CH. 0)

**54 (36h) Y**

Z=1, если Y, иначе Z=0. (STACK CH. 0)

**55 (37h) Y0**

Z=1, если Y0, иначе Z=0. (STACK CH. 0)

**56 (38h) ENDCALC**

Передает управление Z80 к следующему адресу. (STACK CH. 0)

**57 (39h) GET ARGT**

Внутренняя программа калькулятора для задания значения Y для SIN Y или COS Y. (STACK CH. 0)

**58 (3Ah) TRUNCATE**

Z=INT Y, где Z - это Y, округляется в сторону 0. (STACK CH. 0)

**59 (3Bh) FP CALC2**

Используется интерпретатором для выполнения одной команды калькулятора. Для целей данной книги практического применения не имеет.

**60 (3Ch) E TO F.P.**

Z=Y E (показатель) регистр «A». (STACK CH. 0)

**61 (3Dh) RE STACK**

Z= положение плавающей точки в Y, где Y может быть маленьким целым числом. (STACK CH. 0)

**62 (3Eh) SERIES**

Используется калькулятором для генерации полинома Чебышева. Практической ценности для нас не имеет. Вызывается программой, использующей значения полинома.

**63 (3Fh) STACK NO**

Как и предыдущее, но используется для констант.

**64 (40h) ST MEM**

Используется для размещения числа в памяти. На входе регистр «A» должен содержать "C0-C5", в соответствии с которыми пять ячеек памяти будут использованы. (STACK CH.-5)

**65 (41h) REC MEM**

Получение числа из памяти. Процедура, обратная предыдущей. (STACK CH.-5)

Для операций со строками параметры могут быть размещены на стеке калькулятора с помощью программы 10934 (2AB6h), описанной выше. Пара регистров "BC" содержит длину строки, пара регистров "DE" - адрес начала, регистр «A» - имя (в форме, описанной в главе 2, команды "SAVE", "LOAD" и "VERIFY"). Для того, чтобы помочь разобраться в работе калькулятора и поэкспериментировать, в приложении «Полезные программы» дан соответствующий текст.

# ПРИЛОЖЕНИЕ **A**

## ☐ Десятично-шестнадцатеричные преобразования

**MSB**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 256 | 512 | 768 | 1024 | 1280 | 1536 | 1792 |
| 1 | 4096 | 4352 | 4608 | 4864 | 5120 | 5376 | 5632 | 5888 |
| 2 | 8192 | 8448 | 8704 | 8960 | 9216 | 9472 | 9728 | 9984 |
| 3 | 12288 | 12544 | 12800 | 13056 | 13312 | 13568 | 13824 | 14080 |
| 4 | 16384 | 16640 | 16896 | 17152 | 17408 | 17664 | 17920 | 18176 |
| 5 | 20480 | 20736 | 20992 | 21248 | 21504 | 21760 | 22016 | 22272 |
| 6 | 24576 | 24832 | 25088 | 25344 | 25600 | 25856 | 26112 | 26368 |
| 7 | 28672 | 28928 | 29184 | 29440 | 29696 | 29952 | 30208 | 30464 |
| 8 | 32768 | 33024 | 33280 | 33536 | 33792 | 34048 | 34304 | 34560 |
| 9 | 36864 | 37120 | 37376 | 37632 | 37888 | 38144 | 38400 | 38656 |
| A | 40960 | 41216 | 41472 | 41728 | 41984 | 42240 | 42496 | 42752 |
| B | 45056 | 45312 | 45568 | 45824 | 46080 | 46336 | 46592 | 46848 |
| C | 49152 | 49408 | 49664 | 49920 | 50176 | 50432 | 50688 | 50944 |
| D | 53248 | 53504 | 53760 | 54016 | 54272 | 54528 | 54784 | 55040 |
| E | 57344 | 57600 | 57856 | 58112 | 58368 | 58624 | 58880 | 59136 |
| F | 61440 | 61696 | 61952 | 62208 | 62464 | 62720 | 62976 | 63232 |

| | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|
| 0 | 2048 | 2304 | 2560 | 2816 | 3072 | 3328 | 3584 | 3840 |
| 1 | 6144 | 6400 | 6656 | 6912 | 7168 | 7424 | 7680 | 7936 |
| 2 | 10240 | 10496 | 10752 | 11008 | 11264 | 11520 | 11776 | 12032 |
| 3 | 14336 | 14592 | 14848 | 15104 | 15360 | 15616 | 15872 | 16128 |
| 4 | 18432 | 18688 | 18944 | 19200 | 19456 | 19712 | 19968 | 20224 |
| 5 | 22528 | 22784 | 23040 | 23296 | 23552 | 23808 | 24064 | 24320 |
| 6 | 26624 | 26880 | 27136 | 27392 | 27648 | 27904 | 28160 | 28416 |
| 7 | 30720 | 30976 | 31232 | 31488 | 31744 | 32000 | 32256 | 32512 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 | 34816 | 35072 | 35328 | 35584 | 35840 | 36096 | 36352 | 36608 |
| 9 | 38912 | 39168 | 39424 | 39680 | 39936 | 40192 | 40448 | 40704 |
| A | 43008 | 43264 | 43520 | 43776 | 44032 | 44288 | 44544 | 44800 |
| B | 47104 | 47360 | 47616 | 47872 | 48128 | 48384 | 48640 | 48896 |
| C | 51200 | 51456 | 51712 | 51968 | 52224 | 52480 | 52736 | 52992 |
| D | 55296 | 55552 | 55808 | 56064 | 56320 | 56576 | 56832 | 57088 |
| E | 59392 | 59648 | 59904 | 60160 | 60416 | 60672 | 60928 | 61184 |
| F | 63488 | 63744 | 64000 | 64256 | 64512 | 64768 | 65024 | 65280 |

**LSB**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 2 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 3 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 4 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 5 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 6 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| 7 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| 8 | 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 | 142 | 143 |
| 9 | 144 | 145 | 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 |
| A | 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 | 173 | 174 | 175 |
| B | 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 190 | 191 |
| C | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 |
| D | 208 | 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 |
| E | 224 | 225 | 226 | 227 | 228 | 229 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 |
| F | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 255 |

**NIBBLES**

| HEX | DEC | BIN | HEX | DEC | BIN |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0000 | 8 | 8 | 1000 |
| 1 | 1 | 0001 | 9 | 9 | 1001 |
| 2 | 2 | 0010 | A | 10 | 1010 |
| 3 | 3 | 0011 | B | 11 | 1011 |
| 4 | 4 | 0100 | C | 12 | 1100 |
| 5 | 5 | 0101 | D | 13 | 1101 |
| 6 | 6 | 0110 | E | 14 | 1110 |
| 7 | 7 | 0111 | F | 15 | 1111 |

# ПРИЛОЖЕНИЕ **В**

## ☐ Схема памяти СПЕКТРУМа

| |
|---|
| P RAMT FFFFh (65535) или 7FFFh (33767) |
| (23675) UDO |
| (23730) RAMTOP |
| (23653) END |
| стек кальк. (23651) BOT |
| рабоч. обл. (23649) WORKSP |
| обл. редакт. (23641)E LINE |
| переменные (23627)VARS |
| Бейсик (23635) PROG |
| информация о каналах (23631) CHANS схемы микродрайва |
| 5CEFh (23731) последний байт 8K системных переменных |
| 5CB6h (23733) последний байт 16K системных переменных |
| 5BFFh (23551) последний байт буфера принтера |
| 5AFFh (23295) последний байт атрибутов экрана |
| 57FFh( 22527) последний байт экрана |
| 3FFFh (16383) последний байт 16K ПЗУ |

# ПРИЛОЖЕНИЕ **C**

## ☐ Схема экрана СПЕКТРУМа

Экран **СПЕКТРУМа** разделен на **192** ряда по **256** пикселей. Каждый ряд занимает **32 байта** памяти в порядке возрастания. Ниже приведены адреса первых байтов каждого ряда экрана с адресом атрибута для первого байта. Первый ряд - верхний на экране.

| РЯД | ПИКСЕЛЬ | АТРИБУТ | РЯД | ПИКСЕЛЬ | АТРИБУТ |
|---|---|---|---|---|---|
| 1 | 4000h | 5800h | 49 | 40C0h | 5800h |
| 2 | 4100h | | 50 | 41C0h | |
| 3 | 4200h | | 51 | 42C0h | |
| 4 | 4300h | | 52 | 43C0h | |
| 5 | 4400h | | 53 | 44C0h | |
| 6 | 4500h | | 54 | 45C0h | |
| 7 | 4600h | | 55 | 46C0h | |
| 8 | 4700h | | 56 | 47C0h | |
| 9 | 4020h | 5820h | 57 | 40E0h | 58E0h |
| 10 | 4120h | | 58 | 41E0h | |
| 11 | 4220h | | 59 | 42E0h | |
| 12 | 4320h | | 60 | 43E0h | |
| 13 | 4420h | | 61 | 44E0h | |
| 14 | 4520h | | 62 | 45E0h | |
| РЯД | ПИКСЕЛЬ | АТРИБУТ | РЯД | ПИКСЕЛЬ | АТРИБУТ |
| 15 | 4620h | | 63 | 46E0h | |
| 16 | 4720h | | 64 | 47E0h | |
| 17 | 4040h | 5840h | 65 | 4800h | 5900h |
| 18 | 4140h | | 66 | 4900h | |
| 19 | 4240h | | 67 | 4A00h | |
| 20 | 4340h | | 68 | 4B00h | |
| 21 | 4440h | | 69 | 4C00h | |
| 22 | 4540h | | 70 | 4D00h | |
| 23 | 4640h | | 71 | 4E00h | |
| 24 | 4740h | | 72 | 4F00h | |
| 25 | 4060h | 5860h | 73 | 4820h | 5920h |
| 26 | 4160h | | 74 | 4920h | |
| 27 | 4260h | | 75 | 4A20h | |
| 28 | 4360h | | 76 | 4B20h | |
| 29 | 4460h | | 77 | 4C20h | |
| 30 | 4560h | | 78 | 4D20h | |

| | | | | | |
|---|---|---|---|---|---|
| 31 | 4660h | | 79 | 4E20h | |
| 32 | 4760h | | 80 | 4F20h | |
| 33 | 4080h | 5880h | 81 | 4840h | 5940h |
| 34 | 4180h | | 82 | 4940h | |
| 35 | 4280h | | 83 | 4A40h | |
| 36 | 4380h | | 84 | 4B40h | |
| 37 | 4480h | | 85 | 4C40h | |
| 38 | 4580h | | 86 | 4D40h | |
| 39 | 4680h | | 87 | 4E40h | |
| 40 | 4780h | | 88 | 4F40h | |
| 41 | 40A0h | 58A0h | 89 | 4860h | 5960h |
| 42 | 41A0h | | 90 | 4960h | |
| 43 | 42A0h | | 91 | 4A60h | |

| РЯД | ПИКСЕЛЬ АТРИБУТ | | РЯД | ПИКСЕЛЬ АТРИБУТ | |
|---|---|---|---|---|---|
| 44 | 43A0h | | 92 | 4B60h | |
| 45 | 44A0h | | 93 | 4C60h | |
| 46 | 45A0h | | 94 | 4D60h | |
| 47 | 46A0h | | 95 | 4E60h | |
| 48 | 47A0h | | 96 | 4F60h | |
| 97 | 4880h | 5980h | 145 | 5040h | 5A40h |
| 98 | 4980h | | 146 | 5140h | |
| 99 | 4A80h | | 147 | 5240h | |
| 100 | 4B80h | | 148 | 5340h | |
| 101 | 4C80h | | 149 | 5440h | |
| 102 | 4D80h | | 150 | 5540h | |
| 103 | 4E80h | | 151 | 5640h | |
| 104 | 4F80h | | 152 | 5740h | |
| 105 | 48A0h | 59A0h | 153 | 5060h | 5A60h |
| 106 | 49A0h | | 154 | 5160h | |
| 107 | 4AA0h | | 155 | 5260h | |
| 108 | 4BA0h | | 156 | 5360h | |
| 109 | 4CA0h | | 157 | 5460h | |
| 110 | 4DA0h | | 158 | 5560h | |
| 111 | 4EA0h | | 159 | 5660h | |
| 112 | 4FA0h | | 160 | 5760h | |
| 113 | 48C0h | 59C0h | 161 | 5080h | 5A80h |
| 114 | 49C0h | | 162 | 5180h | |
| 115 | 4AC0h | | 163 | 5280h | |
| 116 | 4BC0h | | 164 | 5380h | |
| 117 | 4CC0h | | 165 | 5480h | |

| | | | | | |
|---|---|---|---|---|---|
| 118 | 4DC0h | | 166 | 5580h | |
| 119 | 4EC0h | | 167 | 5680h | |
| 120 | 4FC0h | | 168 | 5780h | |
| 121 | 48E0h | 59E0h | 169 | 50A0h | 5AA0h |
| 122 | 49E0h | | 170 | 51A0h | |
| 123 | 4AE0h | | 171 | 52A0h | |
| 124 | 4BE0h | | 172 | 53A0h | |
| 125 | 4CE0h | | 173 | 54A0h | |
| 126 | 4DE0h | | 174 | 55A0h | |
| 127 | 4EE0h | | 175 | 56A0h | |
| 128 | 4FE0h | | 176 | 57A0h | |
| 129 | 5000h | 5A00h | 177 | 50C0h | 5AC0h |
| 130 | 5100h | | 178 | 51C0h | |
| 131 | 5200h | | 179 | 52C0h | |

| РЯД | ПИКСЕЛЬ АТРИБУТ | | РЯД | ПИКСЕЛЬ АТРИБУТ | |
|---|---|---|---|---|---|
| 132 | 5300h | | 180 | 53C0h | |
| 133 | 5400h | | 181 | 54C0h | |
| 134 | 5500h | | 182 | 55C0h | |
| 135 | 5600h | | 183 | 56C0h | |
| 136 | 5700h | | 184 | 57C0h | |
| 137 | 5020h | 5A20h | 185 | 50E0h | 5AE0h |
| 138 | 5120h | | 186 | 51E0h | |
| 139 | 5220h | | 187 | 52E0h | |
| 140 | 5320h | | 188 | 53E0h | |
| 141 | 5420h | | 189 | 54E0h | |
| 142 | 5520h | | 190 | 55E0h | |
| 143 | 5620h | | 191 | 56E0h | |
| 144 | 5720h | | 192 | 57E0h | |

# ПРИЛОЖЕНИЕ **D**

## ☐ Схема клавиатуры СПЕКТРУМа

| | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|
| 245 (FEh) | V | C | X | Z | C/S |
| 253 (FDh) | G | F | D | S | A |
| 251 (FBh) | T | R | E | W | Q |

| | | | | | |
|---|---|---|---|---|---|
| 247 (F7h) | 5 | 4 | 3 | 2 | 1 |
| 239 (EFh) | 6 | 7 | 8 | 9 | 0 |
| 223 (DFh) | Y | U | I | O | P |
| 191 (BFh) | H | J | K | L | ENT |
| 127 (7Eh) | B | N | N | S/S | B/S |

# ПРИЛОЖЕНИЕ **E**

## ☐ УПРАВЛЯЮЩИЕ КОДЫ СПЕКТРУМа

0 и 1 нет. Используется только после "INK", "PAPER", "OVER", "INVERSE", "FLASH", "BRIGHT", "AT" или "TAB".

2-5 нет. Используется только после "INK", "PAPER", "AT" или "TAB".

"6". Выполняет табуляцию в первую позицию половины строки.

7 EDIT. Код, создаваемый программой ввода клавиатуры при нажатии "CAPS SHIFT" и "1". Печатаемого симво́ла не имеет. Часто "BELL" код на принтерах и терминалах. 8 BACKSPACE Код, создаваемый программой ввода клавиатуры при нажатии "CAPS SHIFT" и "5". Передвигает курсор влево. Воспринимается большинством принтеров, т.к. это ASCII код перехода назад. 9 RIGHTSPACE Код, создаваемый программой ввода клавиатуры при нажатии "CAPS SHIFT" и

"8". Код перемещения курсора вправо, но при его использовании позиция печати не меняется. ASCII код табулированной печати строки.

"10" DOWNSPACE. Как и вышеприведенное, но для "CAPS SHIFT" и "6". ASCII код заполнения строки.

"11" UPSPACE. Как и вышеприведенное, но для "CAPS SHIFT" и "7". ASCII код для смещения печатаемой позиции вверх.

"12" DELETE. Как и вышеприведенное, но для "CAPS SHIFT" и "0". ASCII код заполнения формата.

"13" ENTER. Генерируется при нажатии клавиши "ENTER", выполняет возврат каретки и заканчивает печать строки при выводе. Это также ASCII код перевода каретки.

"14". Предшествует номеру строки в программе на БЕЙСИКе. На практике не применяется. ASCII код "SHIFT OUT".

"15". СПЕКТРУМом не используется. ASCII код "SHIFT IN".

"16". Код контроля "INK". Используется перед кодом, содержащим численное значение цвета "INK". Например для изменения цвета печати на экран всех символов на красный вам нужно использовать программу RST 16 с 16 в регистре «A» и двойкой. Отметьте, не ASCII код 2 а значение 2. Это показано в программе DEBASE.

"17". Как и вышеприведенное, но для "PAPER".

"18". Как и вышеприведенное, но для "FLASH", код может быть 0 или 1.

"19". Как и вышеприведенное, но для "BRIGHT".

"20". Как и вышеприведенное, но для "INVERSE".

"21". Как и вышеприведенное, но для "OVER".

"22". Код контроля "AT", должен сопровождаться номерами строки и столбца.

"23". Как и вышеприведенное, но для "TAB", требуется только значение столбца.

Коды с 24 по 31 СПЕКТРУМом не используются, но код 27 используется периферийными устройствами. Он сопровождается буквой, индицирующей требуемое действие. Хотя код "CHR$27" был стандартизирован, сопровождающие его коды стандартизированы не были. Остальные коды - это представление символов, коды 32-126 соответствуют ASCII кодам, приведенным в справочнике СПЕКТРУМа. Код 127 - знак СПЕКТРУМа - будьте осторожны, так как это ASCII код вычеркивания.

## ПРИЛОЖЕНИЕ F

### ☐ Указатели прерываний ПЗУ

| | 16K ПЗУ Mk 2 СПЕКТРУМ | 8K ПЗУ Mk 1 Интерфейс |
|---|---|---|
| I=0 | 20430 | 23755 |

| | | |
|---|---|---|
| I=1 | 52818 | 8501 |
| I=2 | 22269 | 25888 |
| I=3 | 39020 | 4196 |
| I=4 | 10419 | 52486 |
| I=5 | 2294 | 6701 |
| I=6 | 29149 | 51711 |
| I=7 | 16039 | 32459 |
| I=8 | 2088 | 14353 |
| I=9 | 65129 | 58170 |
| I=10 | 32802 | 1200 |
| I=11 | 58888 | 2039 |
| I=12 | 53183 | 59861 |
| I=13 | 52503 | 8205 |
| I=14 | 14367 | 49921 |
| I=15 | 27928 | 58884 |
| I=16 | 51984 | 32724 |
| I=17 | 82729 | 32477 |
| I=18 | 52481 | 4831 |
| I=19 | 49749 | 21965 |
| I=20 | 25705 | 56790 |
| I=21 | 51673 | 45182 |
| I=22 | 51568 | 65535 |
| I=23 | 12493 | 57851 |
| I=24 | 15582 | 61847 |
| I=25 | 23842 | 60952 |
| I=26 | 1382 | 4 52189 |
| I=27 | 7306 | 16129 |
| I=28 | 49947 | 6400 |
| I=29 | 2344 | 4870 |
| I=30 | 26573 | 65535 |
| I=31 | 3360 | 57855 |

| | | |
|---|---|---|
| I=32 | 52513 | 23755 |
| I=33 | 33485 | 8501 |
| I=34 | 544 | 25888 |
| I=35 | 49537 | 4196 |
| I=36 | 8527 | 52486 |
| I=37 | 23670 | 6701 |
| I=38 | 20444 | 51711 |
| I=39 | 288 | 32459 |
| I=40 | 32348 | 14353 |
| I=41 | 58154 | 58170 |
| I=42 | 19754 | 1200 |
| I=43 | 23653 | 2039 |
| I=44 | 7117 | 59861 |
| I=45 | 5578 | 1 8205 |
| I=46 | 23713 | 49921 |
| I=47 | 4569 | 58884 |
| I=48 | 60208 | 32742 |
| I=49 | 57640 | 32477 |
| I=50 | 13627 | 4831 |
| I=51 | 13256 | 21965 |
| I=52 | 1560 | 56790 |
| I=53 | 57124 | 45182 |
| I=54 | 34307 | 65535 |
| I=55 | 41231 | 57851 |
| I=56 | 65535 | 61947 |
| I=57 | 65535 | 60952 |
| I=58 | 65535 | 52189 |
| I=59 | 65535 | 16129 |
| I=60 | 255 | 6400 |
| I=61 | 0 | 4870 |
| I=62 | 255 | 65535 |

| | | |
|---|---|---|
| I=63 | 60 | 255 |

# ПРИЛОЖЕНИЕ **G**

## ☐ Подпрограмма калькулятора

Чтобы помочь вам понять работу калькулятора, научится им пользоваться и поэкспериментировать, я привожу следующие программы.

Первая программа - **демонстрационная.**

```
EXX ; Вы должны всегда сохранять H'L' для ; правильного возврата
PUSH HL
EXX
CALL PRSTK ; напечатаем начало и конец стека
LD BC,X ; первое число
PUSH BC ; сохраним его
CALL 2D2Bh ; поместим на стек X
POP BC ; восстановим X
CALL 2D2Bh ; вновь поместим X на стек
CALL 2DE3h ; напечатаем X
LD 2,20h
RST 10h ; напечатаем пробел
LD HL,NUMBER ; базовый адрес числа в ASCII форме
CALL STKNUM ; поместим число на стек (это будет Y)
RST 28h ; вызовем калькулятор
DEFB 31h ; продублируем Y
DEFB 38h ; выйдем из режима калькулятора
CALL 2DE3h ; напечатаем Y
RST 28h ; войдем в режим калькулятора
Здесь вы можете разместить набор определенных байтов (DEFB), чтобы поэкспериментировать с калькулятором.
DEFB 38h ; конец режима калькулятора
CALL 2DE3h ; печать результата
CALL PRSTK ; печать "STKBOT" и "STKEND" для
; проверки, находился ли стек в балансе
EXX ; восстановим H'L'
POP HL
EXX
```

```
RET
NUMBER DEFM "1234.567"
DEFB 13 ; здесь всегда должно быть DEFB 13, для
; сигнализации об окончании числа.
```

Нижеприведенная программа напечатает адреса начала и конца стека калькулятора:

```
PRSTK LD BC,(23651) ; это STKBOT
CALL 2D28h ; разместим ее на стек
LD A,"B"
RST 10h
LD A," "
RST 10h
CALL 2DE3h ; напечатаем STKBOT
LD A," "
RST 10h
LD A,"T"
RST 10h
LD A," "
RST 10h
LD BC,(23653) ; это STKEND
CALL 2D2Bh
CALL 2DE3h ; напечатаем STREND
LD A,0Dh
RST 10h
RET
```

А эта программа разместит на стеке число в ASCII форме. При ее использовании в паре "HL" разместите начальный адрес числа, помещаемого на стек.

```
STKNUM LD DE,(23645) ; это CH_ADD
PUSH DE ; сохраним
LD (23645),HL ; поместим в CH_ADD начало числа
LD A,(HL) ; поместим первый символ в «A»
CALL 2C9Bh ; поместим число на стек
POP DE ; восстановим CH_ADD
LD (23645),DE
RET
```

*Драйвы интерфейсов "MOREX" и "KEMPSTON"*

При входе в эту подпрограмму ASCII код, выводимый на линию "CENTRONICS", должен содержаться в регистре «A». Расширенные

коды не выводятся; если это требуется см. «Расширение символов для вывода» глава 2.

### ☐ Программа вывода "CENTRONICS" интерфейса "MOREX"

```
PUSH AF ; сохраним код символа
BUSY IN A,(0FBh) ; считаем линию занятости принтера
AND 1 ; проверим бит 0
JR NZ,BUSY ; если не сброшен - принтер занят
POP AF ; восстановим символ
OUT (0FBh),A ; перешлем его
LD A,1 ; стробируем принтер
OUT (7Fh),A
XOR A
OUT (7Fh),A
RET ; конец программы
```

Программа вывода "CENTRONICS" интерфейса "KEMPSTON"

```
PUSH BC ; сохраним пару регистров "BC", т.к. они
; используются для ввода/вывода
PUSH AF ; сохраним код символа
LD BC,0E2BFh ; порт занятости
BUSY IN A,(C) ; считываем линию занятости
RRA ; бит 0 для переноса
JR C,BUSY ; если не сброшен принтер занят
POP AF ; восстановим символ
DEC B ; изменим порт на 02BFh
DEC B
OUT (C),A ; перешлем символ
LD A,00Eh ; стробируем данные для пересылки
LD B,0E3h ; изменим порт на E3BFh, стробируем порт
OUT (C),A
INC A
OUT (C),A
POP BC ; восстановим "BC"
RET ; конец
```

Подпрограмма передвижения спрайта с помощью прерываний.

```
; Демонстрационная программа для спрайтов, см. главу 6
;
```

```
    ORG 51455
    ENT 51455
    ;
    ; 51455 содержит указатель прерываний
    ;
    DEFW 51500
    SETUP LD A,200
    LD I,A
    LD BC,(COORD)
    CALL PLOT
    IM 2
    RET
    ;
    ; это важная программа, делающая указатель таким, чтобы
    ; программа прерываний стартовала с нужного адреса (здесь
    51500)
    ;
    ORG 51500
    ;
    ; сохраним все используемые регистры
    ;
    PUSH HL
    PUSH BC
    PUSH DE
    PUSH AF
    ;
    ; сохраним TV флаг, т.к. он может быть изменен
    ;
    LD A,(23612)
    PUSH AF
    ;
    ; сохраним CO_ORDS, т.к. они будут изменены, перед воз-
    вратом их
    ; нужно восстановить
    ;
    LD HL,(23677)
    PUSH HL
    LD BC,(COORD) ; координаты спрайта
    PUSH BC ; сохраним их
    ;
```

```
; вызываем программу вывода спрайта. По окончании каждого вывода
; это сотрет предыдущую позицию
;
CALL PLOT
POP BC ; восстановим координаты спрайта
;
; "L" используется как флаговый регистр для определения
; направления движения спрайта
; бит 0 установлен - вверх, сброшен - вниз
; бит 1 установлен - влево, сброшен - вправо
;
LD HL(FLAG)
BIT 0,L
JR Z,UP
DOWN DEC 8 ; передвижение спрайта вниз на пиксель
JR NZ,LEFT ; если флаг 0 - дошли до низа экрана и
; направление меняется
RES 0,L
JR LEFT
UP INC B ; один пиксель вверх
LD A,B ; дошли ли до верха экрана?
CP 174
JR NZ,LEFT
SET 0,L ; изменили направление
LEFT BIT 1,L ; те же проверки для влево/вправо
JR Z,RIGHT
DEC C
JR NZ,CPLOT
RES 1,L
JR CPLOT
RIGHT INC C
LD A,C
CP 254
JR NZ,CPLOT
SET 1,L
;
; новые координаты для спрайта записаны
;
CPLOT LD (COORD),BC ; теперь сохраним флаги
```

```
LD (FLAG),HL
;
CALL PLOT ; вывод новой позиции спрайта
;
; вывод закончен, восстановим начальные значения систем-
ных
; переменных
;
NRET POP HL
LD (23677),HL
POP AF
LD (23612),A
;
; восстановим регистры
POP AF
POP DE
POP BC
POP HL
;
; сканируем клавиатуру, т.к. этого не делалось потому, что
; программа прерываний была перемещена в ОЗУ
;
RST #38
RETI
COORD DEFW 0
FLAG DEFW 0
;
; "D" используется как счетчик, т.к. «B» уже использован
;
PLOT LD D,4
;
; запомним координаты, т.к. они будут испорчены програм-
мами ПЗУ
;
PUSH BC
LOOP LD A,D
PUSH DE ; сохраним счетчик
;
; вывод делается четырежды, каждый раз выдавая следую-
щий спрайт
```

```
;
CP 4
JR Z,LOOP4
CP 3
JR NZ,LOOP1
INC C
LOOP1 CP 2
JR NZ,LOOP2
INC B
LOOP2 CP 1
JR NZ,LOOP4
DEC C
;
; проверим, какая ПЗУ подключена
;
LOOP4 LD A,(#14) ; запросим ячейку, отличающуюся в раз-
ных
: ПЗУ
; Устанавливаем вывод на основной экран
RES 0,(IY+2) ; IY+2 - TV FLAG
PUSH BC ; сохраним "BC" для перехода ПЗУ-ПЗУ
CP #D5
JR Z,ROM2 ; подключена ПЗУ 2, поэтому подпрограммы
; основной ПЗУ впрямую вызвать нельзя
; если мы здесь, значит подключена основная ПЗУ и возмо-
жен
; прямой вызов
;
CALL #D4D ; запросим цвета для вывода
SET 0(IY+87) ; установим "OVER 1" через "P FLAG"
CALL 8933 ; программа из ПЗУ для вывода пикселя
JR CONT1 ; делаем непрямой переход к запросам 16K
; из 8K
;
; если мы здесь, то подключена ПЗУ интерфейса
ROM2 RST 16 ; не прямо вызывается 16K ПЗУ
DEFW #D4D
SET 0,(IY+87)
RST 16
CONT1 RES 0,(IY+87) ; сбросим "OVER 1"
```

```
POP BC ; восстановим последнюю позицию печати
POP DE ; и счетчик
DEC D
JR NZ,LOOP
;
POP BC ; восстановим начальную позицию вывода
LD A,B ; проверим, достигнут ли край
AND A
JR Z,PING ; если да, то выполняем необходимые
; действия
CP 174
JR Z,PING
LD A,C
AND A
JR Z,PING
CP 254
JR Z,PING
RET ; иначе - возврат
;
PING PUSH BC ; сохраним координаты вывода, чтобы «B»
; можно было использовать как счетчик
LD B,0
PINGL LD A,B ; выдадим тон на динамик
;
OUT (#FE),A ; меняем состояние, чтобы воспроизвести
; тон
DJNZ PINGL
POP BC
;
; теперь посмотрим, откуда пришел вызов
;
LD HL,NRET
LD A,L
POP HL
PUSH HL
XOR L
;
; мы в этой точке, только если возврат не был сделан и
следующая
; программа установит черный цвет бордюра
```

```
;
RET Z
;
; увеличим значение цвета бордюра, биты 0-2
;
LD A,(COL)
INC A
LD (COL),A
;
; зададим цвет бордюра
;
OUT (#FE),A
RET
COL DEFB 0
```

## ☐ DEBASE

DEBASE состоит из нескольких отдельных подпрограмм, собранных в грубую, но эффективную систему для работы с базами данных, есть программы для записи и загрузки как с ленты, так и с МИКРОДРАЙВа. Программа позволяет производить следующие опера́ции: вводить, печатать, находить, вычеркивать и изменять. Нет ограничений на число отдельных записей и размер каждой записи, за исключением одного требования: запись должна помещаться на экране без "Scroll". Может быть найдена любая часть любого текста, курсор будет помещен в начало найденного набора символов. Запись может быть изменена, дополнена или сокращена после первого ввода без уничтожения или изменения любых других частей. После этого освободившаяся память может быть использована. Программа использует 1 байт памяти на символ и 1 байт на запись для маркировки конца записи; 32 000 байт свободны для записи и есть место для дополнительных функций.

```
; Эта программа печатает все, что начинается с байта с
; активизированным битом 7 до следующего байта с таким битом 7.
; В ней есть три основные точки входа и два способа
; использования. Если регистр A содержит на входе 255, регистр
; DE должен содержать адрес первого символа сообщения,
; предваренного байтом с активизированным битом 7. Для всех
```

```
; других значений A напечатается сообщение из таблицы, идущее в
; ней под заданным номером.
;
;
MC1 ORG 62500
PRESS PUSH AF ; эта точка входа устанавливает начало
; печати в левую верхнюю позицию экрана
PUSH DE
LD A,2
CALL #1601 ; подтверждаем, что основной экран
; текущий, иначе позиция печати
; установится в другой экран, а в
; основном экране не изменится
LD BC,#1821
CALL 3545 ; используем ПЗУ, см. Главу 2
POP DE
POP AF
PRINAT PUSH AF ; эта точка входа сохраняет позицию
; печати и использует основной экран
PUSH DE
LD A,2
CALL #1601
LD A,255 ; задает SCROLL до сообщения "Scroll?"
LD (23692),A ; это системная переменная SCR ST.
LD A,13 ; перевод каретки в конец строки
RST 16 ; рестарт печати
POP DE
POP AF
PRINT INC A ; этот вход использует текущий поток
DEC DE ; поместим в "DE" начало маркера
JR Z,PRINTP ; если в «A» #FF используем адрес из DE
DEC A
LD DE,MESS-1 ; DE установлен в начало таблицы
; сообщений
PRINTP CALL 3082 ; программа ПЗУ для печати
RET
MESS EQU 1
; Эта программа ищет в памяти строку, начало которой находится в
```

```
; SCHAR. Завершается маркер-байтом, содержащим 255. Если
; соответствие найдено, то находится начало записи путем подъема
; вверх до байта с активизированным битом 7, который печатается
; предыдущей программой.
;
; Если конец каждой записи промаркирован #8D, то содержимое
; SCHAR будет #FF (ничего не нашли) вместо сообщений ничего
; напечатано не будет, так как ничего не найдено.
; На выходе регистры BC будут содержать либо адрес начала
; записи, содержащей строку, либо 0, если строка не была найдена
;
;
MC2 ORG #F450
FINDIT LD HL,SCHAR ; основная точка входа
LD A,(HL)
CP #FF
JR NZ,SEARCH
INC HL
LD (HL),A
DEC HL
LD A,#8D
LD (HL),A
SEARCH PUSH HL
LD HL,30002
LD BC,34000
POP DE
LOOKMO PUSH DE
LOOKON CPIR ; ищем первый символ, если строка не
; найдена
JR NZ,NOTFIND
DEC HL
LD (LOCFND),HL; сохраним ячейку, где нашли, и отсчитаем
; влево 80
LD (LOCCNT),BC
```

```
    POP DE
    PUSH DE
    MAYBE INC HL ; теперь проверяем посимвольно
    EX DE,HL
    INC HL
    LD A,(HL)
    CP 255 ; когда найден конец строки - строка
    ; найдена
    JR NZ,FOUND
    ;
    EX DE,HL
    CP (HL)
    JR Z,MAYBE
    LD HL,(LOCFND); если соответствия нет - снова ищем
    ; первый символ
    INC HL
    LD A,(SCHAR)
    JR LOOKON
    FOUND POP HL
    LD HL,(LOCFND); сначала проверим, в разрешенной ли
    ; области делается поиск
    BACK DEC HL
    LD (MEMPOS),HL
    INC HL
    PUSH HL
    LD DE,(LIMIT)
    AND A
    SBC HL,DE
    JR NC,NOTFND ; если нет - отменяем поиск
    POP HL
    BACKMO DEC HL ; ищем начало входа для печати
    BIT 7,(HL)
    JR Z,BACKMO
    INC HL
    PUSH HL
    PUSH HL
    CALL 5435 ; очистим весь экран
    POP DE
    LD A,#FF ; программа печати должна использовать
```

```
; адрес из DE
CALL PRESS
POP BC
RET
NOTFND LD A,3 ;печать сообщения 3
CALL PRINAT
POP HL
LD BC,0
RET
;
CONTLK LD HL,SCHAR ; эта точка входа для поиска других
; таких же строк
PUSH HL
LD A,(HL)
LD BC,(LOCCNT)
LD HL,(LOCFND)
INC HL
JP LOOKON
;
SCHAR DEFS 33
LOCCNT DEFW 0 ; после поиска отсчет влево
LOCFND DEFW 30002 ; ячейка где нашли
MEMPOS DEFW 30001 ; ячейка предваряющая найденную
LIMIT DEFW 62400 ; количество доступной памяти
PMESS EQU #F424 ; программа печати сообщений см. выше
в
; этом же приложении
PRINAT EQU #F423

; При использовании вместе с предыдущей программой вы-
черкивает
; запись
;
; На выходе регистры BC содержат количество освободив-
шейся
; памяти
;
MC3 ORG #F4CB
XREC CALL FINDIT ; используется для поиска начала
; вычеркиваемой записи
```

```
XRECON LD A,B
OR C
RET Z ; если не нашли BC=0
PUSH HL
PUSH BC
LD A,2 ; сообщение 2
CALL PRINAT
CHECKB LD A,127 ; это один из способов сканирования
; нажатия ключей, см. главу 2
IN A,(#FE)
RRA
JR NC,BREAK
LD A,127
IN A,(#FE)
BIT 3,A
JR Z,NEXTR
LD A,254
IN A,(#FE)
BIT 2,A
JR NZ,CHECKB
POP BC
POP HL
PUSH BC
PUSH BC ; BC - начало записи
POP HL
LD BC,0
F_END BIT 7,(HL)
INC HL
INC BC ; конец записи должен быть найден
JR Z,F_END
POP DE ; теперь все вплоть до LIMIT опускается
; и запись уничтожается
PUSH BC
PUSH HL
POP BC
PUSH HL
LD HL,(LIMIT)
AND A
SBC HL,BC
```

```
PUSH HL
POP BC
POP HL
LDIR
POP BC ; количество добавленной памяти
RET
BREAK POP HL ; очистим стек, скажем о ненахождении
POP BC
LD BC,0
RET
NEXTR POP HL ; восстановим старые детали и продолжим
; поиск
POP BC
CALL CONTLK
JP XRECON
;
; Эта программа позволяет разместить символ с клавиатуры
в памяти
; и отобразить его на экран.
;
; Она выполняет перемещение курсора вперед и назад, де-
лает
; вставки и вычеркивание символов
;
; Есть так же "HELP" и меню.
;
; Здесь показано как ее использовать вместе с другими
; программами для работы с базами данных, но она приме-
нима и для
; других целей.
; Программа проверяет доступную память и по достижении
предела
; останавливает ввод.
;
; Я включил вторую программу ввода и внешние программ
для выбора
; функций, чтобы продемонстрировать разные способы вво-
да и некоторые
; проблемы, упомянутые в основном тексте книги.
;
```

```
;
; Это адрес вызова из БЕЙСИКа.
;
MC4 ORG #F51D
USR LD A,#A0 ; сначала проверяется свободная память
LD HL,(LIMIT)
LD DE,(CHPOS)
AND A
SBC HL,DE
JP NC,NOTFUL
LD A,9 ; сообщение 9
CALL PRINAT
JP PMESS4
NOTFUL PUSH HL ; вся память выше последней записи и ниже
; "LIMIT" очищается и маркируется как
; свободная
POP BC
PUSH DE
POP HL
INC DE
LD (HL),A
LDIR
INPUTA LD HL,(CHPOS) ; начало следующей записи
DEC HL
LD A,(HL)
CP #8D
JR Z,ATBEG
INC HL
ATBEG LD (HL),#8D
LD (CUPOS),HL ; маркируется и очищается начало
CALL 3435
CALL OPEN1
LD BC,#1821 ; позиция печати в нижнем экране
; устанавливается влево - вверх
CALL 3545
LD A,7 ; сообщение 7
CALL PRINT
LD A,1 ; фирменный знак программы
CALL PRESS
```

```
XOR A ; очищается флаг режима редактирования
LD (WFLAG),A
LD BC,#1621 ; установим строку 2, колонку 0
CALL 3545
LD A,""
RST 16
CALL CURSOR
INPUT CALL OPEN1
INP1 CALL #10A8 ; это программа ввода с клавиатуры
JR C,KEYPRE ; перенос активизируется при вводе ключа
RES 3,(IY+2) ; это TV флаг, см. «Ожидание ввода»
; глава 2
JR INP1
KEYPRE CP 198 ; теперь запрашиваем специальные клави-
ши
JR Z,BACK1
CP 9
JP Z,RIGHT1
CP 12 ; CAPS SHIFT и 0
CP Z,DELETE
CP 197 ; OR
RET Z
CP 205 ; STEP
JP Z,HELP
CP 198
JP Z,AND
CP 195 ; NOT
JP Z,LOOKM
CP 204 ; то
JP Z,LPRINT
LD HL,(CUPOS)
INC HL ; проверяем, достигнут ли конец записи,
; если ввод больше не разрешен
PUSH AF
LD A,(WFLAG)
AND A
JR Z,OKTHEN ; если не в режиме редактора, то это
; свободная память
LD A,(HL)
CP #8D
```

```
JP Z,ENDREC
OKTHEN POP AF
CP 226 ; стоп, оканчивающий ввод
JR Z,STOP
LD (HL),A ; код символа помещается в текущую
; ячейку памяти, в следующий раз она
; будет изменена
LD (CUPOS),HL
CP 13
JR NZ,KEYOUT ; при выводе перемещаем курсор
CALL OPEN2
CALL CURSOR
KEYOUT CALL OPEN2
RST 16 ; отображение символов на экран
CALL CURSOR ; перемещаем курсор
JR INPUT ; повторяем для ввода следующего кода
;
;
BACK1 LD HL,(CUPOS) ; переводит курсор в текущую позиция в
; памяти назад
LD A,(HL)
BIT 7,A
JR NZ,INPUT
CP 13
DEC HL
LD (CUPOS),HL
JR NZ,B_ON
CALL BACK ; но возврат после ввода означает повтор
; печати и изменение режима
JR FINRET
B_ON CALL OPEN2
CALL CURSOR
LD A,0
RST 16
CALL CURSOR
JP INPUT ; возврат за следующим символом
;
;
STOP PUSH HL ; если нажали STOP при новом вводе, то
```

```
; необходимо маркировать конец записи
DEC HL
ENDIS LD A,(HL) ; мы всегда ищем начало свободной памяти
; и изменяем указатель
INC HL
CP #A0
JR NZ,ENDIS
DEC HL
LD (CHPOS),HL
POP HL
LD A,(WFLAG)
AND A
INC HL
JR NZ,STOPON
LD (CHPOS),HL
DEC HL
LD A,#8D
LD (HL),A
STOPON CALL 3435 ; очистим экран
XOR A ; очистим флаг режима
LD (WFLAG),A
LD A,4 ; выведем меню
CALL PRESS
MENUIN CALL OPEN1
CALL #15DE ; программа ожидания ввода, глава 2
CP 13
JP Z,USR
OR #20
CP "O"
JR Z,XINPUT
CP "F"
JR Z,FINPUT
CP "C"
JR Z,LOOKM
CP "S"
JR Z,SAVEL
JR MENUIN
;
;
```

```
XINPUT CALL FINPT ; начало программы вычеркивания
CALL XREC
LD HL,(CHPOS) ; свободная память увеличивается
; изменением указателя памяти на
; количество вычеркнутых байтов
AND A
SBC HL,BC
LD (CHPOS),HL
JP PMESS4
;
;
FINPUT CALL FINPT ; поиск начала записи
CALL FINDIT
FINRET LD A,B ; не нашли - выходим в меню
OR C
JP Z,PMESS4
DEC BC
LD (CUPOS),HL ; иначе - режим редактора, курсор под
; началом найденного
LD HL,(MEMPOS)
AND A
SBC HL,BC
LD (CURSP),HL
LD A,1
LD (WFLAG),A
CALL OPEN1
LD BC,#1821
CALL 3435
LD A,8
CALL PRINT
LD A,7
CALL PRINT
CALL OPEN2
LD BC,#1821
CALL 3545
LD A,13
RST 16
CALL CURSOR
LD BC,(CURSP)
```

```
R1MORE LD A,B
OR C
JP Z,INPUT
PUSH BC
CALL RIGHTS
POP BC
DEC BC
JR RIMORE
;
;
FINPT CALL 3435 ; похоже на предыдущую программу ввода,
; но не допускает вычеркиваний и
; ограничивает ввод, останавливаясь по
; достижении "IN_LIM"
LD A,5
CALL PRESS
LD A,62
RST 16
LD HL,SCHAR+32
LD (IN_LIM),HL
LD HL,SCHAR
;
;
INPUTF PUSH HL ; подпрограмма ввода
CALL OPEN1
CALL #15DE
POP HD
CP 13 ; прекращается нажатием ENTER
JR Z,SETFIN
CP 32
JR Z,INPUTF
LD (HL),A
INC HL
PUSH HL
CALL OPEN2
RST 16
POP HL
EX DE,HL
LD HL,(IN_LIM)
```

```
AND A
SBC HL,DE
EX DE,HL
JR NZ,INPUTF
SETFIN LD (HL),255
RET
;
;
LOOKM CALL CONTLK ; ищем, есть ли еще такие строки
JP FINRET
;
;
PMESS4 LD B,3 ; очистим нижние три строки экрана,
; смотри главу 2
CALL 3652
LD A,4 ; меню
CALL PRINAT
JP MENUIN
;
;
; OVER1 и OVER2 воспроизводят команды БЕЙСИКа, печатая
; управляющие символы
OVER1 PUSH AF
PUSH HL
LD A,21
RST 16
LD A,1
RST 16
POP HL
POP AF
RET
OVER0 PUSH AF
PUSH HL
LD A,21
RST 16
LD A,0
RST 16
POP HL
POP AF
```

```
RET
CURSOR PUSH AF ; печать курсора и передвижение позиции
; печати назад на 17 позиций
CALL OVER1
LD A,95
RST 16
LD A,8
RST 16
CALL OVER0
POP AF
RET
RIGHT1 CALL RIGHTS
JP INPUT
RIGHTS CALL OPEN2 ; передвижение позиции курсора на 1
; вправо изменяя указатель памяти для
; нового символа
LD HL,(CUPOS)
INC HL
LD A,(HL))
BIT 7,A
RET NZ
CP 127
RET Z
CP 13
JR Z,ONRITE
CP 32
RET C
ONRITE LD (CUPOS),HL
CALL CURSOR
CP 13
JR Z,RITEON
CALL OVER1
LD A,32
RITEON RST 16
CALL CURSOR
RET
;
;
ENOREC PUSH HL ; при попытке ввода после окончания
; записи выдается предупреждающее
```

```
; мигающее сообщение, низ экрана
; очищается
LD B,3
CALL 3652
LD A,6
CALL PRESS
EI ; если они были отключены
LD B,50 ; как всегда 1 секундная пауза
WAIT1 HALT
DJNZ WAIT1
POP HL
POP AF
REENTR LD HL,LOCFND
DEC (HL)
DEC HL
INC (HL)
JP LOOKM
;
;
DELETE LD HL,(CUPOS) ; работает как вычеркивание записи,
но
; для одного символа
INC HL
LD A,(HL)
BIT 7,A
JP NZ,INPUT
PUSH HL
PUSH HL
LD HL,(LIMIT)
POP DE
AND A
SBC HL,DE
PUSH HL
POP BC
POP DE
JP C,INPUT
JP Z,INPUT
PUSH DE
POP HL
INC HL
LDIR
```

```
LD HL,(CHPOS)
DEC HL
LD (CHPOS),HL
LD HL,(CUPOS)
INC HL
CALL BACK
JP FINRET
;
;
AND LD HL,(CUPOS) ; то же самое, но наоборот
INC HL
PUSH HL
LD HL,(LIMIT)
POP DE
AND A
SBC HL,DE
PUSH HL
POP BC
PUSH DE
LD DE,(LIMIT)
PUSH DE
POP HL
DEC HL
LDDR
LD HL,(CHPOS)
INC HL
LD (CHPOS),HL
POP HL
LD A,32
LD (HL),A
CALL BACK
JP FINRET
;
;
LPRINT LD HL,(CUPOS) ; использует поток 3 как поток 2 для
; экрана, если подключен интерфейс
; воспринимающий LPRINT
INC HL
CALL BACK
PUSH BC
```

```
LD A,3
CALL #1601
POP DE
PUSH DE
DEC DE
XOR A
CALL PRINTP
POP BC
JP FINRET
;
;HELP CALL 3435
LD A,10 ; страница HELP
CALL PRESS
WAITE LD A,00F
IN A,(0FE)
AND 1
JR NZ,WAITE
LD HL,(CUPOS)
LD (LOCFND),HL
INC HL
LD A,(HL)
LD HL,SCHAR
LD (HL),A
INC HL
LD (HL),0FF
JP REENTR
; начало программ записи и загрузки
;
; Сначала вычисляется длина записи, включая оглавление и мар-
кер
; окончания, затем байты длины для повторного ввода после
; загрузки
;
;
MC5 ORG 0F7F1
SAVEL LD HL,(CHPOS) ; сначала конец записей помещается в
; начальную область для записи или
; загрузки
LD (29998),HL
LD DE,29998
AND A
```

```
    SBC HL,DE
    INC HL
    INC HL
    PUSH HL
    LD A,11
    CALL PRESS
    LD B,25 ; 25 прерываний за 1/2 секунды если они
    ; были отключены
    EI
    ;
    ;
    ; Это некорректно делать, но продемонстрируем использование
HALT
    ; для хронометрирования
    ;
    HOLDIT HALT ; убедимся, что S получено не из
    ; основного меню, теперь сканируем
    ; клавиатуру
    DJNZ HOLDIT
    ;
    ;
    SOLDEC LD A,#DF
    IN A,(#FE)
    AND 2
    JP Z,LOAD
    LD A,#FD
    ; IN A,(#FE)
    AND 2
    JR NZ,SOLDEC
    ;
    ; программа выбора: лента или м/драйв
    ;
    ;
    ORG #F81C
    SAVE POP BC
    CALL M_OR_T ; если нажато M, делается возврат с
    ; установленным переносом и заголовком
    ;
    JR C,MSAVE
    ;
    ; для записи заголовка и основного блока данных на ленту.
```

```
; Заголовок особый. См. команды "SAVE" и "LOAD", глава 2.
;
TSAVE PUSH BC ; BC=длина основного блока IX всегда
; установлен в начало
LD DE,13 ; заголовок из 13 байтов
XOR A ; A - сигнальный заголовок
CALL 1222 ; программа ПЗУ, смотри главу 2 для
; записи заголовка
POP DE ; восстановим длину основного блока
LD IX,29998 ; IX указывает адрес, с которого начнем
; запись из сигнального основного блока
LD A,0FF
CALL 1222 ; используется программа ПЗУ, возврат с
; вновь отключенными прерываниями
EI
JP LORET
; ниже приведенная программа отключает 16K ПЗУ
; см. главу 3
;
;
ORG #FB38
RSWAP LD HL,ROMOUNT
LD (23789),HL ; программа ПЗУ использует обходной код
; 32, поэтому установите возврат к этой
; программе
EXX
LD (HLSAV),HL
EXX
LD (BCSAV),BC
RST 8 ; используем обходной код
DEFB #32
ROMOUNT POP HL ; два адреса программы должны быть
; перемещены и возврат из этой программы
; это "ERR SP", см. главу 4
POP HL
POP DE
LD HL,(23613)
PUSH HL ; сохраним позицию возврата по ошибке на
; стеке для дальнейшей работы
LD HL,MSLRET
```

```
PUSH HL ; поместим на стек новый адрес возврата
; по ошибке, укажем в "ERR SP" для
; восстановления адреса возврата
LD (23613),SP
PUSH DE
RET
; программа записи на МИКРОДРАЙВ
; ;
ORG #FB5A
MSAVE CALL RSWAP
CALL MHEAD ; зададим заголовок и системные
; переменные
CALL VARSET
SET 5,(IY+124) ; FLAGG 3 - запись
CALL #1E7F ; на самом деле вызов не нужен, т.к.
; возврат через ошибку 16K
MSLRET POP HL ; восстановим "ERR SP"
LD (23613),HL ; "ERR SP"
EI
EXX
LD HL,(HLSAV) ; регистры HL
EXX ; отметьте, что 16K ПЗУ подключена,
; потому что возврат через ошибку в
; БЕЙСИКе 16K ПЗУ
JP LORET
ORG #F877
VARSET LD HL,23769 ; подробно описано в главе 3
LD (HL),"M"
LD HL,10
LD (23770),HL
LD HL,SCHAR+1
LD (23772),HL
LD HL,1
LD (23766),HL
RET
;
;
; Программа загрузки с МИКРОДРАЙВа, детально изложена в
Главе 3
;
;
```

```
MLOAD LD BC,0 ; детали заголовка нужно использовать,
; т.к. они неизвестны
CALL RSWAP
CALL MHEAD
CALL VARSET
SET 4,(IY+124) ; сигнализирует о загрузке
CALL #BAF ; вызов программы загрузки
JP MSLRET ; реально это нежелательно
;
;
; зададим микродрайверный заголовок, см. Главу 3
;
;
MHEAD LD HL,23782
LD (HL),3
INC HL
LD DE,(BCSAV)
LD (HL),E
INC HL
LD (HL),D
INC HL
LD DE,29998
LD (HL),E
INC HL
LD (HL),D
INC HL
LD (HL),#80
RET
;
;
; определяем куда производить загрузку - на ленту или МИКРО-
ДРАЙВ
;
; загрузка с ленты смотри Главу 2
;
LOAD POP BC
CALL M_OR_T
JR C,MLOAD
TLOAD XOR A ; заголовок
SCF ; загрузка
LD DE,13
```

```
INC D
EX AF,A'F'
DEC D
DI
LD IX,SCHAR+15
CALL #562
EI
CALL 8020
RET NC ; нажат BREAK
TESTHE LD IX,SCHAR ; проверка имени заголовка
LD HL,SCHAR+15
LD B,11
TESTLP LD A,(HL)
CP (IX+00)
JR NZ,TLOAD ; несоответствие - заголовок неверен
INC IX ; соответствие - выполняем загрузку
INC HL
DJNZ TESTLP
LD IX,29998
LD E,(HL)
INC HL
LD D,(HL)
SCF ; загрузка
LD A,#FF ; основной блок данных
INC D
EX AF,A'F'
DEC D
DI
CALL #562
EI
CALL 8020
RET NC ; нажат BREAK
JP LORET ; загружено, повторный ввод
;
;
; программа запрашивает имя файла и что используется - лента
или
; МИКРОДРАЙВ
;
;
M_OR_T CALL SAVET
```

```
    PUSH BC
    PUSH IX
    LD A,12
    CALL PRINAT
    MIORTA LD A,#7F
    IN A,(#FE)
    AND 4
    SCF
    JR Z,MOTFIN
    LD A,#FB
    IN A,(#FE)
    AND 16
    JR NZ,MIORTA
    MOTFIN POP IX
    POP BC
    RET
    ;
    ;
    ; берет параметры длины из начала загруженной памяти или то,
что
    ; туда было помещено при записи
    ;
    ;
    LORET LD HL,(29998)
    LD (CHPOS),HL
    JP USR
    ;
    ;
    ; программа генерирует заголовок в области памяти SCHAR
    ;
    ;
    SAVET LD HL,SCHAR
    PUSH BC
    PUSH HL
    PUSH HL
    LD (HL),3
    PUSH BC
    LD B,10
    LD A,32
    SETHED INC HL
    LD (HL),A
```

```
    DJNZ SETHED
    PUSH HL
    LD (IN_LIM),HL
    POP HL
    POP BC
    INC HL
    LD (HL),C
    INC HL
    LD (HL),B
    CALL 3435
    LD A,13
    CALL PRESS
    POP HL
    INC HL
    CALL INPUTF
    LD (HL),32
    POP IX
    POP BC
    RET
    ;
    ;
    BCSAV DEFW 0
    CUPOS DEFW 0
    CHPOS DEFW 30001
    HLSAV DEFW 0
    IN_LIM DEFW 0
    LIMIT DEFW 62000
    LOCCNT DEFW 0
    LOCFND DEFW 0
    WFLAG DEFB 0
    MEMPOS DEFW 30001
    CURSP DEFW #1821
    SCHAR DEFS 33
    MESSM1 DEFB #80
    MESS DEFB #80
    MESS1 DEFM "DEBASE COPYRIGHTE SGK 1984"
    DEFB #8D
    MESS2 DEFW #110
    DEFM "PRESS SPACE TO ABORT"
    DEFB 13
```

```
DEFM "X TO ERASE OR N TO NEXT RECORD"
DEFW #10
DEFB #8D
MESS3 DEFW #210
DEFM "NOT FOUND"
DEFW #10
DEFB #8D
MESS4 DEFW #110
DEFM " MENU"
DEFW #0D0D
DEFM "PRESS"
DEFW #0D0D
DEFM "F TO FIND"
DEFW #0D0D
DEFM "E TO ERASE"
DEFW #0D0D
DEFM "C TO CONTINUE SEARCH" DEFW #0D0D
DEFM "ENTER TO MAKE ANOTHER ENTRY"
DEFW #0D0D
DEFM "S TO SAVE OR LOAD"
DEFW #10
DEFB #8D
MESS5 DEFM "ENTER DETAILS TO FIND"
DEFB #8D
MESS6 DEFW #210
DEFB 13
DEFM "END OF RECORD, NO MORE INPUT"
DEFB 13
DEFM "POSSIBLE"
DEFB #8D
MESS7 DEFW #110
DEFM "PRESS"
DEFW #210
DEFM "STOP"
DEFW #110
DEFM "FOR MENU"
DEFW #210
DEFB 2
DEFM "TO"
DEFW #110
```

```
DEFM "TO PRINTOR"
DEFW #210
DEFM "STEP"
DEFW #110
DEFM "FOR HELP"
DEFB #A0
MESS8 DEFW #310
DEFM "YOU ARE IN EDIT MODE"
MC9 DEFB #8D
MESS9 DEFW #210
DEFM "YOU JUST RUN OUT OF MEMORY"
DEFB 13
DEFM "SAVE THE RECORDS OR SOMETHING"
DEFW #10
DEFB #8D
MESS10 DEFM "WHEN YOU ARE IN EDIT MODE"
DEFB 13
DEFM "PRESSING"
DEFW #210
DEFM "NOT"
DEFW #010
DEFM "WILL FIND THE NEXT OCCURENCE OF THE LAST"
DEFM " STRING THAT WAS SOUGHT"
DEFW #0D0D
DEFW #210
DEFM "AND"
DEFW #010
DEFM "WILL INSERT A CHARACTER"
DEFB 13
DEFM "AT THE CURRENT CURSOR POSITION"
DEFB #0D0D
DEFM "DELETE WILL REMOVE THE CHARACTER"
DEFB 13
DEFM "AT THE CURRENT CURSOR POSITION"
DEFB #0D0D
DEFM "IF THERE IS NO SPACE IN A RECORD"
DEFB 13
DEFM "YOU ARE ALTERING USE THE INSERT"
DEFB 13
DEFM "FUNCTION TO MAKE SOME SPACE."
```

```
DEFW #0D0D

DEFM "THE CURSOR KEYS ALLOW YOU TO"

DEFB 13

DEFM "MOVE THROUGH THE TEXT BUT YOU"

DEFB 13

DEFM "CANNOT GO BACK PAST AN ENTER"

DEFW #0D0D

DEFW #410

DEFM "PRESS ENTER TO RETURN TO TEXT"

DEFW #10

DEFB #8D

MESS11 DEFM "PRESS S TO SAVE L TO LOAD"

DEFB #8D

MESS12 DEFM "PRESS M FOR M/DRIVE"

DEFB 13

DEFM "OR T TO TAPE"

DEFB #8D

MESS13 DEFM "INPUT FILE NAME"

DEFB #8D

ZEND DEFB #8D
```

# Оглавление