К. И. ФАХРУТДИНОВ
И. И. БОЧАРОВ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА В СИСТЕМЕ MSX-2

Владивосток
Издательство ИУУ
1991

УДК 681.325.5-181.4

Фахрутдинов К.И., Бочаров И.И.
Программирование на языке ассемблера в системе MSX-2.- Владивосток: Приморский ИУУ.- 1991. 192 с.

В книге описан язык ассемблера для микропроцессора Z-80, используемого в качестве основного процессора системы MSX-2 (КУВТ "YAMAHA"). Рассмотрен процесс создания и выполнения программ на языке ассемблера, организации их связи с программами на языках MSX-BASIC и ASCII-C. На примерах показано использование основных типов команд ассемблера и макроассемблера, описаны основные типы вредоносного программного обеспечения и методы защиты информации.

Книга рассчитана на широкий круг учителей информатики, программистов и школьников и будет полезна в качестве учебного пособия по архитектуре микрокомпьютеров.

ВВЕДЕНИЕ

В этой книге описан язык ассемблера для микропроцессора Z-80, используемого в качестве основного процессора в системе MSX-2 КУВТ "YAMAHA". Компьютеры этого типа широко распространены в Приморском крае и стране в целом. Однако доступной и достоверной информации об архитектуре этой системы микрокомпьютеров на сегодняшний день очень мало.

В первой нашей книге этой серии - "Архитектура микрокомпьютера MSX-2" были собраны и систематизированы разнообразные и иногда, к сожалению, анонимные материалы по устройству микрокомпьютера MSX-2. Во второй книге, которая сейчас перед Вами, рассмотрен процесс создания и выполнения программ на языке ассемблера, организации их связи с программами на языках MSX-BASIC и ASCII-C, на примерах показано использование основных типов команд ассемблера и макроассемблера, описаны основные типы вредоносного программного обеспечения и методы защиты информации.

В приложениях приводится краткий справочник по системе команд микропроцессора Z-80/INTEL 8080 и несколько примеров достаточно больших программ на языке ассемблера.

Для чего нужно знать язык ассемблера ? Этот язык позволяет при достаточной квалификации и некотором опыте очень эффективно управлять работой всех устройств компьютера. В то же время программирование на этом языке требует большой аккуратности и терпения.

Во многих случаях большая часть программы пишется на языке высокого уровня - C, Pascal, Ada и только некоторые подпрограммы - на языке ассемблера. Программы на языках высокого уровня обычно выполняются медленней, чем написанные на языке ассемблера, и такое сочетание позволяет повысить эффективность работы программы и уменьшить время программирования в целом.

Если Вы действительно хотите научиться программировать на языке ассемблера Z-80, внимательно изучайте все примеры и пробуйте выполнить программы на компьютере.

Книга рассчитана на широкий круг учителей информатики, программистов и школьников, желающих получить достаточно глубокие знания по архитектуре микрокомпьютеров вообще и MSX-2 в частности.

ГЛАВА 1. РАЗРАБОТКА И ВЫПОЛНЕНИЕ ПРОГРАММЫ

В этой главе мы рассмотрим, как можно записать, оттранслировать и выполнить программу на языке ассемблера.

Ассемблер - это специальная программа, осуществляющая перевод (трансляцию) программы на языке ассемблера в программу в машинных кодах. Поэтому любой ассемблер можно назвать и транслятором.

Программа записывается на диск с помощью текстового редактора, затем транслируется ассемблером в коды, после этого загружается в память компьютера и выполняется.

Если этих средств нет, то иногда можно оттранслировать программу на языке ассемблера вручную и записать в память машины полученные коды с помощью специальных операторов языка MSX-BASIC.

Команда на языке ассемблера - это, как правило, мнемоническая запись одной команды микропроцессора (машинной команды).

Кроме команд в тексте программы на языке ассемблера могут встретиться и директивы ассемблеру. Директивы обычно не транслируются, а являются указанием ассемблеру выполнить определенное действие.

Для микропроцессора Z-80 применимы ассемблеры DUAD, M80, GEN80, Роботрон-1715 и другие. Кроме различных качественных характеристик они имеют различающиеся наборы директив.

Оттранслированная машинная программа обычно может быть представлена в одном из двух форматов - ".OBJ" или ".COM". Программы типа ".OBJ" загружаются и выполняются в среде MSX-BASIC и обычно транслируются с адреса 8000h и выше. Программы типа ".COM" работают как задачи операционной системы MSX-DOS и размещаются с адреса 100h.

Разные трансляторы могут давать машинный код для микропроцессоров разных типов, например для Z80 или для INTEL8080.

1. Редактирование текста программы

Итак, мы хотим написать программу. Для этого надо загрузить текстовый редактор (например: TOR, MIM, SCED или другой). Это можно сделать в MSX-DOS, вставив диск с редактором в дисковод и набрав в ответ на приглашение DOS имя редактора:

```
A>Tor
```

После загрузки редактора можно начинать набирать программу.

Если мы хотим получить файл типа ".OBJ", то можно воспользоваться ассемблером DUAD. Пример программы для этого транслятора приводится ниже.

Укажем адрес, с которого нужно оттранслировать нашу программу и с которого она будет загружена в память компьютера. Адрес может быть в интервале от 8000h до DE76h для машины учителя и 8000h - F37Fh для машины ученика. Это можно сделать директивой ассемблера ORG. Например,

```
ORG  9000h.
```

При написании программ поле экрана обычно распределяют следующим образом:

- первые восемь позиций отводится под метки;
- вторые восемь под команду;
- следующие восемь под операнды;
- все остальное место под комментарии.

Директиву ORG нужно разместить во втором столбце, а адрес - в третьем. При наборе удобно пользоваться клавишей табуляции TAB.

В программу в любом месте можно вставлять комментарии. Комментарий должен начинаться с символа ';' (точка с запятой). После комментария не может стоять команда (т.е. вся строка после ';' считается комментарием). Комментарии не транслируются ассемблером ни в какой код, а служат для пояснения смысла программы другим программистам или его напоминания самому автору.

Вставим в нашу программу комментарий и напишем несколько команд и директив:

```
        ORG     9000h
; установили начальный адрес компиляции
        LD      b,40h   ; загрузили в регистр b 40h
beg:    DEC     b       ; b <= b-1
        JR      NZ,beg  ; если не 0 - перейти на метку beg
        RET             ; иначе возврат
; конец программы
        END
```

В конце программы ставится директива END - указание ассемблеру, что дальше транслировать не надо или нечего.

Если разрабатывается не главная программа, а подпрограмма, например, вызываемая из MSX-BASICa, то в конце подпрограммы необходимо поставить команду возврата в место вызова - RET.

После того как программа написана, ее необходимо записать на диск. Желательно присвоить ей расширение ".ASM" или ".MAC", указывающее, что это исходный текст программы на языке ассемблера или макроассемблера, например "example.ASM" или "keys.MAC".

Затем программу можно оттранслировать ассемблером.

2. Ассемблирование программы

В примерах мы будем использовать ассемблер системы DUAD фирмы ASCII или макроассемблер M80 (Вы можете использовать любой другой). Название ассемблера можно увидеть в первых строках листинга программы.

п.1. Ассемблирование в системе DUAD

Как уже говорилось, если мы хотим получить файл типа ".OBJ", можно воспользоваться ассемблером DUAD.

Для вызова системы DUAD необходимо перейти в режим MSX-BASIC и набрать команду: run"DUAD".

Затем Вы должны выбрать в "меню" Assembler, нажав клавишу "1". При правильном запуске ассемблера Вам будут заданы вопросы. В ответ на них Вы должны ввести имя файла или нажать <ВВОД>, если вам соответствующее действие не нужно. Системой задаются следующие вопросы:

```
Source   : [ файл - исходный текст ]
List     : [ файл для записи листинга ]
Object   : [ файл для записи оттранслированой программы ]
Label A  : [ файл для меток и адресов по алфавиту]
Label L  : [ файл для меток и адресов по возрастанию адресов ]
Cross ref: [ файл перекрестных ссылок ]
```

На первый вопрос ответ обязателен, на другие - нет. Например, можно ответить так:

```
Source   : example.asm
List     : example.lst
Object   : example.obj
Label A  :
Label L  :
Cross ref:
```

После этого ассемблер начнет трансляцию и через некоторое время выдаст сообщение 'No errors in text' (Нет ошибок в тексте). Это значит, что наша программа написана без ошибок и успешно оттранслирована. Если будут другие сообщения, то транслирование не удалось (причиной может быть ошибка в программе, защита на диске или еще что-то).

Для выхода из системы DUAD на вопрос "Source:" введите *BASIC.

Если в программе были ошибки, можно просмотреть с помощью текстового редактора файл с расширением ".LST", например,

```
A>TOR example.lst
```

Листинг нашей программы, если все было сделано правильно, будет иметь вид:

```
           Z80-Assembler   Page:    1
                    ORG     9000h
      ; установили начальный адрес компиляции
9000 0640           LD      B,40h   ; загрузили в регистр B 40h
9002 05      BEG:   DEC     B       ; B <= B-1
9003 20FD           JR      NZ,BEG  ; если не 0 - перейти на BEG
9005 C9             RET             ; иначе возврат
      ; конец программы
                    END
No errors in text
```

Колонка слева - шестнадцатеричные адреса машинных команд (9000, 9002, 9003, 9005). Рядом с адресом - машинный код команды. Таким образом наша программа в машинных кодах выглядит так:

06400520FDC9.

п.2. Ассемблирование посредством M80

С помощью ассемблера M80 можно получить программы типа ".COM". Текст программы, которую мы набирали выше, будет иметь несколько отличающийся вид:

```
       .Z80
       LD      B,40h   ; load 40h  in B
BEG:   DEC     B       ; B <= B-1
       JR      NZ,BEG  ; if not 0 - go to BEG
       RET             ; else return
; end of program
       END
```

Первая директива этой программы сообщает ассемблеру, что команды записаны в соответствии с мнемоникой Z80.

После записи текста программы в файл с расширением ".ASM", не выходя из MSX-DOS в MSX-BASIC, наберите команду вида:

```
A>M80  =example.asm/L
```

Листинг программы example.ASM будет записан в файл example.PRN. Результат трансляции будет записан в файл example.REL (который затем должен быть обработан редактором связей и преобразован в файл типа ".COM").

Более полная форма вызова ассемблера M80 выглядит так:

```
A>M80  example,example=example.asm/L
```

Здесь первое имя - имя результирующего файла, второе - для листинга, третье - текст на ассемблере.

Если не нужен ни машинный код, ни листинг, пишут:

```
A>M80  ,=example.asm
```

Если нужен только листинг, то так:

```
A>M80  ,example=example.asm/L
```

Если исходный файл имеет расширение MAC, то в командах это расширение можно опускать.

При трансляции могут быть использованы следующие ключи:

/O - печатать восьмеричные адреса и числа в восьмеричном виде;
/P - создание стека для ассемблера;
/R - имя объектного файла как у исходного;
/X - установка STFCOND;
/L - вывод листинга в файл типа PRN с именем как у исходного;
/C - генерировать файл перекрестных ссылок с именем как у исходного и расширением CRF;
/Z - используется мнемоника Z80;
/I - используется мнемоника INTEL 8080;
/H - печатать шестнадцатеричные адреса и числа в шестнадцатеричном виде;
/M - инициализировать память для директивы DS нулями.

Для создания файла типа COM вызывается редактор связей L80.

3. Редактирование связей и сборка программы

Редактирование связей и сборка программы выполняются после трансляции ассемблером M80. Они осуществляются при помощи редактора связей L80. На этом этапе объединяются воедино все разрозненные части программы, записанные в различных REL-файлах или библиотеках. Могут также быть подключены библиотеки стандартных или дополнительных функций языка C или других языков. Результатом редактирования является неперемещаемый объектный код программы, записываемый в файл с расширением ".COM".

Задание на редактирование программы, написанной на языке ассемблера, как правило, выглядит следующим образом:

```
A>L80 имя,имя/n/e
```

Здесь имя - это имя REL-файла, оттранслированного ассемблером M80.

Задание на редактирование программы, в которой управляющим модулем является функция main, написанная на языке C, обычно производится следующим образом:

```
A>L80 ck,Осн-Имя,Имя-Файла1[/s],Имя-Файла2[/s],...,clib/s,crun/s,
      cend,Имя-COM-файла/n/e:xmain
```

Здесь Осн-Имя - это имя REL-файла, содержащего код функции main. REL-файлы ck, clib, crun и cend входят в комплект ASCII C и их подключение обязательно для пользователя, избегающего вникать в тонкости структуры компилятора языка C.

Пользователь должен подключить все REL-файлы, содержащие модули, необходимые для разрешения внешних ссылок.

Опции редактора связей L80:

```
/s        - подключить не всю библиотеку, а только необходимые мо-
            дули;
/p        - установить начальный адрес размещения программы в
            памяти;
/d        - установить адрес размещения сегмента данных;
/u        - выдать список неразрешенных ссылок;
/m        - выдать адреса глобальных имен;
/n[:имя]  - записать COM-файл на диск[,установить точку входа
            в программу];
/g[:имя]  - выполнить программу [ с указанной точки входа];
/e[:имя]  - выйти в DOS [,установить точку входа в программу].
```

Редактор связей создает программу, загружающуюся и стартующую с адреса 100H.

Чтобы создать программу, загружающуюся с адреса, отличного от 100H, необходимо не только использовать опцию /p, но и указать имя точки входа в опции n или e.

4. Выполнение программы

В случае успешной трансляции мы можем выполнить нашу программу. Для запуска программы типа ".OBJ" надо выйти в MSX-BASIC и загрузить программу командой:

```
Bload"example.obj",R
```

Буква "R" обозначает "выполнить". Машина тут же должна выдать Ok. Так как наша программа уже загружена, ее можно выполнить снова. Для этого надо определить ее как функцию и передать ей управление.

```
Defusr = &h9000: i = usr(0)
```

Машина снова выдаст Ok.

Как Вы могли понять, программа написанная нами выше - не что иное, как обыкновенная задержка во времени. Но надо сказать, что эту задержку при выполнении Вы не заметите. Это объясняется большой скоростью выполнения программ на языке ассемблера. Поэтому для написания задержек обычно используют пару регистров.

Это мы рассмотрим несколько ниже, а сейчас попробуем написать эту же программу, но "оттранслировав" самостоятельно и записав через BASIC (можно предварительно загрузить в текстовый редактор листинг нашей программы и посмотреть, как она оттранслирована):

```
10 DATA 06,40        :REM LD   B,40    9000
20 DATA 05           :REM DEC  B       9002
30 DATA 20,FD        :REM JR   NZ,9002 9003
40 DATA C9           :REM RET          9005
100 REM загружаем коды в память
110 DATA Z
120 N=&H9000
130 READ A$:IF A$<>"Z" THEN POKE N+I,VAL("&H"+A$):I=I+1:
     GOTO 130
140 DEFUSR=N:I=USR(0)
150 END
```

Если Вы не уверены, что оттранслировали правильно, то загрузите в текстовый редактор листинг (расширение LST) и сверьте коды.

Запустив эту программу, Вы получите тот же результат, что и при запуске файла OBJ, однако постоянного обращения к диску уже не требуется.

Как уже говорилось выше, программу типа ".REL" нужно обработать редактором связей, чтобы получить соответствующую программу типа ".COM".

Для запуска программы с расширением ".COM" наберите в режиме MSX-DOS ее имя без расширения:

```
A>example
```

Итак, если у Вас все получилось, поздравляем Вас с выполнением Вашей первой программы на языке ассемблера Z80 !!!

5. Организация связей с программами на языке MSX-BASIC

При разработке подпрограмм, написанных в кодах, которые должны вызываться из программ на языке MSX-BASIC, часто возникает проблема передачи параметров в подпрограмму и получения результата из подпрограммы.

Для осуществления этого возможны два основных способа - использование общей памяти и собственно передача/получение параметров. Может использоваться и комбинация этих способов.

п.1. Общая память

В этом случае выделяется одна или несколько ячеек памяти с заранее известными адресами, и программы обмениваются данными через эти ячейки. На языке MSX-BASIC для этого используются оператор POKE и функция PEEK.

Пример программы на языке MSX-BASIC.

```
10 CLEAR 200,&H9000      : REM установка границ
20 DEFUSR = &H9000       : REM адрес подпрограммы
30 BLOAD "example.obj"   : REM загрузка с диска
40 X = 124               : REM число для передачи
50 POKE &HA000,X         : REM запись аргумента
60 I = USR( 0)           : REM вызов подпрограммы
70 Y = PEEK( &HA001)     : REM берем результат
80 PRINT Y
90 END
```

Листинг вызываемой программы на языке ассемблера:

```
'comm. memory '   Z80-Assembler   Page:    1
                        TITLE   'comm. memory '
                        ORG     9000h
9000 3A00A0             LD      A,(0A000h) ; берем аргумент
9003 3C                 INC     A          ; увеличить A
9004 3C                 INC     A          ; еще раз
9005 3201A0             LD      (0A001h),A ; записываем результат
9008 C9                 RET                ; возврат
                        END
'comm. memory '   Z80-Assembler   Page:    2
No errors in text
```

В результате работы BASIC-программы с этой подпрограммой должно получиться число 126.

Достоинством этого способа передачи данных является возможность передачи и получения из подпрограммы больших массивов информации.

п.2. Передача и получение параметров

При вызове подпрограммы, определенной как USR, интерпретатор языка MSX-BASIC записывает в регистр HL адрес арифметической переменной, в DE - ссылку на адрес строкового выражения, а в аккумулятор A и ячейку &hF663 - тип переменной. При этом для значений целого типа адрес в HL нужно увеличить на 2, а первые три байта строкового указателя в DE хранят длину и реальный адрес строки.

Иногда удобнее воспользоваться готовыми подпрограммами передачи и преобразования данных.

Для передачи однобайтного аргумента из программы на языке MSX-BASIC подпрограмме в кодах через регистр A используется подпрограмма ПЗУ по адресу &h521F.

Для передачи двухбайтного аргумента из MSX-BASICa используется подпрограмма с начальным адресом &h2F8A. Она записывает аргумент в регистр HL.

Для передачи двухбайтного результата из подпрограммы в кодах в MSX-BASIC используется подпрограмма &h2F99. Она возвращает значение, записанное в HL.

Рассмотрим примеры.

Передача однобайтного аргумента и получение результата. Программа на MSX-BASICe:

```
10 CLEAR 200,&H9000      : REM установка границ
20 DEFUSR = &H9000       : REM адрес подпрограммы
30 BLOAD "example.obj"   : REM загрузка с диска
40 X = USR(45)           : REM вызов подпрограммы
50 PRINT X               : REM возврат значения в X
60 A = 73
70 X = USR( A)           : REM вызов подпрограммы
80 PRINT X
90 END
```

Нельзя в качестве аргумента использовать значение не в диапазоне 0..255. Например, значение 260 вызовет ошибку. Листинг программы example.asm:

```
'pass one-byte'   Z80-Assembler   Page:    1
                        TITLE   'pass one-byte'
                        ORG     9000h
                ; --- записать аргумент в A
9000 CD1F52             CALL    521Fh
                ; --- обработка аргумента
9003 3C                 INC     A       ; увеличить A
9004 3C                 INC     A       ; еще раз
                ; --- возврат результата через HL
9005 2600               LD      H,0
9007 6F                 LD      L,A
9008 C3992F             JP      2F99h   ; возвращаем результат
                        END
```

После выполнения программы будут напечатаны числа 47 и 75.

Передача двухбайтного аргумента и получение результата. Программа на MSX-BASICe:

```
10 CLEAR 200,&H9000      : REM установка границ
20 DEFUSR = &H9000       : REM адрес подпрограммы
30 BLOAD "example.obj"   : REM загрузка с диска
40 X = USR(1045)         : REM вызов подпрограммы
50 PRINT X
60 A = 23678
70 X = USR( A)           : REM вызов подпрограммы
80 PRINT X
90 END
```

Листинг программы example.asm:

```
'pass two-byte'   Z80-Assembler   Page:    1
                        TITLE   'pass two-byte'
                        ORG     9000h
                ; --- записать аргумент в HL
9000 CD8A2F             CALL    2F8Ah   ; перед. 2-х байт. пар.
                ; --- обработать аргумент
9003 23                 INC     HL      ; увеличить HL
9004 23                 INC     HL      ; еще раз
                ; --- возврат результата через HL
9005 C3992F             JP      2F99h   ; возвращаем результат
                        END
'pass two-byte'   Z80-Assembler   Page:    2
```

После выполнения программы будут напечатаны числа 1047 и 23680.

Как Вы заметили, программы на языке MSX-BASIC отличаются фактически только допустимым диапазоном передаваемых функции USR значений. Отличен же способ их обработки в подпрограмме: вызов &h2F8A или &h521F.

6. Организация связей с программами на языке C

Основные вопросы, которые нужно иметь в виду при организации связей программ, написанных на языке C с программами на языке ассемблера, - это порядок передачи параметров, правила написания имен и редактирование связей.

п.1. Передача параметров

Передача параметров для функций языка C с фиксированным числом параметров подчинена следующему соглашению о связях:

Первый параметр передается:

- через регистр A, если он занимает один байт;
- через пару HL, если занимает два байта.

Второй и третий параметры передаются соответственно через регистры E и C или через DE и BC.

Остальные параметры записываются в стек, по 2 байта на параметр независимо от типа.

В случае функции с переменным числом параметров количество параметров передается через HL, все параметры записываются в стек по два байта независимо от типа.

Результат любой функции помещается в A (char) или в HL (int и другие типы).

Параметры из стека убирает вызывающая функция. При выходе из вызываемой функции значение SP должно быть равным значению при входе.

п.2. Символические имена

Транслятор языка MSX-C распознает первые 16 символов имен. Внешние имена распознаются по первым 6 символам. При флаге -l cg в полученном тексте на ассемблере будет более 6 символов.

Отметим, что ASCII C заменяет при компиляции символ "_" на "@", а к каждому символическому имени, являющемуся внешней ссылкой или глобальной переменной и состоящему менее чем из 6 символов, приписывает справа тот же символ "@". Пользователь, пишущий программы на языке ассемблера, должен это учитывать, чтобы правильно обращаться к C-функциям, и чтобы к его процедурам можно было обращаться из C-программ.

Например, пусть функция на языке C имеет заголовок

```
char a_fun (arg1,arg2)
int arg1,arg2;
```

Чтобы обратиться к ней из программы на языке ассемблера, следует написать :

```
LD   hl,...
LD   de,...
CALL a@fun@
LD   (...),a
...
```

Очевидно, что к программам на языке ассемблера, имена которых содержат символ "_" или не имеют на конце "@" при длине менее 6 байт, доступ из С-программ невозможен.

Все глобальные переменные при трансляции С-программы в ассемблерный текст получают описатель PUBLIC, а внешние ссылки - описатель EXTRN. Такое же соглашение работает в ассемблере М80.

п.3. Трансляция и сборка разноязыковых модулей

Возможно создание программ, в которых часть модулей написана на языке ассемблера, а часть - на языке С. Чаще всего основу составляют программы на языке С, в том числе и главный модуль, а "тонкие" вещи делаются на языке ассемблера.

Допустим, мы разработали следующие подпрограммы на языке ассемблера:

```
      MSX.M-80 1.0001-Apr-85  PAGE 1
                              .Z80
                              PUBLIC KillBf@
                              PUBLIC InKey@
                              PUBLIC Wait@
; ===  чистка буфера клавиатуры
0000'   F7            KillBf@: RST  30h
0001'   00                     DB   0
0002'   0156                   DW   156h
0004'   C9                     RET
; ===  ввод кода нажатой клавиши
; ===  выход: [a]=0, если ничего не нажато,
; ===         иначе - [a] - код нажатой клавиши
0005'   AF            InKey@:  XOR  A     ; чистим A
0006'   F7                     RST  30h   ; нажато что-нибудь ?
0007'   00                     DB   0
0008'   009C                   DW   9Ch
000A'   C8                     RET  Z     ; если нет - выход
000B'   F7                     RST  30h   ; иначе берем код
000C'   00                     DB   0
000D'   009F                   DW   9Fh
000F'   C9                     RET
; ===  ожидание нажатия клавиши
; ===  вход [a] - код символа, который нужно ждать
0010'   CD 0000'      Wait@:   CALL KillBf@
0013'   47                     LD   B,A
0014'   CD 0005'              CALL InKey@
0017'   B8                     CP   B
0018'   20 FA                 JR   NZ,$-4
001A'   C9                     RET
                               END
```

Если эти подпрограммы записать в файл keys.MAC, то получить из него файл типа REL можно командой:

```
A>m80 =Keys.mac/L
```

Теперь приведем исходный текст программы на языке C, вызывающей эти подпрограммы на языке ассемблера.

```
#include <stdio.h>
main()
{
  VOID KillBf();  /* описания подпрограмм */
  VOID Wait();
  char InKey();
  KillBf();       /* чистим буфер и ждем нажатия любой клавиши */
  while( InKey() == '\0' );
  printf("%s\r\n\n"," .... ");
  Wait(' ');      /* ждем нажатия пробела */
  printf("%s\r\n"," ::::");
}
```

Для трансляции и редактирования файлов exam.C и keys.REL можно выполнить следующие команды:

```
A>cf -me exam
A>cg exam
A>m80 =exam.asm
A>l80 ck,exam,keys,clib/s,crun/s,cend,exam/n/e:xmain
A>exam
```

ГЛАВА 2. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА

В этой главе мы рассмотрим основные команды языка ассемблера и директивы самого ассемблера, приведем тексты и листинги программ.

Полностью система команд Z80 приведена в Приложении 1.

1. Директивы ассемблера

С помощью директив (псевдокоманд) программист дает указания ассемблеру по трансляции программы на языке ассемблера, управляет процессом трансляции. В отличие от команд языка ассемблера директивы, как правило, не транслируются в машинные команды.

У разных ассемблеров могут быть отличающиеся наборы директив. Мы будем в основном придерживаться набора директив ассемблеров системы DUAD и M80.

Во первой главе мы уже сталкивались с некоторыми директивами. Рассмотрим их немного подробнее.

ORG - определение начального адреса трансляции (или загрузочного адреса). Ассемблер настраивает программу с указанного адреса. В основном это касается команд перехода, использующих метки, вместо которых нужно будет подставить конкретные адреса, и меток данных. Пример:

```
ORG   9000h
```

DUAD-ассемблер допускает только одну команду ORG. Другие ассемблеры могут допускать и больше (как установку счетчика размещения).

Нужно иметь в виду, что в разных режимах трансляции директива ORG может иметь различные смыслы. Подробнее об этом смотрите в Главе 3.

END - указание на конец текста транслируемой программы. Многие ассемблеры кроме этого воспринимают адрес или метку в поле операндов директивы END (если она есть) как стартовый адрес программы.

INCLUDE - указание включить в текст программы текст, находящийся в указанном в директиве файле. Включение производится в то место, где стоит INCLUDE. Система DUAD не разрешает, чтобы включаемый таким образом файл в свою очередь тоже имел директиву INCLUDE. Пример:

```
INCLUDE a:stdbeg.ASM
```

MACLIB - указание включить в текст программы макробиблиотеку, находящуюся в указанном в директиве файле. Включение производится в то место, где стоит MACLIB. Пример:

```
MACLIB a:macros.MAC
```

EQU - приписывание имени константе. С помощью этой директивы константе или константному выражению приписывается имя, которое затем можно использовать везде, где использовалась константа. Если использовано выражение, ассемблер вычисляет его значение (значения всех имен должны быть уже вычислены) и подставляет это значение в команду. В выражении могут использоваться операции +, -, *, /, а также скобки. Имена обычно приписываются тем константам, значения которых могут меняться в ходе разработки программы или в ходе ее эксплуатации. Пример:

```
scrnum    EQU   2
```

```
nospr     EQU   16
dma       EQU   fcb+len*3
argum     EQU   0A001h
```

.REQUEST - просмотр неопределенных внешних меток. Сборщик просматривает файлы типа .REL, ищет глобальные имена. Пример:

```
.REQUEST subr
```

.COMMENT или %COMMENT - комментарии к программе. Первый непустой символ после слова COMMENT - ограничитель. Текст комментария длится до нового появления ограничителя.

NAME ('имя-модуля') - дает имя модулю.

.Z80 - используется мнемоника Z80

.8080 - используется мнемоника INTEL 8080.

Имеются также директивы для резервирования и заполнения памяти значениями, управления выдачей листинга, для условной генерации и т.д. Они будут рассмотрены ниже.

2. Системы счисления

Кроме привычной всем нам десятичной системы счисления существуют также двоичная, восьмеричная и шестнадцатеричная системы счисления. В десятичной системе мы имеем 10 знаков (цифры от 0 до 9), в двоичной системе их всего два (0 и 1), зато в шестнадцатеричной - 16 (цифры от 0 до 9 и латинские буквы A-F). Ниже приведена таблица соответствия между первыми 16 числами разных систем счисления:

дес.	двоич.	восьм.	шест.
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Как Вы могли заметить, для того, чтобы умножить число в двоичной системе на 2, необходимо просто сдвинуть биты (разряды числа, 0 или 1) на одну позицию влево. Аналогично происходит деление; разумеется, сдвиг происходит вправо. Это свойство положено в конструкцию ЭВМ. В языке ассемблера есть команды сдвига влево/вправо, что дает возможность достаточно просто

умножать/делить на любое число, кратное двум.

Необходимо заметить, что при написании программ на языке ассемблера пользуются в основном двоичной, шестнадцатеричной и реже - десятичной системами счисления.

Легко освоить и перевод из двоичной системы в шестнадцатеричную: необходимо разбить двоичное число на группы по 4 бита и воспользоваться вышеприведенной таблицей.

3. Выделение памяти и запись значений

Несколько директив ассемблера предназначены для выделения памяти для переменных программы, а также для первоначального заполнения выделенной памяти необходимыми значениями.

Если шестнадцатеричная константа начинается с буквы, то перед ней обязательно нужно ставить цифру 0. Например, 0B31Ch.

DEFS - резервирование указанного количества байт. Допустимо сокращение DS. Если имеется второе число через запятую, то это означает, что выделенную память нужно заполнить указанным значением (в противном случае память либо обнуляется, либо заполнена "мусором"). Например,

```
storage:   DEFS   16
block:     DS     255
fillvrm:   DS     56,0
units:     DS     24,0Fh
```

DEFB - запись указанных значений в память побайтно с одновременным резервированием памяти. Допустимо сокращение DB. Например,

```
x:         DEFB   25
st:        DB     0F2h
data:      DEFB   1Fh,93h,0A0h,0,56
year:      DB     '(C) ДВГУ. 1989',0
```

DEFW - запись указанных значений в память в двухбайтном формате с одновременным резервированием памяти. При записи младший байт значения будет поставлен первым (интеловский способ хранения значений). Допустимо сокращение DW. Например,

```
word:      DW     13A7h
integ:     DEFW   1F39h,0Ah,8000h,125
```

DEFM - запись строкового значения в память. Например:

```
text:      DEFM   'I am very glad!'
```

DC - запись строкового значения. Первый бит каждого байта обнуляется, но последний байт строки запоминается с установленным 7-м битом.

Метки перед всеми перечисленными директивами могут и отсутствовать.

Рекомендуем Вам внимательно изучить, как были оттранслированы ассемблером директивы из приведенных ниже примеров.

```
                Z80-Assembler   Page:    1
                        ORG 9000h
9000            storage:    DEFS   16
9010            block:      DS     255
910F 19         x:          DEFB    25
9110 F2         st:         DB      0F2h
9111 1F93A000   data:       DEFB    1Fh,93h,0A0h,0,56
9115 38
9116 28432920   year:       DB      '(C) ДВГУ. 1989',0
911A E4F7E7F5
911E 2E203139
9122 383900
9125 A713       word:       DW      13A7h
9127 391F0A00   integ:      DEFW    1F39h,0Ah,8000h,125
912B 00807D00
912F 4920616D   text:       DEFM    'I am very glad!'
9133 20766572
9137 7920676C
913B 616421
                            END
```

В ассемблере M80 имеется директива .RADIX, которая позволяет устанавливать любое основание системы счисления с 2 до 16 для констант, действующее по умолчанию. Явно основание указывается буквами: b - двоичное, d - десятичное, o - восьмеричное, h - шестнадцатеричное. Изучите пример трансляции ассемблером M80:

```
      MSX.M-80 1.00 01-Apr-85 PAGE 1
                                        ORG     9000h
               ; ----
  9000'   38                            DB      56d
  9001'   07                            DB      111b
  9002'   3F                            DB      77o
  0002                                  .RADIX  2
  9003'   9C                    byte:   DB      10011100
  9004'   0A                            DB      1010
  000C                                  .RADIX  12
  9005'   79 23                 nmb:    DB      0A1,2B
  0010                                  .RADIX  16
  9007'   FCAC 01DE             addr:   DW      0FCAC,01DE
               ; ----
                                        END
No Fatal error(s)
```

Обратите внимание на то, что в листинге M80 в отличие от листинга ассемблера DUAD младший и старший байты значения не переставлены.

Локальные метки обычно могут иметь длину до шестнадцати символов, а глобальные и внешние (см.ниже) - до шести. Метка может содержать символы A..z, 0..9, Ъ, точку, ?, @, подчерк.

В командах языка ассемблера можно использовать не только просто метки, но и выражения над метками и числами. Допускаются круглые скобки и следующие операции:

NOT e - отрицание, инверсия
e1 AND e2 - конъюнкция
e1 OR e2 - дизъюнкция
e1 XOR e2 - исключающее или
e1 SHL e2 - сдвиг первого операнда влево на значение e2
e1 SHR e2 - сдвиг первого операнда вправо на значение e2
e1 + e2 - сложение e1 с e2
e1 - e2 - вычитание e2 из e1
e1 / e2 - деление e1 на e2
e1 * e2 - умножение e1 на e2
e1 MOD e2 - остаток от деления e1 на e2
HIGH e - восемь старших битов двухбайтного слова
LOW e - восемь младших битов двухбайтного слова
NULL e - истина, если аргумент равен нулю

Допускаются также сравнения EQ (=), NE(╬), LT (<), LE (є), GT (>), GE (Є).

Текущий адрес обозначается знаком "$" или "Ъ". Например, если текущий счетчик размещения равен 901Ah, то после выполнения директивы

```
          EndLoad   EQU  $-7
```

значением имени EndLoad станет 9013h.

Если значением имени TrapLoc является FCCAh, то команда

```
                    LD  (HL),Low(TrapLoc)
```

будет эквивалентна команде

```
                    LD  (HL),0CAh.
```

Еще один пример - использование ассемблером логических операций. Пара - директива и команда

```
          P.Stop   EQU   3
                   LD    A,not(1 SHL P.Stop)
```

эквивалентны команде

```
                   LD    A,11110111b.
```

4. Команды загрузки и обмена

С помощью команд загрузки производится обмен данными между регистрами, памятью и регистром, регистром и памятью. Команд пересылки непосредственно из одной ячейки памяти в другую нет, но это можно сделать через регистры.

Кроме этого, команды загрузки позволяют записать некоторое число в регистр или регистровую пару.

Команды загрузки делятся на две большие группы - команды 8-разрядной загрузки и команды 16-разрядной загрузки. Посмотрите примеры команд загрузки одного байта, оттранслированные в системе DUAD.

```
                  Z80-Assembler   Page:    1
                  ORG  0A000h
A000 0603         LD   b,3         ; 3 => регистр b
A002 2632         LD   h,32h       ; 32h => регистр h
A004 3AAFFC       LD   a,(0FCAFh)  ; содержимое FCAFh =>
                                   ;     регистр a
A007 3A17A0       LD   a,(data)    ; содержимое data =>
                                   ;     регистр a
A00A 4B           LD   c,e         ; регистр e => рег. c
A00B 7E           LD   a,(HL)      ; содерж. ячейки по
                                   ; адресу в HL =>
                                   ;      регистр a
A00C 02           LD   (BC),a      ; регистр a =>
                                   ;      по адресу в BC
A00D 32ACFC       LD   (0FCACh),a  ; регистр a => в FCACh
A010 3217A0       LD   (data),a    ; регистр a => в data
A013 3219A0       LD   (data+2),a  ; рег. a => в data+2
A016 C9           RET
A017 46     data: DEFB 46h
A018              DEFS 2,0
                  END
```

При загрузке в регистровую пару, например BC, двухбайтного значения с адресом adr, байт, хранящийся по адресу adr, загружается в регистр C, а байт по адресу adr+1 - в регистр B. Аналогично - для регистров DE и HL. Запись из регистровой пары в память снова переставляет байты. Поскольку в памяти байты переставлены, это означает, что в регистровой паре - обычная запись.

Изучите примеры трансляции команд 16-ти разрядной загрузки, приведенные ниже.

```
                  Z80-Assembler   Page:    1
                     ORG  0A000h
A000 111F20          LD   DE,201Fh    ; 20h => в D
                                      ; 1Fh => в E
A003 2AAFFC          LD   HL,(0FCAFh) ; байт адр. FCAFh => L
                                      ; байт адр. FCB0h => H
A006 2117A0          LD   HL,data     ; A0h  => H
                                      ; 17h  => L
A009 2A17A0          LD   HL,(data)   ; 46h (data) => L
                                      ; A7h (data+1) => H
A00C ED4362DE        LD   (0DE62h),BC ; содерж. B => в DE63h
                                      ; содерж. C => в DE62h
A010 2217A0          LD   (data),HL   ; содерж. H => в data+1
                                      ; содерж. L => в data
A013 2219A0          LD   (data+2),HL ; содерж. H => в data+3
                                      ; содерж. L => в data+2
A016 C9              RET
A017 46A7 data:      DEFW 0A746h
A019                 DS   2,0
                     END
```

Обратите особое внимание на команду LD HL,data и ее отличие от следующей за ней команды.

Приведем три листинга программ, осуществляющих перестановку однобайтных и двухбайтных значений.

```
    MSX.M-80 1.00  01-Apr-85  PAGE 1
; === перестановка двух байтов - first и second
                         .Z80
8000              first  EQU  8000h
8010              second EQU  8010h
0000'   3A 8000          LD   A,(first)
0003'   47               LD   B,A
0004'   3A 8010          LD   A,(second)
0007'   32 8000          LD   (first),A
000A'   78               LD   A,B
000B'   32 8010          LD   (second),A
000E'   C9               RET
                         END
```

Для перестановки двух смежных байтов можно использовать команды загрузки двух байтов.

```
      MSX.M-80 1.00     01-Apr-85  PAGE 1
; === перестановка двух смежных байтов - first и first+1
                          .Z80
8000                first   EQU  8000h
0000'   2A 8000             LD   HL,(first)
0003'   7C                  LD   A,H
0004'   65                  LD   H,L
0005'   6F                  LD   L,A
0006'   22 8000             LD   (first),HL
0009'   C9                  RET
                            END
```

Перестановка двухбайтных значений очень проста, если можно использовать две регистровые пары.

```
      MSX.M-80 1.00     01-Apr-85  PAGE 1
; === перестановка двухбайтных значений  first и second
                          .Z80
8000                first   EQU  8000h
8010                second  EQU  8010h
0000'   2A 8000             LD   HL,(first)
0003'   ED 4B 8010          LD   BC,(second)
0007'   ED 43 8000          LD   (first),BC
000B'   22 8010             LD   (second),HL
000E'   C9                  RET
                            END
```

Команды обмена позволяют производить обмен содержимым между регистровыми парами DE и HL, стеком и регистрами HL, IX, IY, основным и дополнительным наборами регистров. Например,

```
      MSX.M-80 1.00     01-Apr-85  PAGE 1
                          .Z80
8000                data1    EQU   8000h
8010                data2    EQU   8010h
0000'   21 8000             LD    HL,data1
0003'   EB                  EX    DE,HL
0004'   21 8010             LD    HL,data2
0007'   23                  INC   HL
0008'   EB                  EX    DE,HL
0009'   C9                  RET
                            END
```

Дополнительный набор регистров может использоваться для кратковременного хранения значений основных регистров. Например,

```
      MSX.M-80 1.00     01-Apr-85  PAGE 1
                          .Z80
0000'   D9                EXX
0001'   2A 000E'          LD   HL,(data)
0004'   23                INC  HL
0005'   44                LD   B,H
0006'   4D                LD   C,L
0007'   03                INC  BC
0008'   ED 43 0010'       LD   (data+2),BC
000C'   D9                EXX
000D'   C9                RET
000E'   34A1      data:   DW   34A1h
0010'                     DS   2,0
                          END
```

Содержимое регистровых пар можно сохранить в стеке. Стек - это область памяти, организованная по принципу "последним пришел, первым вышел" или принципу "стопки тарелок". Например, для обмена содержимым регистровых пар HL, BC, DE можно написать:

```
      MSX.M-80 1.00     01-Apr-85  PAGE 1
                          .Z80
; === в стек
0000'   D5                PUSH  DE
0001'   C5                PUSH  BC
0002'   E5                PUSH  HL
; === из стека
0003'   C1                POP   BC
0004'   D1                POP   DE
0005'   E1                POP   HL
0006'   C9                RET
                          END
```

В результате произойдет запись HL => BC, BC => DE, DE => HL.

5. Управление печатью листинга

Несколько директив ассемблера предназначены для управления порядком выдачи листинга программы. Имеется следующий набор директив:

TITLE строка - определение заголовка длиной до 16 символов для каждой страницы программы. Например,

```
TITLE Conversion
```

PAGE число - задание размера страницы листинга в заданное число строк. Например,

```
PAGE 44
```

.LIST - печатать листинг. Поскольку этот режим действует по умолчанию, основное применение директивы - включение печати после директивы .XLIST.

.XLIST - выключить выдачу листинга сразу после этой директивы

.PRINTX сообщение - вывод на экран сообщения по ходу трансляции программы. Используется для отслеживания процесса трансляции. Первый непустой знак после директивы означает ограничитель, которым должно закончиться сообщение. Например,

```
.PRINTX * OCEAN have been assembled *
```

.CREF - создание файла перекрестных ссылок.

.XCREF - прекращение выдачи файла перекрестных ссылок.
SUBTTL текст - печать подзаголовка титула.
*eject выражение - новая страница размером "выражение".

6. Арифметические команды

К арифметическим командам Z-80 относятся команды увеличения и уменьшения значения на единицу, команды сложения и вычитания, а также команда изменения знака (вычитания из нуля). Арифметические команды работают с одно- и двухбайтными операндами. Команд умножения и деления нет, они моделируются сложением и вычитанием.

п.1. Представление операндов

При выполнении арифметических команд каждый операнд обычно представляется как 7-и разрядное число со знаком в старшем разряде, в дополнительном двоичном коде.

Двоичное	Шестнадц.	Десятичн.
0111 1111	7F	127
0111 1110	7E	126
...	...	...
0000 0011	03	3
0000 0010	02	2
0000 0001	01	1
0000 0000	00	0
1111 1111	FF	-1
1111 1110	FE	-2
...	...	...
1000 0001	81	-127
1000 0000	80	-128

Аналогично представляются 16-разрядные значения:

Двоичное	Шестнадц.	Десятичн.
0111 1111 1111 1111	7FFF	32767
0111 1111 1111 1110	7FFE	32766
...	...	...
0000 0000 0000 0011	0003	3
0000 0000 0000 0010	0002	2
0000 0000 0000 0001	0001	1
0000 0000 0000 0000	0000	0
1111 1111 1111 1111	FFFF	-1
1111 1111 1111 1110	FFFE	-2
...	...	
1000 0000 0000 0001	8001	-32767
1000 0000 0000 0000	8000	-32768

Кроме этого, для однобайтовых величин иногда используют двоично-десятичное представление (BCD). В этом случае каждая из двух десятичных цифр значения представляется четырьмя битами (полубайтом). В таком представлении в байте не может быть сочетаний битов, соответствующих шестнадцатеричным цифрам A..F, т.е. 1010, 1011, ... , 1111. Например,

Двоичн. запись	Шестнадц.	Десятичн.	Двоично-десят.(BCD)
0011 0111	37	55	37
1001 0110	96	150	96
0001 1010	1A	26	-

п.2. Работа с восьмиразрядными числами

При выполнении команд один из операндов обычно должен быть помещен в регистр A, а другой (если команда имеет длину один байт) - в один из 8-разрядных регистров микропроцессора или в ячейку памяти, адресуемую косвенно. В двухбайтовой команде значение второго операнда непосредственно задается во втором байте команды. Результат выполнения команды помещается в регистр A (аккумулятор).

Команда ADD позволяет сложить два операнда. Сложение двух операндов со значением бита переноса C происходит по команде ADC. Вычитание из аккумулятора второго операнда и учет значения бита заема C производится соответственно командами SUB и SBC.

Очень часто при написании программ используют команды INC и DEC, служащие для увеличения или уменьшения содержимого регистра, регистровой пары или ячейки памяти, адресуемой по содержимому регистровой пары на единицу.

Для изменения знака числа, находящегося в аккумуляторе A, используется команда NEG. Эта команда работает как вычитание из нуля содержимого аккумулятора.

Арифметические команды, работающие с однобайтными значениями, выставляют флаги Z (ноль), S (отрицательное число), N (команда вычитания или уменьшения), H (полуперенос), C (перенос), V (переполнение).

Рассмотрим на примерах выполнение групп арифметических команд. Обратите внимание на установку признаков и переходы значений из положительных в отрицательные и наоборот.

```
; ————————— установка знака
;   команда                 результат
;                  аккумулятор   десят.знач.    флаги
   LD   A,0         0000 0000       0             —
   ADD  A,A         0000 0000       0             Z
   INC  A           0000 0001       1             —
   DEC  A           0000 0000       0             Z N
   DEC  A           1111 1111      -1             S H N
; ————————— переход положит. чисел в отрицательные через макс.
;   команда                 результат
;                  аккумулятор   десят.знач.    флаги
   LD   A,7Eh       0111 1110      126            —
   INC  A           0111 1111      127            —
   INC  A           1000 0000     -128           S H V
   INC  A           1000 0001     -127           S
; ————————— переход отрицат. чисел в положительные через макс.
;   команда                 результат
;                  аккумулятор   десят.знач.    флаги
   LD   A,81h       1000 0001     -127            —
   DEC  A           1000 0000     -128            S N
   DEC  A           0111 1111      127            H N V
   DEC  A           0111 1110      126            N
; ————————— переход положит. чисел в отрицательные через 0
;   команда                 результат
;                  аккумулятор   десят.знач.    флаги
   LD   A,1         0000 0001        1            —
   SUB  1        (1)0000 0000        0            Z N
   SUB  1           1111 1111      - 1            S H N C
   SUB  1           1111 1110      - 2            S N
; ————————— переход отрицат. чисел в положительные через 0
;   команда                 результат
;                  аккумулятор   десят.знач.    флаги
   LD   A,FFh       1111 1111      - 1            —
   ADD  A,1      (1)0000 0000        0            Z H C
   ADD  A,1         0000 0001        1            —
; ————————— изменение знака в аккумуляторе
;   команда                 результат
;                  аккумулятор   десят.знач.    флаги
   LD   A,7Eh       0111 1110      126            —
   NEG              1000 0010     -126           S H N C
   NEG              0111 1110      126             H N C
; ————————— изменение знака в аккумуляторе
;   команда                 результат
;                  аккумулятор   десят.знач.    флаги
   LD   A,FEh       1111 1110     - 2             —
   NEG              0000 0010       2             H N C
   NEG              1111 1110     - 2            S H N C
```

Как вы заметили, флаг переполнения V устанавливается при переходах 127 => -128 и -128 => 127, а флаг переноса C - при переходах "знак плюс <=> знак минус" через число ноль. При этом команды INC и DEC флаг C не изменяют.

Уменьшение или увеличение значения, хранящегося в памяти возможно посредством косвенной адресации через регистры HL, IX или IY. Например,

```
     LD   HL,0FCACh   ; загружаем адрес
     INC  (HL)        ; увеличиваем значение
     INC  (HL)        ; увеличиваем значение еще раз
```

Команда сложения или вычитания двух чисел, представленных в двоично-десятичном формате BCD, дает неправильный результат, поскольку она складывает их просто как двоичные значения. Для коррекции результата (приведения его снова в формат BCD) используется команда десятичной коррекции DAA.

Изучите примеры ее работы.

```
; ————————— десятичная коррекция
;   команда                    результат может означать:
;                  аккумулятор     шестн.  десят.знач.    флаги
   LD   A,06h      0000 0110        6         6            —
   ADD  A,11h      0001 0111       17        17            —
   DAA             0001 0111       17        17            P
; ————————— десятичная коррекция
;   команда                    результат может означать:
;                  аккумулятор     шестн.  десят.знач.    флаги
   LD   A,36h      0011 0110       36        36            —
   ADD  A,24h      0101 1010       5A         -            —
   DAA             0110 0000       60        60          H P
; ————————— десятичная коррекция
;   команда                    результат может означать:
;                  аккумулятор     шестн.  десят.знач.    флаги
   LD   A,72h      0111 0010       72        72            —
   ADD  A,63h      1101 0101       D5         -          S P
   DAA             0011 0101    (1)35     (1)35          P C
```

В последнем примере во флаге C появился старший разряд результата (бит сотни).

Для иллюстрации работы арифметических команд приведем программы умножения и деления восьмиразрядных чисел. В них использованы команды, которые мы изучим чуть позже.

```
      MSX.M-80 1.00 01-Apr-85 PAGE 1
                              .Z80
; умножение first * second
0000'   3A 0010'              LD   A,(first)  ; A <= first
0003'   47                    LD   B,A        ; B <= first
0004'   05                    DEC  B          ; B <= first-1
0005'   3A 0011'              LD   A,(second) ; A <= second
0008'   57                    LD   D,A        ; D <= second
0009'   82                    ADD  A,D        ; сложение ╢┐
000A'   10 FD                 DJNZ $-1        ; цикл по B ┘
000C'   32 0012'              LD   (result),A ; запись результата
000F'   C9                    RET
0010'   0C          first:    DB   12
0011'   08          second:   DB   8
0012'               result:   DS   1
                              END
```

Программа деления числа, находящегося в акккумуляторе, на число в регистре B. На выходе в регистре C должно находиться частное, а в регистре A - остаток от деления.

```
                  Z80-Assembler   Page:    1
                 ORG     9000h
9000 0E00        LD      c,0     ; частное равно 0
9002 0C   L01:   INC     c       ; частное <= частное + 1
9003 90          SUB     b       ; вычитаем из a - b
9004 30FC        JR      nc,L01  ; если нет переноса,
                                 ; то повторить
9006 80          ADD     a,b     ; добавить к a - b
9007 0D          DEC     c       ; уменьшить частное
9008 C9          RET             ; возврат
                 END
```

Флаг переноса в данном примере выставляется в том случае, если мы вычитаем из регистра A регистр B и при этом содержимое регистра B больше содержимого регистра A.

п.3. Работа с шестнадцатиразрядными числами

В системе команд микропроцессора есть команды ADD, позволяющие сложить два 16-разрядных числа. Одно из них должно быть записано в регистровую пару HL, IX, IY, а другое - в регистровую пару HL, DE, BC или SP. Результат сложения помещается в регистровую пару HL, IX или IY.

Так же, как и для 8-разрядных операндов, существуют команды сложения 16-разрядных чисел с битом признака C.

Одной из команд, позволяющих облегчить программирование на ассемблере Z80, является команда вычитания из регистровой пары HL 16-разрядного операнда - SBC.

Для 16-разрядных регистров (регистровых пар) есть команды

уменьшения/увеличения на единицу DEC и INC.

Флаги выставляют практически только две команды - ADC и SBC. Команды сложения устанавливают только флаг переноса.

Рассмотрим на примерах выполнение групп арифметических команд. Обратите внимание на установку признаков и переходы значений из положительных в отрицательные и наоборот; отличия от команд работы с восьмиразрядными числами.

```
; ———————— установка знака
;   команда                 результат
;                  шест. HL      десят.знач.    флаги
   LD   HL,0       0000            0             —
   ADD  HL,HL      0000            0             —
   INC  HL         0001            1             —
   DEC  HL         0000            0             —
   DEC  HL         FFFF           -1             —
; ———————— переход положит. чисел в отрицательные через макс.
;   команда                 результат
;                  шест. HL      десят.знач.    флаги
   LD   HL,7FFEh    7FFE          32766           —
   INC  HL          7FFF          32767           —
   INC  HL          8000         -32768           —
   INC  HL          8001         -32767           —
; ———————— переход отрицат. чисел в положительные через макс.
;   команда                 результат
;                  шест. HL      десят.знач.    флаги
   LD   HL,8001h   8001          -32767           —
   DEC  HL         8000          -32768           —
   DEC  HL         7FFF           32767           —
   DEC  HL         7FFE           32766           —
; ———————— переход положит. чисел в отрицательные через 0
;   команда                 результат
;                   шест. HL      десят.знач.    флаги
   LD   HL,1       0001            1             —
   LD   BC,1                                     —
   SBC  HL,BC      0000            0             Z N
   SBC  HL,BC      FFFF          - 1             S H N C
   SBC  HL,BC      FFFD          - 3             S N
; ———————— переход отрицат. чисел в положительные через 0
;   команда                 результат
;                  шест. HL      десят.знач.    флаги
   LD   HL,FFFFh   FFFF          - 1             —
   LD   BC,1                                     —
   ADD  HL,BC      0000            0             H C
   ADD  HL,BC      0001            1             —
```

Микропроцессор Z80 не имеет команд умножения и деления для двухбайтных значений, поэтому приходится их моделировать группами простых команд. Ниже приводятся примеры программ умножения и деления шестнадцатиразрядных чисел.

Первая программа - умножение шестнадцатиразрядных чисел, содержащихся в регистрах DE и HL, с результатом в четырех регистрах HLBC. Используется обычный алгоритм - сдвиги и деление.

```
     MSX.M-80 1.00 01-Apr-85 PAGE 1
; Умножение: [de] * [bc]  =>  [HLbc]
                          .Z80
  0000'   21 0000 mult16: LD   HL,0000  ; чистка HL
  0003'   78              LD   A,B      ; A <= B
  0004'   06 11           LD   B,11h    ; цикл 16 раз
  0006'   18 07           JR   Loop
  0008'   30 01     Next: JR   NC,Jump
  000A'   19              ADD  HL,DE    ; если есть бит - сложить
  000B'   CB 1C     Jump: RR   H        ; сдвиги HL
  000D'   CB 1D           RR   L
  000F'   1F        Loop: RRA
  0010'   CB 19           RR   C
  0012'   10 F4           DJNZ Next     ; повторить все
  0014'   47              LD   B,A
  0015'   C9              RET
```

Вторая программа - деление содержимого регистров BC на DE с частным в регистрах BC и остатком - в регистрах HL.

```
     MSX.M-80 1.00 01-Apr-85 PAGE 1
; ---            [BC]  %  [DE] => [BC]
; ---            [BC] MOD [DE] => [HL]
                           .Z80
  0000'   21 0000          LD   HL,0     ; чистим HL
  0003'   78               LD   A,B      ; A <= B
  0004'   06 10            LD   B,10h    ; цикл 16 раз
  0006'   CB 11            RL   C
  0008'   17               RLA
  0009'   CB 15    Shift:  RL   L
  000B'   CB 14            RL   H
  000D'   38 0D            JR   C,Again  ; переход по C
  000F'   ED 52            SBC  HL,DE
  0011'   30 01            JR   NC,Round
  0013'   19               ADD  HL,DE
  0014'   3F       Round:  CCF
  0015'   CB 11    Next:   RL   C
  0017'   17               RLA
  0018'   10 EF            DJNZ Shift
  001A'   47               LD   B,A
  001B'   C9               RET           ; выход в DOS
  001C'   B7       Again:  OR   A
  001D'   ED 52            SBC  HL,DE
  001F'   18 F4            JR   Next
                           END
```

7. Логические команды

Логические операции, подобно основным арифметическим операциям сложения и вычитания, выполняются над содержимым аккумулятора и данными из регистра, ячейки памяти или операндами, заданными непосредственно.

Основное отличие от арифметических заключается в том, что логические операции выполняются поразрядно, т.е. логическую операцию можно разбить на восемь независимых операций над соответствующими битами байтов-операндов. В результате операции формируется новое значение аккумулятора и устанавливаются флаги регистра F.

Микропроцессор Z-80 имеет следующие логические команды:

AND - логическое И (конъюнкция): бит результата устанавливается в единицу, если оба соответствующих бита аргумента равны 1; иначе - 0;

OR - логическое ИЛИ (дизъюнкция): бит результата равен 1, если хотя бы у одного аргумента соответствующий бит равен 1;

CPL - логическое НЕ (отрицание): если бит аргумента равен 1, то бит результата 0; если бит аргумента равен 0, то соответствующий бит результата - 1;

XOR - исключающее ИЛИ (не эквивалентность): результат 1, если только у одного аргумента соответствующий бит равен 1; иначе ноль.

Ниже приведена таблица истинности для логических операций:

X	Y	X AND Y	X OR Y	CPL X	X XOR Y
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Команда AND обычно применяется для выделения, обнуления или проверки значения определенных битов аккумулятора. При этом второй операнд используется как маска. Рассмотрим примеры.

```
   A:  1011 1100      A: 1011 0110      A: 0101 1100
   S:  1101 1011      S: 1111 0000      S: 1111 1111
   ---------------    ---------------   ---------------
AND:  1001 1000   AND: 1011 0000   AND: 0101 1100
```

Команда OR обычно применяется для того, чтобы установить в 1 определенные разряды аккумулятора, чтобы собрать содержимое аккумулятора из нескольких нужных полей, для выяснения равенства содержимого регистра или двойного регистра нулю.

```
A:  1011 1100      A: 1011 0110      A: 0101 1100
S:  1101 1011      S: 1111 0000      S: 0101 1100
   ──────────        ──────────        ──────────
OR: 1111 1111     OR: 1111 0110     OR: 0101 1100
```

Например, для проверки регистровой пары BC на ноль можно написать:

```
        LD   A,B   ; копируем B в A
        or   C     ; ноль, если ни в A, ни в C нет единиц
```

Команда CPL позволяет изменить значение каждого бита аккумулятора на обратное (инвертировать аккумулятор). Например,

```
A:  1011 1100      A: 1011 0110      A: 0101 1100
   ──────────        ──────────        ──────────
CPL: 0100 0011    CPL: 0100 1001    CPL: 1010 0011
```

Команда XOR позволяет выборочно инвертировать биты аккумутора, очистить содержимое аккумулятора.

Например, команда XOR 1, выполняемая многократно, формирует чередующуюся последовательность 0 и 1 в младшем разряде аккумулятора, а команда XOR A очищает аккумулятор.

```
A:  1011 1100      A: 1011 0110      A: 0101 1100
S:  1101 1011      S: 1111 0000      S: 0101 1100
   ──────────        ──────────        ──────────
XOR: 0110 0111    XOR: 0100 0110    XOR: 0000 0000
```

После выполнения рассмотренных команд логической обработки двух операндов значение признаков C и N регистра признаков F всегда равны 0.

Команда CP позволяет сравнить два операнда. Сравнение происходит посредством "воображаемого" вычитания из первого операнда, хранящегося в аккумуляторе, второго операнда. Содержимое аккумулятора при этом не изменяется, зато устанавливаются или сбрасываются соответствующие флаги регистра F.

Если в результате операции сравнения окажется, что операнды равны, то устанавливается признак нуля Z, если же значение операнда, хранящегося в аккумуляторе, меньше значения второго операнда, то устанавливается флаг S.

С помощью команды CCF можно изменить значение бита признака переноса C на противоположное, т.е. инвертировать флаг переноса C. Команда SCF позволяет установить значение признака переноса в 1.

В качестве примера приведем программу умножения числа, находящегося в HL, на число, большее нуля в DE; при этом на выходе в двойном регистре HL будет произведение.

```
                  Z80-Assembler   Page:    1
                     ORG     9000h
9000 44              LD      B,H     ;  BC <= HL
9001 4D              LD      C,L     ;
9002 1801            JR      L02     ; уменьшить DE
9004 09       L01:   ADD     HL,BC   ; добавить к HL, BC
9005 1B       L02:   DEC     DE      ; уменьшаем DE
9006 7A              LD      A,D     ; если DE <> 0, то
9007 B3              OR      E
9008 20FA            JR      NZ,L01  ; повторяем
900A C9              RET             ; возврат
                     END
```

Обратите внимание на то, что программа будет работать неверно, если в DE будет число, равное или меньшее нуля. В качестве упражнения попробуйте написать программу умножения любых чисел (после изучения следующего параграфа).

Кроме перечисленных выше команд Z-80 имеет команды установки, сброса и проверки состояния одного бита в байте. Команда BIT проверяет состояние заданного бита, SET устанавливает бит в единицу, RES - сбрасывает бит в ноль.

Биты операнда нумеруются следующим образом: 76543210. Рассмотрим пример.

команда		результат		
		аккумулятор	шестн.знач.	флаги
LD	A,B9h	1011 1001	B9	—
BIT	7,A	1011 1001	B9	S H
BIT	6,A	1011 1001	B9	Z H P
RES	0,A	1011 1000	B8	Z H P
SET	2,A	1011 1100	BC	Z H P
RES	7,A	0011 1100	3C	Z H P
BIT	2,A	0011 1100	3C	H

8. Команды перехода и условного перехода

Эти команды играют особую роль в организации выполнения программ в микроЭВМ. Пока в программе не встретится команда этой группы, счетчик команд PC постоянно увеличивается на длину команды, и микропроцессор выполняет команду за командой в порядке их расположения в памяти.

Порядок выполнения программы может быть изменен, если занести в регистр счетчика команд микропроцессора код адреса, отличающийся от адреса очередной команды. Это вызовет передачу управления другой части программы.

Такая передача управления (переход) может быть выполнена с помощью трехбайтовой команды безусловного перехода JP адрес.

Как только эта команда встретится в программе, в регистр счетчика команд PC микропроцессора запишется значение указанного адреса. Таким образом, следующей командой, которую будет выполнять микропроцессор вслед за командой JP, будет команда, код операции которой записан в ячейке с этим адресом.

Безусловную передачу управления можно произвести также при помощи команд JP (HL), JP (IX) , JP (IY), в результате выполнения которых происходит передача управления по адресу, хранящемуся соответственно в регистровой паре HL, IX или IY.

Кроме команды безусловного перехода микропроцессор имеет трехбайтовые команды условного перехода. При появлении команды условного перехода передача управления по адресу, указанному в команде, происходит только в случае выполнения определенного условия.

Условия, с которыми оперируют команды условной передачи управления, определяются состоянием битов (разрядов) регистра признаков F:

Мнемоника	Условие	Флаг	Код CCC
NZ (Not Zero)	- ненулевой результат	Z =0	000
Z (Zero)	- нулевой результат	Z =1	001
NC (No Carry)	- отсутствие переноса	C =0	010
C (Carry)	- перенос	C =1	011
PO (Parity Odd)	- нечетный результат	P =0	100
PE (Parity Even)	- четный результат	P =1	101
P (Plus)	- число положительное	S =0	110
M (Minus)	- число отрицательное	S =1	111

Команда условного перехода может иметь, например, такой вид:

```
JP   NC,Again.
```

Кроме команд перехода, в которых адрес указан непосредственно - "длинного" перехода, существуют команды "короткого" перехода, в которых адрес указан как смещение к текущему адресу, т.е. относительный адрес. Эти команды позволяют осуществить переход вперед/назад на -126..+129 ячеек памяти и используются для написания перемещаемых программ.

Хотя один байт обычно задает смещение -128..+127, здесь нужно учитывать, что счетчик команд увеличивается на длину самой команды перехода (2) до прибавления смещения, указанного в команде. Мнемоникой этих двухбайтных команд является JR. Если используется символическое имя, ассемблер сам определит смещение и запишет его в код. Нужно только следить, чтобы переход не оказался слишком большим.

Таким образом, можно написать команду JR Label вместо JP Label, если расстояние до метки Label небольшое.

Команды относительного перехода также могут быть условными. Однако обратите внимание, что допускаются только условия C, NC, Z, NZ.

И последняя команда перехода - это команда цикла

DJNZ смещение.

Суть этой команды следующая:

1) DEC B ; уменьшить регистр B
2) JR NZ,$+смещение ; если B<>0, сделать относительный переход.

Теперь, как мы обещали, попробуем написать программу (или подпрограмму) задержки (или цикла, если в тело вставить какие-либо команды). Для этого будем использовать регистровую пару BC (также можно использовать DE или HL).

```
      ORG     9000h
; установили начальный адрес компиляции
       LD      BC,8000h ; загрузили в BC 8000h
beg:   DEC     BC       ; BC=BC-1
       LD      A,B      ; проверяем
       OR      C        ; если BC<>0
       JR      NZ,beg   ; перейти на beg
       RET              ; иначе возврат
; конец программы
       END
```

Рассмотрим подробнее, как работает эта программа. По команде LD BC, 8000h в регистровую пару BC загружается некоторое число (в данном случае 8000h). Команда DEC BC содержимое регистровой пары BC уменьшает на 1. Затем в аккумулятор пересылается содержимое регистра B и производится операция логического сложения с содержимым регистра C этой регистровой пары.

Если в регистровой паре BC код еще не стал равным 0, то после выполнения этой команды будет установлен признак NZ и выполнится команда условного перехода JR NZ,beg к началу цикла, после чего все действия повторяются.

При этом программисты говорят, что в программе организован цикл. Выход из него возможен только тогда, когда в результате выполнения команды DEC BC в регистровой паре BC окажется ноль.

Тогда работа подпрограммы закончится выполнением команды RET, и произойдет возврат к выполнению основной программы.

Вы, наверное, уже догадались, что временная задержка, обеспечиваемая этой подпрограммой, определяется, во-первых, временем, необходимым для однократного выполнения всех команд этой подпрограммы, и, во-вторых, содержимым регистровой пары BC. Последнее и определяет количество программных циклов.

Как же определить число, которое надо поместить в регистровую пару BC для задания временной задержки в 0.5 секунд?

Выполнение любой команды микропроцессора занимает строго определенное время. Поэтому, зная длительность выполнения каждой команды, можно вычислить общее время однократного выполнения данной подпрограммы. Оно составляет 9.6 мкс. Следовательно, для задания временной задержки в 0.5 сек. подпрограмма должна выполниться 0.5/(9.6*10^-6)=52080 раз.

В качестве примера приведем небольшую программу, которая ищет минимальное из двух чисел.

```
     MSX.M-80 1.00 01-Apr-85  PAGE 1
; --- Нахождение минимального из двух чисел
;     min( fst,sec) => res
                           .Z80
  0000'   3A 0010'         LD   A,(fst)
  0003'   47               LD   B,A       ; fst => B
  0004'   3A 0011'         LD   A,(sec)   ; sec => A
  0007'   B8               CP   B         ; sec ? fst
  0008'   FA 000C'         JP   M,minA    ; переход, если sec<fst
  000B'   78               LD   A,B       ; fst => A
  000C'   32 0012'  minA:  LD   (res),A   ; min => res
  000F'   C9               RET
  0010'   22        fst:   DB   34
  0011'   80        sec:   DB   128
  0012'             res:   DS   1,0
                           END
```

Приведем программу преобразования числа, находящегося в регистре A, в строку символов в коде ASCII. Адрес строки должен находиться в регистровой паре DE.

```
       MSX.M-80 1.00     01-Apr-85  PAGE 1
                        .Z80
; Вход:  число в A
; Выход: строка по адресу [DE] в коде ASCII
0000'   06 2F  DaaDig:   LD    B,'0'-1
0002'   04               INC   B
0003'   D6 0A            SUB   10
0005'   30 FB            JR    NC,DaaDig+2
0007'   C6 3A            ADD   A,'9'+1
0009'   EB               EX    DE,HL
000A'   70               LD    (HL),B
000B'   23               INC   HL
000C'   77               LD    (HL),A
000D'   2B               DEC   HL
000E'   EB               EX    DE,HL
000F'   C9               RET
                         END
```

Обратите внимание на то, что записи '0' и '9' означают коды знаков "0" и "9". Попробуйте разобраться, как работает программа, и подумайте над вопросом - будет ли она работать с отрицательными числами или с числами, большими чем 99.

9. Команды сдвига

Команды сдвига позволяют сдвинуть биты одного регистра или байта памяти влево или вправо на один бит.

Имеются следующие типы команд:

- арифметический сдвиг влево (SLA - Shift Left Arithmetical);
- арифметический и логический сдвиг вправо (SRA - Shift Right Aritmetical, SRL - Shift Right Logical);
- циклический сдвиг (RLCA, RLC, RRCA, RRC);
- циклический сдвиг через флаг C (RLA, RRA);
- перестановка полубайт (RLD, RRD).

В командах SLA и SRL освободившийся разряд заполняется нулевым битом. В команде SRA тиражируется знаковый бит.

Схема работы команды SLA:

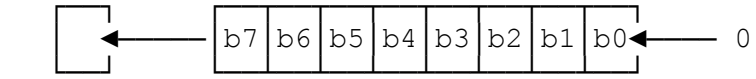


Схема работы команды SRA:

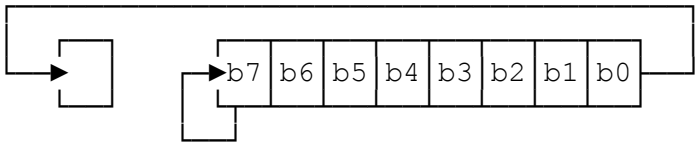
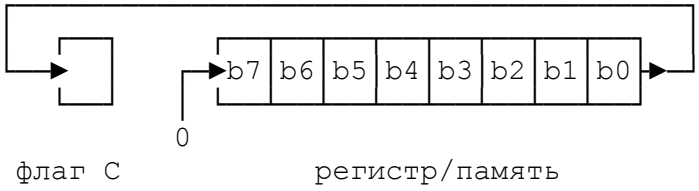


Схема работы команды SRL:



Например, если в регистре B было двоичное значение 00111011, то после выполнения команды SLA B в регистре B появится значение 01110110.

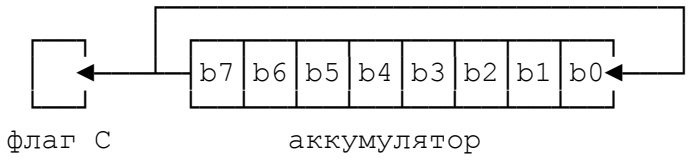
Арифметический сдвиг влево SLA можно использовать для умножения на степень двойки, а арифметический сдвиг вправо SRA - для деления на два без остатка (нацело).

Приведем листинг программы, делящей содержимое регистра A нацело на 8. Она должна вызываться из программы на языке MSX-BASIC с помощью функции USR.

```
'divide on 8'     Z80-Assembler   Page:    1
                      TITLE   'divide on 8'
                      ORG     9000h
521F =        getA    EQU     521Fh
2F99 =        outHL   EQU     2F99h
              ; === вход из USR
9000 CD1F52           CALL    getA    ; записать аргумент в A
              ; === деление на 8 нацело
9003 CB2F             SRA     A       ; делим на 2
9005 CB2F             SRA     A       ; еще раз
9007 CB2F             SRA     A       ; и еще
              ; === возврат
9009 2600             LD      H,0     ;
900B 6F               LD      L,A     ;
900C C3992F           JP      outHL   ; возвращаем результат
                      END
```

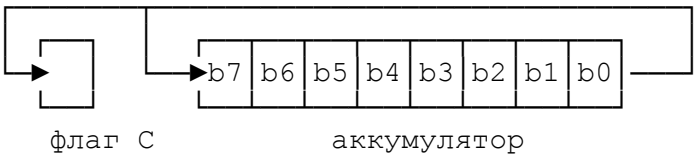
Команды циклического сдвига сдвигают содержимое регистра или байта памяти влево или вправо на один бит. При этом выдвинувшийся за разрядную сетку бит не теряется, а переносится на первое место с другого конца байта.

Схема работы команды RLCA:



Команда RLC выполняется аналогично над регистром или косвенно адресуемой памятью. Если в аккумуляторе было записано число 10110100, то после выполнения команды RLCA в нем появится значение 01101001.

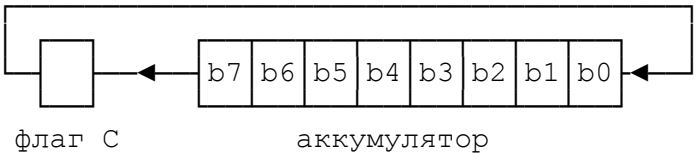
Схема работы команды RRCA:



Команда RRC выполняется аналогично над регистром или памятью.

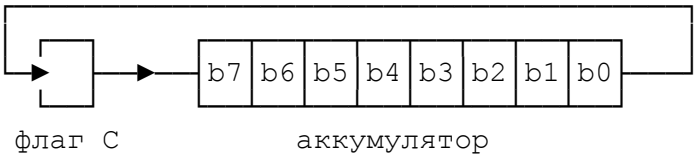
Кроме этого можно использовать команды циклического сдвига влево или вправо через флаг C.

Схема работы команды RLA:



Команда RL выполняется аналогично над регистром или памятью.

Схема работы команды RRA:



Команда RR выполняется аналогично над регистром или памятью.

Ниже приводится листинг программы преобразования однобайтного числа в двоично-десятичном коде (BCD) в однобайтное двоичное число. Аргумент программа берет в ячейке A000h, а результат записывает в A0001h. Примеры выполняемых преобразований:

```
  двоично-десятичный код        двоичный код
     10 = 0001 0000    =>   0A = 0000 1010
     47 = 0100 0111    =>   2F = 0010 1111
     87 = 1000 0111    =>   57 = 0101 0111
```

```
'conversion'      Z80-Assembler   Page:    1
                        TITLE   'conversion'
  ; === преобразование BCD-числа в двоичное число
A000 =        bcdarg    EQU     0A000h
A001 =        hexres    EQU     0A001h
                        ORG     9000h
              ; === берем аргумент
9000 3A00A0             LD      A,(bcdarg); записать однобайтный
                                          ; параметр в A
9003 47                 LD      B,A       ; скопировали в B
              ; === меняем десятич. цифры местами
9004 07                 RLCA              ; 4 циклических сдвига
9005 07                 RLCA              ; аккумулятора влево,
9006 07                 RLCA              ; можно и вправо
9007 07                 RLCA              ;
              ; === оставляем только десятки
9008 E60F               AND     0Fh       ; десятки - в младший
                                          ; полубайт
900A 4F                 LD      C,A       ; копируем в C
              ; === умножаем десятки на десять
900B CB27               SLA     A         ; умножение на 8
900D CB27               SLA     A         ;
900F CB27               SLA     A         ;
9011 81                 ADD     A,C       ; добавляем еще два
9012 81                 ADD     A,C       ;
9013 4F                 LD      C,A       ; в C - преобр.десятки
              ; === добавляем единицы
9014 78                 LD      A,B       ; восстанавливаем арг.
9015 E60F               AND     0Fh       ; оставляем единицы
9017 81                 ADD     A,C       ; складыв. с десятками
              ; === возврат результата
9018 3201A0             LD      (hexres),A; возвращаем результат
901B C9                 RET
                        END
```

Иногда оказываются полезными команды циклической перестановки полубайтов влево (RLD) и вправо (RRD). Команды используют младшие 4 бита аккумулятора и байт, адрес которого должен быть записан в регистровую пару HL.

Схема работы команды RLD:

аккумулятор память, адресуемая (HL)

Схема работы команды RRD:

аккумулятор память, адресуемая (HL)

Изучите приведенный ниже листинг программы перевода BCD-числа в двоичный код с использованием команды RLD.

```
'Conversion-2'    Z80-Assembler   Page:    1
                        TITLE   'Conversion-2'
              ; === преобразование BCD-числа в двоичное число
A000 =        bcdarg    EQU     0A000h
A001 =        hexres    EQU     0A001h
                        ORG     9000h
              ; === берем аргумент
9000 2100A0             LD      HL,bcdarg ; берем адрес аргумента
9003 46                 LD      B,(HL)    ; записать аргумент в B
9004 AF                 XOR     A         ; чистим A
              ; === десятки - в младший полубайт, с порчей арг.
9005 ED6F               RLD               ; десятки - в младший
                                          ; полубайт A
9007 4F                 LD      C,A       ; копируем в C
              ; === умножаем десятки на десять
9008 CB27               SLA     A         ; умножение на 8
900A CB27               SLA     A         ;
900C CB27               SLA     A         ;
900E 81                 ADD     A,C       ; добавляем еще два
900F 81                 ADD     A,C       ;
9010 4F                 LD      C,A       ; в C - преобр.десятки
              ; === добавляем единицы
9011 78                 LD      A,B       ; восстанавливаем арг.
9012 E60F               AND     0Fh       ; оставляем единицы
9014 81                 ADD     A,C       ; складыв. с десятками
              ; === возврат результата
9015 3201A0             LD      (hexres),A; возвращаем результат
9018 C9                 RET
                        END
```

10. Пересылки блока данных

При помощи команд пересылки блока данных можно скопировать (переслать) содержимое участка памяти в другое место как пошагово, так и в автоматическом режиме.

Перед выполнением этих команд необходимо загрузить в регистровые пары HL, DE и BC необходимые параметры. В HL записывается адрес начала блока, в DE - адрес памяти, куда необходимо переслать блок, в BC - длину блока.

Имеются следующие команды:

a) LDI - пересылка байта с инкрементом выполняется так:
   1) LD (DE),(HL)
   2) INC HL
   3) INC DE
   4) DEC BC

b) LDIR - пересылка блока с автоинкрементом:
   1) LD (DE),(HL)
   2) INC HL
   3) INC DE
   4) DEC BC
   5) если BC<>0, то перейти на 1

c) LDD - пересылка байта с декрементом:
   1) LD (DE),(HL)
   2) DEC HL
   3) DEC DE
   4) DEC BC

d) LDDR - пересылка блока с автодекрементом:
   1) LD (DE),(HL)
   2) DEC HL
   3) DEC DE
   4) DEC BC
   5) если BC<>0, то перейти на 1.

Например, для сохранения текущего состояния матрицы клавиатуры (см. Рабочую область MSX) можно написать:

```
       ORG     9000h
       LD      HL,0FBE5h       ; адрес матрицы клавиатуры
       LD      DE,kon          ; куда переписать
       LD      BC,11           ; длина матрицы
       LDIR                    ; переписываем
       RET                     ; возврат
kon:   DS      11,0
       END
```

Другой пример: та же задача, но конечным адресом матрицы должен быть начальный адрес нашей программы:

```
start  EQU     9000h           ; константа start = 9000h
       ORG     start
       LD      HL,0FBFFh       ; адрес конца матрицы клавиатуры
       LD      DE,start-1      ; адрес конца, куда переписать
       LD      BC,11           ; длина матрицы
       LDDR                    ; переписываем
       RET                     ; возврат
       END
```

Предлагаем Вам написать подпрограмму, которая обнуляет участок памяти ЭВМ. Для нее требуются следующие исходные данные:

1) начальный адрес памяти;
2) длина участка памяти.

Будем считать, что при обращении к нашей подпрограмме эти данные заносятся соответственно в регистры HL и BC. Попробуйте написать эту подпрограмму самостоятельно. Ниже мы приводим два варианта решения этой задачи.

```
                 Z80-Assembler   Page:    1
0000 3600  fillm: LD      (HL),0  ; обнулить содерж. по адр.(HL)
0002 23           INC     HL      ; следующий адрес
0003 0B           DEC     BC      ; уменьшить длину
0004 78           LD      a,b     ; проверить длина <> 0 ?
0005 B1           OR      c
0006 20F8         JR      nz,fillm; если нет, то повторить
0008 C9           RET             ; иначе возврат
                  END
```

Недостаток этой программы в том, что она выполняется в течении некоторого времени, которое зависит от длины участка памяти. Другой вариант этой же программы позволяет выполнять те же действия, но за более короткое и фиксированное время:

```
                  Z80-Assembler   Page:    1
0000 3600  fillm: LD      (hL),0  ; обнулить первый байт
0002 54           LD      d,h     ; загрузить в DE след.адрес
0003 5D           LD      e,L
0004 13           INC     DE
0005 0B           DEC     BC      ; уменьшить длину
0006 EDB0         LDir            ; обнулить участок памяти
0008 C9           RET             ; возврат
                  END
```

Теперь давайте немного усложним нашу задачу. Пусть исходными данными являются:

1) начальный адрес участка памяти (HL);
2) конечный адрес участка памяти (DE);
3) константа, которой надо заполнить участок (B);

```
                  Z80-Assembler   Page:    1
0000 70  fillmc:LD      (HL),b  ; записать данные в первый адрес
0001 23         INC     HL      ; подготовить следующий адрес
0002 7C         LD      a,h     ; сравнить старш. байты текущего
0003 92         SUB     d       ; адреса и адреса конца участка
0004 20FA       JR      nz,fillmc; если они <>, то повторить
0006 7D         LD      a,l     ; сравнить младш. байты текущего
0007 BB         CP      e       ; адреса и адреса конца участка
0008 20F6       JR      nz,fillmc; если они <>, то повторить
000A 70         LD      (HL),b  ; обнулить последний адрес
000B C9         RET             ; вернуться
                END
```

Теперь разберем немного подробнее, как работает эта программа. Так как нам предстоит запись данных в последовательность ячеек, то организуем циклическую работу программы. В каждом цикле будем заполнять одну ячейку, а затем подготавливать адрес очередной ячейки памяти для ее заполнения в следующем цикле. Для этого в цикле можно использовать команду INC HL, увеличивающую каждый раз на 1 содержимое регистровой пары HL.

Работа программы должна прекратиться после заполнения последней ячейки памяти заданного участка. В ходе выполнения каждого цикла программы необходимо следить, чтобы постоянно увеличивающееся значение адреса в регистровой паре HL не превысило значения конечного адреса в регистровой паре DE.

Другой вариант программы выглядит так:

```
                  Z80-Assembler   Page:    1
0000 70  fillmc:LD      (HL),b  ; записать данные в первый адрес
0001 E5         PUSH    HL      ; сохранить в стеке первый адрес
0002 37         SCF             ; сбросить бит переноса рег. F
0003 3F         CCF
0004 ED52       SBC     HL,DE   ; получить длину участка
0006 44         LD      b,h     ; переслать ее в BC
0007 4D         LD      c,l
0008 E1         POP     HL      ; считать начальн. адрес участка
0009 54         LD      d,h     ; переслать его в DE
000A 5D         LD      e,l
000B 13         INC     DE      ; увеличить DE, т.е. след.адрес
000C EDB0       LDIR            ; заполнить константой блок
000E C9         RET             ; вернуться
                END
```

11. Команды поиска

Следующей группой команд, которые мы рассмотрим, будет группа команд поиска. Они предназначены для поиска в памяти заданного в аккумуляторе значения. Имеются следующие команды:

a) CPI - сравнение A с байтом памяти с инкрементом:
   1) CP A,(HL)
   2) INC HL
   3) DEC BC

b) CPIR - сравнение A с блоком пошаговое с автоинкрементом:
   1) CP A,(HL)
   2) INC HL
   3) DEC BC
   4) если BC<>0 и A<>(HL), то перейти на 1

c) CPD - сравнение A с байтом с декрементом:
   1) CP A,(HL)
   2) DEC HL
   3) DEC BC

d) CPDR - сравнение A с блоком с автодекрементом:
   1) CP A,(HL)
   2) DEC HL
   3) DEC BC
   4) если BC<>0 и A<>(HL), то перейти на 1.

Так же как и в командах пересылки блока перед выполнением этих команд необходимо занести в регистры нужные параметры.

В регистре A должно находиться число, которое мы хотим искать, в HL - начальный адрес участка памяти, в BC - длина участка.

Если число найдется, будет установлен флаг Z. Если BC уменьшится в процессе поиска до нуля, будет сброшен флаг P/V (в противном случае он будет установлен).

Например, на участке памяти с A000h по DE77h (исключительно) мы хотим найти и заменить все числа 32 на число 34. Один из возможных вариантов программы:

```
       ORG     9000h
       LD      HL,0A000h         ; адрес начала блока
       LD      BC,0DE77h-0A000h  ; длина блока поиска
Next:  LD      A,32              ; что искать ?
       CPIR                     ; поиск
       RET     NZ                ; возврат, если не нашли
       DEC     HL                ; возврат на один байт назад
       LD      (HL),34           ; запись по адресу нового числа
       INC     HL
       LD      A,B               ; BC = 0 ?
       OR      C
       JR      NZ,Next           ; если нет, повторяем
       RET                       ; возврат
       END
```

12. Подпрограммы и прерывания

При написании программ обычно можно выделить одинаковые последовательности команд, часто встречающиеся в разных частях программы. Для того, чтобы многократно не переписывать такие последовательности команд, их объединяют в так называемые подпрограммы. В любой части основной программы программист может вставить трехбайтовую команду безусловного вызова подпрограммы CALL adr , во втором и третьем байте которой указывается адрес вызываемой подпрограммы.

Выполнение команды CALL adr начинается с побайтовой засылки в стек адреса следующей после этой команды ячейки памяти. Этот адрес называется адресом возврата из подпрограммы. Он необходим для того, чтобы по окончании выполнения подпрограммы вернуться к продолжению основной программы.

После записи в стек адреса возврата из подпрограммы в счетчик команд PC микропроцессора загружается величина adr, т.е. адрес первой команды вызываемой подпрограммы. Таким образом, управление из основной программы передается на вызываемую подпрограмму.

Выполнение подпрограммы обычно заканчивается командой возврата из подпрограммы, например, однобайтовой командой безусловного возврата из подпрограммы RET . При этом содержимое верхушки стека, т.е. адрес возврата из подпрограммы, пересылается из стека в счетчик команд PC микропроцессора, и управление вновь передается основной программе.

Ниже приводится пример программы, вызывающей подпрограмму умножения содержимого двойного регистра BC на DE.

```
                  Z80-Assembler   Page:    1
                    ORG     9000h
9000 018400         LD      BC,132        ; загрузка арг.
9003 118401         LD      DE,388        ; загрузка арг.
9006 CD0D90         CALL    mpy           ; вызов подпрогр.
9009 2200A0         LD      (0A000h),HL   ; запись результата
900C C9             RET                   ; возврат
    ; ------------------------
    ; умножение BC на DE
    ; результат - в HL
    ;
900D 210000 mpy:    LD      HL,0000       ; чистим HL
9010 19     next:   ADD     HL,DE         ; прибавляем DE
9011 0B             DEC     BC            ; уменьшаем BC
9012 78             LD      A,B           ; проверяем на 0
9013 B1             OR      C
9014 20FA           JR      NZ,next       ; повторить
9016 C9             RET                   ; возврат
                    END
```

Кроме команды безусловного вызова и возврата из подпрограммы, в

системе команд имеется восемь команд условного вызова подпрограммы и восемь команд условного возврата из подпрограмм, действие которых определяется так же, как и у команд условной передачи управления, состоянием регистра признаков F. Если условие для выполнения команды отсуствует, то вызов подпрограммы или возврат из нее не выполняются.

Кроме трехбайтовой команды безусловного вызова подпрограммы CALL adr, в системе команд микропроцессора имеется восемь однобайтовых команд RST 0 - RST 7 вызова подпрограмм, расположенных по фиксированному адресу. Появление в основной программе любой из этих команд вызывает запись в стек адреса возврата из подпрограммы и передачу управления на соответствующую ячейку памяти, где расположена первая команда подпрограммы.

Ниже приведена таблица соответствия между этими командами и шестнадцатеричными адресами ячеек памяти, куда передается управление при их выполнении:

Команда	Адрес начала подпрограммы	Команда	Адрес начала подпрограммы
RST 0	0000	RST 4	0020
RST 1	0008	RST 5	0028
RST 2	0010	RST 6	0030
RST 3	0018	RST 7	0038

В мнемонике микропроцессора ZILOG-80 (в отличие от мнемоники INTEL 8080) команда записывается с указанием непосредственного адреса обращения к подпрограмме, например, RST 7 записывается как RST 38h .

К группе команд работы с подпрограммами относятся еще две команды возврата из маскируемого и немаскируемого прерываний: RETI и RETN.

В программе на языке ассемблера могут использоваться внешние переменные, то есть переменные, определенные вне данной программы. Внешние значения транслируются в двухбайтные величины (однобайтные не поддерживаются). Внешние переменные описываются директивой ассемблера EXT или EXTRN. Можно также отметить внешнюю переменную двумя символами "#" в конце ее имени.

Кроме этого, в программе могут быть определены глобальные имена, то есть имена, доступные извне данной программы (для той программы они являются внешними). Для указания, что имя является глобальным, используются директивы ENTRY или PUBLIC. Глобальное имя можно также обозначить двумя знаками ":" в конце имени.

Эти возможности удобно использовать для организации связей с программами, написанными на языке С.

В качестве примера рассмотрим программу, устанавливающую позицию курсора на текстовом или графическом экране, и программу вывода одного символа на графический экран.

```
      MSX.M-80 1.00 01-Apr-85 PAGE 1
; Установка позиции курсора на текстовом или графическом
; экране. Координаты - в HL и DE
                        .Z80
                        PUBLIC Loc@
0000'   3A FCAF Loc@: LD   A,(0FCAFh)    ; тип экрана
0003'   FE 02         CP   2
0005'   38 0F         JR   C,Locate
      ; ----- размещение на графическом экране
0007'   22 FCB3       LD   (0FCB3h),HL
000A'   22 FCB7       LD   (0FCB7h),HL
000D'   ED 53 FCB5    LD   (0FCB5h),DE
0011'   ED 53 FCB9    LD   (0FCB9h),DE
0015'   C9            RET
      ; ----- размещение на текстовом экране
0016'   65    Locate: LD   H,L
0017'   6B            LD   L,E
0018'   F7            RST  30h
0019'   00            DEFB 0
001A'   00C6          DEFW 0C6h
001C'   C9            RET
                        END
```

Вывод одного символа на графический экран.

```
  MSX.M-80 1.00 01-Apr-85 PAGE 1
                        .Z80
                        PUBLIC putgc@
; --- выводится символ с кодом в аккумуляторе
0000'   F7    putgc@: RST 30h
0001'   00            DEFB 0
0002'   008D          DEFW 08Dh
0004'   C9            RET
                        END
```

13. Подпрограммы BIOS

В системе MSX имеется набор стандартных подпрограмм, использование которых иногда может значительно облегчить програмирование на языке ассемблера. Их английская аббревиатура - подпрограммы BIOS. При помощи подпрограмм BIOS можно работать с клавиатурой, программируемым звукогенератором, магнитофоном, видеопроцессором и другими устройствами.

Программы, вызывающие только подпрограммы BIOS и не работающие непосредственно с устройствами (например, при помощи портов ввода/вывода), смогут работать и на других версиях системы MSX.

п.1. Клавиатура

Для работы с клавиатурой применяются следующие подпрограммы BIOS: чтение статуса строки матрицы клавиатуры (SNSMAT, 0141H), обнаружение нажатия клавиш CTRL/STOP при отключенных прерываниях (BREAKX, 0B7h), проверка буфера клавиатуры (CHSNS, 09Ch), ввод символа из буфера клавиатуры (CHGET, 09Fh), чистка буфера клавиатуры (KILBUF, 156h), ввод строки (PINLIN, 0AEh и INLIN, 0B1h), ввод графического символа (CNVCHR, 0ABh) и другие.

В первом примере опрашивается матрица клавиатуры на предмет нажатия клавиши "Z". Выход из программы - по клавишам CTRL/STOP.

```
                Z80-Assembler   Page:    1
   ; === Проверка, нажата ли клавиша Z
00A2 =       CHPUT   EQU  00A2h      ; вывод символа
0141 =       SNSMAT  EQU  0141H      ; опрос матрицы клавиатуры
00B7 =       BREAKX  EQU  00B7h      ; нажато ли CTRL/STOP ?
                     ORG  9000h
9000 CDB700  Again:  CALL BREAKX     ; нажато ли CTRL/STOP
9003 D8              RET  C          ; возврат, если "да"
9004 3E04            LD   A,4        ; опрашиваем строку 4
9006 CD4101          CALL SNSMAT
9009 E620            AND  00100000b ; нажата ли клавиша Z ?
900B 20F3            JR   NZ,Again   ; если нет, ждем
900D 3E5A            LD   A,'Z'      ; иначе печатаем 'Z'
900F CDA200          CALL CHPUT
9012 18EC            JR   Again      ; все повторяем
                     END
```

Во втором примере производится чтение символа из буфера клавиатуры. Обратите внимание на действие содержимого ячейки REPCNT. Как и в первом примере, выход осуществляется при нажатии клавиш CTRL/STOP.

```
                    Z80-Assembler   Page:    1
009C =        CHSNS    EQU  09Ch     ; опрос буфера клавиатуры
009F =        ChGet    EQU  09Fh     ; ввод символа
00A2 =        ChPut    EQU  0A2h     ; вывод символа
0156 =        KilBuf   EQU  156h     ; чистка буфера
00B7 =        BreakX   EQU  0B7h     ; нажато ли CTRL/STOP ?
F3F7 =        REPCNT   EQU  0F3F7h
                       ORG  9000h
9000 CD9C00   Key:     CALL CHSNS     ; опрос буфера клавиатуры
9003 280A              JR   Z,Key1    ; если буфер пуст, вывод '.'
9005 3E01              LD   A,1         ; маленькая задержка до
9007 32F7F3            LD   (REPCNT),A  ; автоповторения клавиши
900A CD9F00            CALL ChGet     ; вводим символ из буфера
900D 1802              JR   Key2      ; выводим его на экран
900F 3E2E     Key1:    LD   A,'.'
9011 CDA200   Key2:    CALL ChPut     ; вывод символа
9014 CD5601            CALL KilBuf    ; чистка буфера
9017 CDB700            CALL BreakX    ; нажато ли CTRL/STOP ?
901A 30E4              JR   NC,Key    ; если нет, то повторить
901C C9                RET
                       END
```

Третий пример демонстрирует отличия в работе подпрограмм BIOS InLin и PinLin.

```
                    Z80-Assembler   Page:    1
00A2 =  ChPut    EQU  0A2h          ; вывод символа
00B1 =  InLin    EQU  0B1h          ; ввод строки
00AE =  PinLin   EQU  0AEh          ; ввод строки
0156 =  KilBuf   EQU  156h          ; чистка буфера
F55E =  Buf      EQU  0F55Eh        ; буфер
                 ORG  9000h
    ; === InLin
9000 CD5601      CALL KilBuf        ; чистка буфера
9003 212E90      LD   HL,PRMPT1     ; выводим подсказку
9006 CD2590      CALL PUTMSG
9009 CDB100      CALL InLin         ; вводим строку
900C 215EF5      LD   HL,Buf        ; выводим содержимое буфера
900F CD2590      CALL PUTMSG
    ; === PinLin
9012 CD5601      CALL KilBuf        ; чистка буфера
9015 213890      LD   HL,PRMPT2     ; выводим подсказку
9018 CD2590      CALL PUTMSG
901B CDAE00      CALL PinLin        ; вводим строку
901E 215EF5      LD   HL,Buf        ; выводим содержимое буфера
9021 CD2590      CALL PUTMSG
9024 C9          RET
    ; === Подпрограмма печати строки
9025 7E  PUTMSG: LD   A,(HL)        ; берем символ
9026 B7          OR   A             ; если код ноль, выход
9027 C8          RET  Z
```

```
9028 CDA200       CALL CHPUT        ; выводим один символ
902B 23           INC  HL
902C 18F7         JR   PUTMSG       ; повторяем снова
    ; === Данные
902E 0D0A496E PRMPT1: DB   0Dh,0Ah,'InLin: ',0
9032 4C696E3A
9036 2000
9038 0D0A5069 PRMPT2: DB   0Dh,0Ah,'PinLin:',0
903C 6E4C696E
9040 3A00
                  END
```

п.2. Звукогенератор

Для работы со звукогенератором используются следующие подпрограммы BIOS: инициализация PSG (GICINI, 90h), запись данных в регистр PSG (WRTPSG, 93h), чтение данных из регистра PSG (RDPSG, 96h), запуск звучания музыки (STRTMS, 99h), включение/выключение бита звукового порта (CHGSND, 135h) и другие.

В первом примере показана установка однотонного звучания в канале A.

```
                  Z80-Assembler   Page:    1
0093 = WRTPSG  EQU  93h          ; запись в регистр PSG
               ORG  9000h
9000 3E07      LD   A,7          ; выбор канала A
9002 1E3E      LD   E,00111110b
9004 CD9300    CALL WRTPSG       ; запись в регистр 7
9007 3E08      LD   A,8          ; установка громкости звука
9009 1E0F      LD   E,15
900B CD9300    CALL WRTPSG       ; запись в регистр 8
900E 3E00      LD   A,0          ; младшие биты частоты звука
9010 1EFE      LD   E,0FEh
9012 CD9300    CALL WRTPSG       ; запись в регистр 0
9015 3E01      LD   A,1          ; старшие биты частоты звука
9017 1E00      LD   E,0
9019 CD9300    CALL WRTPSG       ; запись в регистр 1
901C C9        RET
               END
```

Второй пример связан с установкой/выключением звукового бита порта AAh.

```
             Z80-Assembler   Page:    1
0090 =      GICINI  EQU  090h     ; инициализация
0135 =      CHGSND  EQU  135h     ; вкл/выкл. бита 7
009F =      CHGET   EQU  9Fh      ; ввод символа
00B7 =      BREAKX  EQU  0B7h     ; нажато ли CTRL/STOP ?
                    ORG  0A000h
A000 CD9000         CALL GICINI   ; инициализация
A003 3E01   Sound:  LD   A,1
```

```
A005 CD3501          CALL CHGSND   ; включаем бит
A008 CD9F00          CALL CHGET    ; ждем нажатия клавиши
A00B AF      Silen:  XOR  A
A00C CD3501          CALL CHGSND   ; выключаем бит
A00F CD9F00          CALL CHGET    ; ждем нажатия клавиши
A012 CDB700          CALL BREAKX   ; нажаты ли CTRL/STOP ?
A015 D8              RET  C        ; если да - выход
A016 18EB            JR   Sound
                     END
```

В последнем примере покажем использование подпрограмм BIOS с адресами C0h (beep) и 93h (запись данных в регистр PSG) для генерации шума моря.

```
                Z80-Assembler Page: 1
                     ORG     9000h
               ; шум моря
9000 CDC000            CALL    0C0h        ; beep
9003 211F90            LD      HL,ENDDATA ; адрес байта данных
                                          ; для 13 регистра PSG
9006 3E0D              LD      a,13        ; кол-во регистр. PSG
9008 5E                LD      e,(HL)      ; загрузить данные
9009 CD9300            CALL    93h         ; записать в регистр
900C 2B                DEC     HL          ; след. байт данных
900D D601              SUB     1           ; след. н-р рег. PSG
900F 30F7              JR      NC,$-7      ; если не -1, повтор.
9011 C9                RET                 ; возврат
               ;-----------------------------------
               ; данные для "шума моря"
9012                   DEFS    6
9018 1EB71000          DEFB    30,183,16,0,0,0,90,14
901C 00005A0E
               ;-----------------------------------
901F =         ENDDATA EQU     $-1
                       END
```

В качестве примера напишем программу установки режима GRAPHIC-2 видеопроцессора, создания спрайта размером 8*8 точек без увеличения и установки его на экране с координатами (128,100) цветом 13. При этом будем использовать подпрограммы BIOS.

```
           Z80-Assembler   Page:    1
                   ORG   9000h
           ; screen 2,2
9000 21E0F3        LD    HL,0F3E0h; адрес хранения рег. #1 VDP
9003 CB8E          RES   1,(HL)   ; спрайт 8*8
9005 CB86          RES   0,(HL)   ; нормальный размер спрайта
9007 CD7200        CALL  72h      ; screen 2
            ; создание шаблона спрайта
900A 3E00          LD    a,0      ; номер образа спрайта =0
900C CD8400        CALL  84h      ; узнаем адрес образа
900F 112990        LD    DE,dat   ; адрес данных для
                                  ; создания шаблона
9012 EB            EX    DE,HL    ; меняем HL и DE
9013 010800        LD    BC,8     ; длина образа
9016 CD5C00        CALL  5Ch      ; заполняем образ спрайта
                                  ; во VRAM
            ; выведение спрайта на экран
9019 3E00          LD    a,0      ; номер спрайта = 0
901B CD8700        CALL  87h      ; адрес таблицы атрибутов
901E 113190        LD    DE,pts   ; адрес данных для табл.
                                  ; атрибутов
9021 EB            EX    DE,HL
9022 010400        LD    BC,4     ; длина таблицы атрибутов
9025 CD5C00        CALL  5Ch
9028 C9            RET
            ; данные для шаблона спрайта
9029 01020408 dat: DEFB  1,2,4,8,16,32,64,128
902D 10204080
            ; атрибуты спрайта
9031 80       pts: DEFB  128      ; координата X
9032 64            DEFB  100      ; координата Y
9033 00            DEFB  0        ; номер шаблона
9034 0D            DEFB  13       ; цвет
                   END
```

При работе со спрайтами размером 16x16 точек учтите, что номер шаблона определяется так же, как и для спрайтов размером 8x8, но умноженный на 4.

В этой программе использована подпрограмма BIOS с адресом вызова 5Ch. Она переписывает блок данных из RAM во VRAM. Ниже мы попытаемся реализовать эту подпрограмму с помощью команд, работающих с портами ввода/вывода.

п.4. Магнитофон

Для работы с магнитофоном используются следующие подпрограммы BIOS: включение мотора и открытие файла (TAPION, E1h и TAPOON, EAh), чтение одного байта (TAPIN, E4h), запись одного байта (TAPOUT, EDh), конец работы с лентой (TAPIOF, E7h и TAPOOF, F0h) и другие.

Приведенная ниже программа "щелкает" реле включения/выключения мотора накопителя на магнитной ленте (т.е. просто включает и выключает его).

```
                Z80-Assembler   Page:    1
                  ORG     9000h
9000 AF      motor: XOR     a       ; очищаем аккумулятор
9001 CDF300         CALL    00F3h   ; вкл/выкл
9004 EE01           XOR     1       ; смена 0 на 1 или 1 на 0
9006 08             EX      AF,AF'  ; сменить аккумулятор
9007 010008         LD      BC,800h ; небольшая задержка
900A 0B      nt:    DEC     BC
900B 78             LD      a,b
900C B1             OR      c
900D 20FB           JR      nz,nt
900F CDB700         CALL    00B7h   ; проверить не нажато ли
                                    ; CTRL/STOP ?
9012 D8             RET     C       ; возврат, если нажато
9013 08             EX      AF,AF'  ; вернуть "наш" аккумулятор
9014 18EB           JR      motor+1 ; повторить действия
                    END
```

В этой подпрограмме используются две подпрограммы BIOS. Это 00F3h - включение и выключение мотора магнитофона и 00B7h - проверка, нажаты ли клавиши CTRL+STOP.

Во втором примере при помощи подпрограмм BIOS просматривается и печатается список файлов на магнитной ленте.

```
                  Z80-Assembler   Page:    1
00A2 =   Chput   EQU  00A2h      ; вывод символа
00E1 =   Tapion  EQU  00E1h      ; вкл. мотор, читать заголовок
00E4 =   Tapin   EQU  00E4h      ; читаем байт с ленты
00E7 =   Tapiof  EQU  00E7h      ; заверш. работу с магнитофоном
                 ORG  9000h
  ; === Просмотр имен файлов на ленте
9000 CDE100 Start: CALL Tapion   ; вкл. мотор, читать заголовок
9003 0610        LD   b,16
9005 21A090      LD   HL,Work    ; адрес рабочей области
9008 E5  Next:   PUSH HL         ; сохранить
9009 C5          PUSH BC
900A CDE400      CALL Tapin      ; читаем байт с ленты
900D C1          POP  BC
900E E1          POP  HL
900F 382B        JR   c,Error    ; если была ошибка, переход
9011 77          LD   (HL),a     ; иначе повторяем
9012 23          INC  HL
9013 10F3        DJNZ Next
9015 215790      LD   HL,Filnam  ; выводим имя файла
9018 CD4D90      CALL Putstr
901B 21AA90      LD   HL,Work+10 ;
901E CD4D90      CALL Putstr
9021 CD4690      CALL Crlf
9024 3AA090      LD   a,(Work)   ; проверяем атрибуты файла
9027 217090      LD   HL,Binfil  ; файл двоичный ?
902A FED3        CP   0D3h
902C 2811        JR   z,Prt
902E 216390      LD   HL,Ascfil  ; файл ASCII ?
9031 FEEA        CP   0EAh
9033 280A        JR   z,Prt
9035 217E90      LD   HL,Macfil  ; файл кодов ?
9038 FED0        CP   0D0h
903A 2803        JR   z,Prt
903C 218B90 Error: LD HL,Errstr  ; сообщение об ошибке
903F CD4D90 Prt: CALL Putstr     ; вывод строки
9042 CDE700      CALL Tapiof     ; заверш. работу с магнитофоном
9045 C9          RET
  ; === Подпрограмма перевода строки
9046 219D90 Crlf:LD   HL,Stcrlf  ; перевод строки
9049 CD4D90      CALL Putstr
904C C9          RET
  ; === Подпрограмма вывода строки
904D 7E  Putstr: LD   a,(HL)
904E FE24        CP   '$'
9050 C8          RET  z
9051 CDA200      CALL Chput
9054 23          INC  HL
9055 18F6        JR   Putstr
  ; === Данные
9057 46696C65 Filnam: DB   'File name: $'
905B 206E616D
905F 653A2024
```

```
9063 41736369 Ascfil: DB   'Ascii file',0Dh,0Ah,'$'
9067 69206669
906B 6C650D0A
906F 24
9070 42696E61 Binfil: DB   'Binary file',0Dh,0Ah,'$'
9074 72792066
9078 696C650D
907C 0A24
907E 42736176 Macfil: DB   'Bsave file',0Dh,0Ah,'$'
9082 65206669
9086 6C650D0A
908A 24
908B 54617065 Errstr: DB   'Tape read error',0Dh,0Ah,'$'
908F 20726561
9093 64206572
9097 726F720D
909B 0A24
909D 0D0A24 Stcrlf: DB   0Dh,0Ah,'$'
90A0        Work:   DS 16,0
90B0 24             DB '$'
                    END
```

Когда при помощи этих подпрограмм BIOS создаются подпрограммы READ/WRITE для файлов на кассете, нужно использовать только READ или WRITE без каких-либо других действий. Например, чтение данных с ленты и отображение их на экране может вызвать ошибку чтения.

п.5. Часы и энергонезависимая память

В книге "Архитектура микрокомпьютера MSX-2" была описана структура и функции микросхемы CLOCK-IC с энергонезависимой памятью. Имеются две подпрограммы расширенного BIOS - REDCLK (01F5h) и WRTCLK (01F9h), которые позволяют записывать информацию в регистры CLOCK-IC или читать ее оттуда (см. MSX-BASIC BIOS).

Ниже приводится пример программы, которая устанавливает некоторые параметры экрана, восстанавливаемые при перезагрузке или включении компьютера - тип и ширину экрана, цвет изображения и фона.

```
                    Z80-Assembler   Page:    1
01F9 =    WRTCLK    EQU   01F9h
015F =    EXTROM    EQU   015Fh
                    ORG 9000h
  ; === Установка типа экрана
9000 0E23           LD    C,23H  ; блок 2, регистр 3
9002 3E00           LD    A,0    ; тип интерфейса и экрана
9004 CD2490         CALL  WrtRAM ; запись в CLOCK-IC
  ; === Установка ширины экрана 40 (28h)
9007 0E24           LD    C,24H  ; блок 2, регистр 4
9009 3E08           LD    A,8    ; младшие биты (28h)
900B CD2490         CALL  WrtRAM ; запись в CLOCK-IC
900E 0E25           LD    C,25H  ; блок 2, регистр 5
9010 3E02           LD    A,2    ; старшие биты (28h)
9012 CD2490         CALL  WrtRAM ; запись в CLOCK-IC
  ; === Установка цвета изображения
9015 0E26           LD    C,26H  ; блок 2, регистр 6
9017 3E04           LD    A,4    ; COLOR 4
9019 CD2490         CALL  WrtRAM ; запись в CLOCK-IC
  ; === Установка цвета фона
901C 0E27           LD    C,27H  ; блок 2, регистр 7
901E 3E0E           LD    A,14   ; COLOR ,14
9020 CD2490         CALL  WrtRAM ; запись в CLOCK-IC
   ; === Возврат в MSX-BASIC
9023 C9             RET
  ; === Подпрограмма записи в CLOCK-IC
9024 E5  WrtRAM:    PUSH  HL        ; сохраняем регистры
9025 C5             PUSH  BC
9026 DD21F901       LD    IX,WRTCLK ; межслотовый вызов
902A CD5F01         CALL  EXTROM    ; SUBROM EBIOS
902D E1             POP   HL        ; восстанавливаем
902E C1             POP   BC        ;    регистры
902F C9             RET             ; возврат
                    END
```

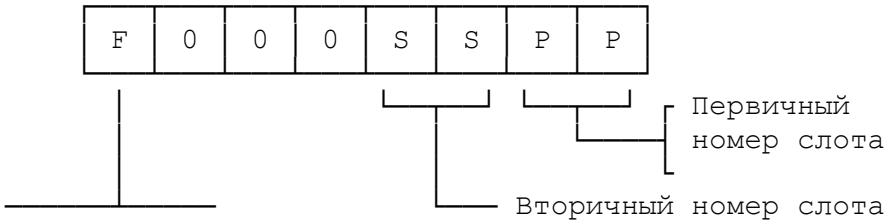
п.6. Межслотовые вызовы подпрограмм

При выполнении программы может возникнуть ситуация, когда необходимо вызвать подпрограмму, находящуюся в текущий момент в неактивном слоте.

Например, при режиме работы, когда на всех страницах включена только оперативная память, может понадобиться вызов подпрограммы BIOS, хранящейся в одном из слотов ПЗУ.

В этом случае можно либо попытаться включить необходимые слоты и вызвать подпрограмму, либо выполнить межслотовый вызов.

Межслотовый вызов выполняется подпрограммой BIOS CALSLT по адресу 1Ch. MSX-DOS также поддерживает эту подпрограмму. Перед вызовом в регистр IX нужно загрузить адрес требуемой подпрограммы BIOS, а в IY - указатель слота в виде:



При межслотовом вызове CALSLT и CALLF прерывания автоматически запрещаются. При возврате управления из этих подпрограмм в MSX-1 прерывания остаются запрещенными, а в MSX-2 восстанавливается статус прерываний, который был установлен до вызова подпрограмм.

```
Например,   LD   IX,0156h  ; адрес чистки буфера клавиатуры
            LD   IY,0      ; указатель слота
            CALL 1Ch       ; межслотовый вызов
            EI             ; включение прерываний в MSX-1
```

Понятно, что в этом случае регистры IX и IY не могут использоваться для передачи параметров.

Для межслотового вызова можно использовать и команду рестарта.

```
Например,   RST  30h
            DB   0         ; указатель слота
            DW   6Ch       ; установка SCREEN 0
```

В некоторых случаях межслотовый вызов оказывается невозможным. Например, если вызываемая подпрограмма работает не в одной, а в двух страницах памяти неактивного слота (межслотовый вызов делает временно активной только одну страницу).

В такой ситуации можно создать свой собственный межслотовый вызов - перейти в страницу, активную в любом случае, запомнить адрес возврата, активировать нужные слоты и страницы, вызвать нужную подпрограмму, восстановить конфигурацию слотов и вернуться.

При вызове подпрограмм расширенного BIOS - SUBROM EBIOS нужно учитывать, что они находятся в различных вторичных слотах памяти компьютера ученика (3-0) и компьютера учителя (3-1). Указатель

слота EBIOS записан в ячейке EXBRASA (FAF8h). Поэтому для вызова подпрограммы EBIOS можно использовать команды вида:

```
        LD   IX,имя
        LD   IY,(EXBRASA-1)
        CALL 1Ch
```

Другая возможность вызова SUBROM - использование подпрограммы BIOS EXTROM (015Fh). Адрес вызова записывается в регистр IX, задание IY уже не требуется.

Ниже приводится пример программы, рисующей в SCREEN 5 букву A и линию при помощи межслотового вызова EBIOS.

```
     MSX.M-80 1.00 01-Apr-85 PAGE 1
                    .Z80
001C        CALSLT  EQU  1Ch        ; межслотовый вызов
0030        CALLF   EQU  30h        ; межслотовый вызов
006C        IniTxt  EQU  006Ch      ; инициирование SCREEN 0
009F        GetChr  EQU  009Fh      ; ввод символа
0156        KillBuf EQU  0156h      ; чистка буфера клавиатуры
; --- Расширенный BIOS
0089        GRPPTR  EQU  89h        ; вывод символа в SCREEN 5
00D1        CHGMOD  EQU  0D1h       ; установка типа экрана
0085        DOGRPH  EQU  85h        ; рисует линию
; --- Системная область
F3E9        FORCLR  EQU  0F3E9h     ; цвет текста
F3F2        ATRBYT  EQU  0F3F2h     ; цвет байта
FAF8        EXBRASA EQU  0FAF8h     ; указатель слота EBIOS
FB02        LOGOPR  EQU  0FB02h     ; логическая операция
FCB3        GXPOS   EQU  0FCB3h     ; позиция X графического курсора
FCB5        GYPOS   EQU  0FCB5h     ; позиция Y графического курсора
; --- установить и инициализировать SCREEN 5
0000' 3E 05    Start:  LD   A,5
0002' DD 21 00D1       LD   ix,CHGMOD
0006' FD 2A FAF7       LD   iy,(EXBRASA-1)
000A' CD 001C          CALL CALSLT    ; MSX-DOS поддерживает
                                      ; межслотовый вызов
; --- вывести символ на экран
000D' AF               XOR  a              ; код логич. операции
000E' 32 FB02          LD   (LOGOPR),a
0011' 3E 0D            LD   a,13           ; фиолетовый цвет
0013' 32 F3E9          LD   (FORCLR),a
0016' 3E 41            LD   a,'A'          ; буква A
0018' DD 21 0089       LD   ix,GRPPTR
001C' FD 2A FAF7       LD   iy,(EXBRASA-1)
0020' CD 001C          CALL CalSlt      ; вывод буквы A
; --- нарисовать линию
0023' 3E 96            LD   A,150          ; куда рисовать
0025' 32 FCB3          LD   (GXPOS),A
0028' 3E 62            LD   A,98
002A' 32 FCB5          LD   (GYPOS),A
002D' 01 0014          LD   BC,20          ; откуда
0030' 11 000A          LD   DE,10
0033' 3E 0A            LD   A,10           ; цвет линии
```

```
0035' 32 F3F2          LD   (ATRBYT),A
0038' AF               XOR  a              ; код логич. операции
0039' 32 FB02          LD   (LOGOPR),A
003C' DD 21 0085       LD   ix,DOGRPH
0040' FD 2A FAF7       LD   iy,(EXBRASA-1)
0044' CD 001C          CALL CALSLT         ; рисуем линию
;--- Выход в SCREEN 0
0047' F7               RST  CALLF
0048' 00               DB   0
0049' 0156             DW   KillBuf        ; чистка буфера
004B' F7               RST  CALLF
004C' 00               DB   0
004D' 009F             DW   GetChr         ; ждем символ
004F' F7               RST  CALLF
0050' 00               DB   0
0051' 006C             DW   IniTxt         ; SCREEN 0
0053' C7               RST  0              ; перезагрузка
                       END  Start
```

п.7. Вывод на печать

Для вывода на печать используются следующие подпрограммы BIOS: проверка статуса принтера (LPTSTT, 0A8h) и вывод символа на принтер (LPOUT, 0A5h и OUTDLP, 14Dh). Подпрограмма OUTDLP в отличие от LPOUT сообщает о ненормальном завершении операции вывода.

В качестве примера приведем программу вывода на печать графического изображения при помощи ESC-последовательности ESC+"Snnnn" (см. описание системы команд принтера).

```
                   Z80-Assembler   Page:    1
00A8 =    LPTSTT   EQU   0A8h
014D =    OUTDLP   EQU   14Dh
                   ORG   9000h
  ; === Проверка статуса принтера
9000 CDA800        CALL  LPTSTT
9003 283A          JR    Z,NotReady  ; если не может
9005 B7            OR    A
9006 2837          JR    Z,NotReady  ; если не может
  ; === Вывод на принтер строки
9008 3E1B          LD    A,27        ; ESC
900A CD4D01        CALL  OUTDLP
900D 3E42          LD    A,'B'       ; B
900F CD4D01        CALL  OUTDLP
9012 0614          LD    B,20        ; все повторить 20 раз
9014 C5  NxtEle:   PUSH  BC
9015 0614          LD    B,20        ; кол-во элементов одного символа
9017 212B90        LD    HL,Data     ; адрес данных для элемента
  ; === Вывод одного элемента
901A E5  NxtCol:   PUSH  HL          ; сохраняем
901B C5            PUSH  BC
```

```
901C 7E           LD    A,(HL)   ; выводим один элемент
901D CD4D01       CALL  OUTDLP   ;    изображения
9020 381E         JR    C,Error  ; переход при ошибке
9022 C1           POP   BC       ; восстанавливаем параметры
9023 E1           POP   HL
9024 23           INC   HL
9025 10F3         DJNZ  NxtCol   ; переходим к след. элементу
9027 C1           POP   BC
9028 10EA         DJNZ  NxtEle   ; переходим к след. символу
902A C9           RET
902B 1B533030 Data:DB 27,'S0014',0FFh,0FEh,0FCh,0F8h,0F0h,0E0h
902F 3134FFFE
9033 FCF8F0
9036 E0C080C0      DB   0C0h,080h,0C0h,0E0h,0F0h,0F8h,0FCh,0FEh
903A E0F0F8FCFE
       NotReady:
    ; ...   ...   подпрограмма обработки неготовности принтера
903F C9           RET
       Error:
    ; ...   ...   подпрограмма обработки ошибки принтера
9040 C9           RET
                  END
```

14. Ловушки

Работа компьютера MSX-2 практически состоит в выполнении набора определенных стандартных подпрограмм, вызываемых явно или неявно пользователем, операционной системой или прикладной программой. К этим подпрограммам, например, относятся программы ввода и запоминания кода символа, нажатого на клавиатуре, выдачи списка файлов и листинга программы, вывода символа на экран и т.п.

При желании программист может дополнить или изменить любую из этих программ. Для этого разработчиками компьютера был предусмотрен механизм ловушек. Идея состоит в том, что перед тем как начать выполнение, многие стандартные программы осуществляют вызов подпрограммы-ловушки.

В системной области для каждой ловушки отводится 5 байт. Список ловушек приводится в книге "Архитектура микрокомпьютера MSX-2". В обычном состоянии в ловушке записан код команды возврата из подпрограммы (RET) - C9h. Таким образом, при вызове ловушки управление обычно тут же возвращается назад, и работает стандартная программа.

В ловушке может находиться и команда перехода (RST) на подпрограммы MSX-Disk-BASIC, локальной сети и других системных программ. Так MSX-Disk-BASIC обрабатывает, например, команды работы с файлами.

Если программист хочет изменить нормальный ход работы, он может в ловушку записать свои команды. Поскольку в 5 байт много не запишешь, обычно в ловушку записывают команду перехода на подпрограмму (JP или RST). Если после выполнения такой подпрограммы-ловушки был обычный возврат управления (RET), то начнется работа стандартной подпрограммы.

Приведем несколько примеров.

Первый пример - обязательная чистка экрана перед выполнением команды LIST языка MSX-BASIC. Ловушка для LIST и LLIST находится по адресу FF89h. Мы можем заполнить ее следующими кодами:

```
Адрес     Код      Команда ассембл.
FF89      F7       RST 30h          ; межслотовый вызов
FF8A      00       DB  0            ; чистка экрана
FF8B      C3 00    DW  0C3h
FF8D      C9       RET              ; возврат из ловушки
```

На языке MSX-BASIC заполнить ловушку можно при помощи команды POKE. После этого перед выполнением команды LIST ловушкой будет выполняться чистка экрана.

п.1. Работа с файлами

Рассмотрим установку ловушки для команды FILES, которая должна очистить экран и вывести список файлов. Особенность здесь заключается в том, что ловушку на эту команду устанавливает и MSX-Disk-BASIC. Поэтому вначале будет работать наша ловушка, а затем нужно обеспечить выполнение ловушки MSX-Disk-BASICa.

```
                  Z80-Assembler   Page:    1
                   ORG  0A000h
FE7B =          FileTrap EQU  0FE7Bh       ; ловушка для FILES
00C6 =          Posit    EQU  0C6h         ; установка позиции
00A2 =          ChPut    EQU  0A2h         ; выдача символа
; ===  установка ловушки "JP Trap"
A000 217BFE         LD   HL,FileTrap  ; сохраняем ловушку
A003 1137A0         LD   DE,ForHook   ; MSX-Disk-BASIC
A006 010500         LD   BC,5
A009 E5             PUSH HL
A00A EDB0           LDIR
A00C E1                  POP HL            ; устанавливаем свою
A00D 36C3           LD   (HL),0C3h    ;     "JP Trap"
A00F 23             INC  HL
A010 3616           LD   (HL),Low(Trap)
A012 23             INC  HL
A013 36A0           LD   (HL),High(Trap)
A015 C9             RET                ; ловушка установлена
; === ловушка для FILES
A016 F5       Trap: PUSH AF            ; сохраняем все регистры
A017 C5             PUSH BC
A018 D5             PUSH DE
A019 E5                  PUSH HL
A01A 21010A              LD   HL,0A01h     ; позиция (10,1)
A01D CDC600              CALL Posit
A020 213CA0              LD   HL,Messg     ; печатаем заголовок
A023 7E         NextCh:  LD   A,(HL)
A024 B7                  OR   A
A025 2806                JR   Z,Exit
A027 CDA200              CALL ChPut
A02A 23                  INC  HL
A02B 18F6                JR   NextCh
A02D 210202      Exit:   LD   HL,0202h     ; позиция (2,2)
A030 CDC600              CALL Posit
A033 E1                  POP  HL           ; восстанавливаем
A034 D1                  POP  DE           ; регистры
A035 C1                  POP  BC
A036 F1                  POP  AF
; === Выполняем ловушку FILES MSX-Disk-BASIC, возврат
A037          ForHook:   DEFS 5
A03C 0CF3D0C9 Messg:     DB   12,'Список файлов:',0
A040 D3CFCB20
A044 C6C1CACC
A048 CFD73A00
                         END
```

Загрузите и выполните эту программу. Если Вы затем наберете команду FILES, будет очищен экран, выведена надпись "Список файлов:" и сам список файлов.

п.2. Работа с клавиатурой

В ходе работы программ клавиатура постоянно опрашивается (сканируется) системой. При нажатии какой-нибудь клавиши информация об этом временно записывается в таблицу (матрицу) сканирования клавиатуры NEWKEY (FBE5h). Каждому байту матрицы соответствует 8 клавиш. Если некоторая клавиша была нажата, соответствующий бит байта обнуляется. При этом в регистр A записывается значение 8*NS+NC, где NS - номер строки, NC - номер колонки.

Матрица NEWKEY обрабатывается системой, и в зависимости от того, какие клавиши были нажаты, либо предпринимаются какие-то действия (например, CTRL/STOP), либо в буфер клавиатуры BUF (F55Eh) записывается код символа (например, Shift/$ дает 0).

Используя ловушку KEYCOD (FDCCh), можно отменить действие некоторой клавиши или провести ее дополнительную обработку.

Приведем пример создания ловушки, которая при нажатии клавиши STOP записывает в матрицу клавиатуры и код клавиши CTRL, а при нажатии любой другой клавиши имитирует одновременное нажатие клавиши Shift.

```
┌──────────────────────────
    MSX.M-80 1.00 01-Apr-85  PAGE 1
                          .Z80
9000              Load    EQU  9000h   ; загрузочный адрес
FBE5              NewKey  EQU  0FBE5h  ; матрица клавиатуры
FDCC              KeyCod  EQU  0FDCCh  ; ловушка для клавиатуры
F931              Work    EQU  0F931h  ; область для ловушки
009F              ChGet   EQU  09Fh    ; ввод символа
00A2              ChPut   EQU  0A2h    ; вывод символа
0007              S.Stop  EQU  7       ; позиции STOP в NEWKEY
0004              P.Stop  EQU  4
0006              S.CTRL  EQU  6       ; позиции CTRL в NEWKEY
0001              P.CTRL  EQU  1
0006              S.Shift EQU  6       ; позиции Shift в NEWKEY
0000              P.Shift EQU  0
; ===  Header Obj-файла
0000'                     ASEG
0000    FE                DB   0FEh     ; obj-файл
0001    9000              DW   Load     ; адрес загрузки
0003    9035              DW   EndLoad  ; конечный адрес
0005    9000              DW   Start    ; стартовый адрес
; ===  Переписываем ловушку в рабочую область
                          .PHASE Load
9000    F3        Start:  DI
9001    21 9023           LD   HL,TrapModule
9004    11 F931           LD   DE,Work
9007    01 0013           LD   BC,EndTrap-Trap
900A    ED B0             LDIR
; ===  Устанавливаем ловушку "JP Trap"
900C    3E C3             LD   A,0C3h      ; код JP
```

```
900E    32 FDCC           LD   (KeyCod),A
9011    21 F931           LD   HL,Trap
9014    22 FDCD           LD   (KeyCod+1),HL
9017    FB                EI
; === Ввод и вывод символов до нажатия клавиши RETURN
9018    CD 009F  Next:    CALL ChGet
901B    FE 0D             CP   13
901D    C8                RET  Z        ; выход !
901E    CD 00A2           CALL ChPut
9021    18 F5             JR   Next
9023         TrapModule   EQU  $
                          .DEPHASE
; === Ловушка для клавиатуры
                          .PHASE Work
F931    F5       Trap:    PUSH AF
F932    FE 3C             CP   8*S.Stop+P.Stop   ; STOP ?
F934    20 07             JR   NZ,Shift
; === "Нажимаем" CTRL и выходим
F936    3E FD             LD   A,not(1 SHL P.CTRL)
F938    32 FBEB           LD   (NewKey+S.CTRL),A
F93B    F1                POP  AF
F93C    C9                RET
; === "Нажимаем" Shift и выходим
F93D    3E FE    Shift:   LD   A,not(1 SHL P.Shift)
F93F    32 FBEB           LD   (NewKey+S.Shift),A
F942    F1                POP  AF
F943    C9                RET
F944           EndTrap    EQU  $
                          .DEPHASE
                          .PHASE Load+$-7
9035            EndLoad   EQU   $-1
                          .DEPHASE
                          END
```

Оттранслировав эту программу ассемблером M80, получим объектную программу с расширением "COM". Переименуйте ее в "KEY.OBJ" и выполните следующую программу на языке MSX-BASIC:

```
10  ON STOP GOSUB 100: STOP ON
20  BLOAD "KEY.OBJ",R
30  GOTO 30
100 PRINT "CTRL/STOP": RETURN
```

Программа будет выводить символы, как если бы была нажата клавиша Shift. Выход из программы в кодах - по нажатию клавиши RETURN. Нажатие клавиши STOP будет обрабатываться как нажатие двух клавиш - STOP и CTRL.

15. Подпрограммы интерпретатора языка MSX-BASIC

Программирующий на языке MSX-BASIC может использовать все операции и функции, имеющиеся в этом языке. Многие из них могли бы быть полезны и при программировании на языке ассемблера Z80. Для обращения к операциям и функциям интерпретатора языка MSX-BASIC нужно знать их входные точки и правила использования.

Необходимо также учитывать, что программы на ассемблере, использующие эти возможности, становятся немобильными из-за того, что входные точки не стандартизированы. Поэтому применять операции и функции MSX-BASICa нужно по мере достаточной необходимости.

Список входных точек и правила вызова операций и функций MSX-BASICa приведены в книге "Архитектура микрокомпьютера MSX-2".

Не забывайте, что вызов подпрограмм MSX-BASICa возможен, например, либо если Вы работаете в обычном режиме MSX-BASICa (слот 0 активен), либо если в режиме MSX-DOS выполняется межслотовый вызов MSX-BASIC BIOS.

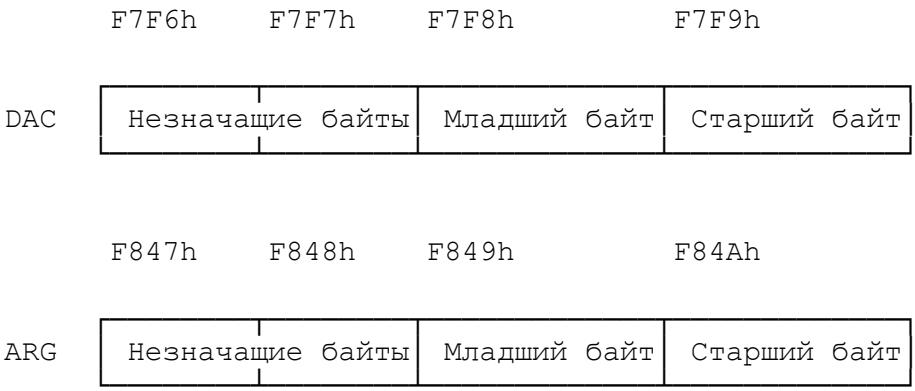
При возникновении ошибки при выполнении подпрограмм происходит обращение к программе обработки ошибки MSX-BASICa. Для применения своей реакции на ошибку следует использовать хук H.ERRO (FFB1h).

В системной области имеются два регистра - аккумуляторы, при помощи которых интерпретатор языка MSX-BASIC выполняет арифметические и некоторые другие операции:

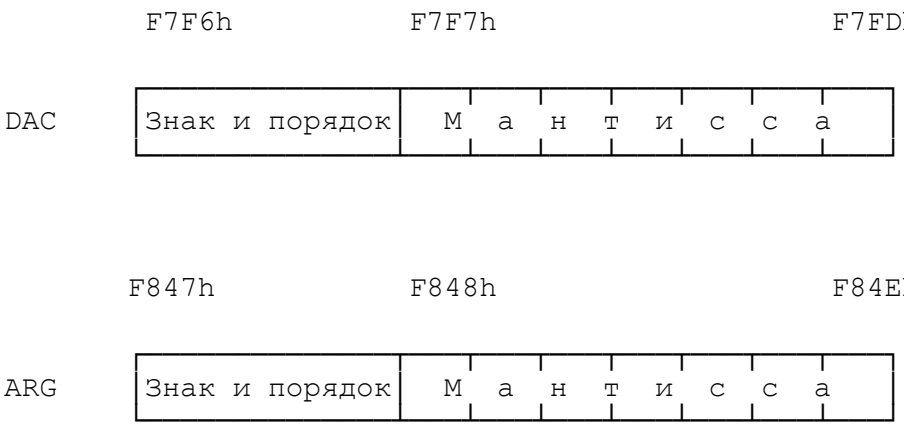
DAC ("Decimal ACcumulator" - "десятичный аккумулятор") - по адресу F7F6h, 16 байт;

ARG ("ARGument" - "аргумент") - по адресу F847h, 16 байт.

Целое число в аккумуляторах размещается следующим образом:



Вещественное число в аккумуляторах размещается следующим образом:



Напомним, что мантисса числа обычной точности состоит из 6 десятичных цифр (3 байта), а двойной точности - из 14 цифр (7 байт).

Подробнее о представлении и хранении чисел было рассказано в книге "Архитектура микрокомпьютера MSX-2".

В ячейке VALTYP (по адресу F663h) системной области хранится тип числа, находящегося в аккумуляторе DAC. Тип закодирован следующим образом:

2 - целое число;
4 - вещественное число одинарной точности;
8 - вещественное число двойной точности;
3 - строка.

п.1. Работа с целыми числами

Для работы с целыми числами имеются подпрограммы сложения, вычитания, умножения, деления, вычисления остатка от деления, возведения в степень.

Например, подпрограмма UMULT по адресу &h314A записывает произведение содержимого BC и DE в DE, а подпрограмма INTEXP по адресу &h383F записывает в DAC степень HL содержимого DE.

Необходимо учитывать, что при работе подпрограммы, как правило, изменяют все регистры. Рассмотрим примеры.

```
                    Z80-Assembler   Page:    1
                     ORG  9000h
314A =         UMULT EQU  314Ah
2F99 =         retHL EQU  2F99h
        ; Умножение целых чисел
        ;    DE := BC * DE
        ; изменяются A,B,C,D,E
9000 ED4B1090        LD   BC,(X)     ; загрузка
9004 ED5B1290        LD   DE,(Y)     ; операндов
9008 CD4A31          CALL UMULT      ; умножаем
900B 62              LD   H,D        ; копируем в HL
900C 6B              LD   L,E        ; для возврата
900D C3992F          JP   retHL      ; с помощью USR
9010 7D00      X:    DEFW 125
9012 2001      Y:    DEFW 288
                     END
```

Возведение целых чисел в степень.

```
                    Z80-Assembler   Page:    1
                       ORG   9000h
F7F6 =         DAC     EQU   0F7F6h
383F =         INTEXP  EQU   383Fh
        ;    DAC := DE ^ HL
9000 111400            LD    DE,0014h   ; загрузка
9003 210300            LD    HL,0003h   ;    операндов
9006 CD3F38            CALL  INTEXP     ; степень
9009 3AF8F7            LD    A,(DAC+2)  ; копируем в память
900C 3200A0            LD    (0a000h),A
900F 3AF9F7            LD    A,(DAC+3)  ; копируем в память
9012 3201A0            LD    (0a001h),A
9015 C9                RET
                       END
```

п.2. Работа с вещественными числами

При работе с веществеными числами можно использовать различные виды пересылок между DAC, ARG и памятью. Например, подпрограмма MFM по адресу &h2C5C переписывает значение двойной точности из памяти, адресуемой HL в DAC, а MOVMF по адресу &h2EE8 переписывает значение обычной точности из DAC в память, адресуемую HL.

Большое количество подпрограмм позволяет выполнять операции вычитания, сложения, нормализации, округления, умножения, возведения в степень и вычислять значения функций косинус, синус, логарифм, псевдослучайное число и других.

Например, программа вычисления косинуса может выглядеть так:

```
                  Z80-Assembler   Page:    1
                     ORG   9000h
F663 =        VALTYP  EQU   0F663h
2EBE =        MOVFM   EQU   2EBEh
2993 =        COS     EQU   2993h
2EE8 =        MOVMF   EQU   2EE8h
         ; вычисление косинуса
9000 211590           LD    HL,data         ; загрузка адреса
9003 3E04             LD    A,4             ; тип - вещественный
9005 3263F6           LD    (VALTYP),A      ;
9008 CDBE2E           CALL  MOVFM           ; данные - в DAC
900B CD9329           CALL  COS             ; COS( DAC) => DAC
900E 211990           LD    HL,result       ; загрузка адреса
9011 CDE82E           CALL  MOVMF           ; DAC - в память
9014 C9               RET
9015 41157080 data:   DB    41h,15h,70h,80h; число: 1.5708
9019          result: DS    4,0
                      END
```

Кроме этого имеется набор подпрограмм сравнений и преобразований значений различных типов. Наиболее полезными из них являются подпрограммы преобразования чисел в текстовый вид для последующего вывода.

Для такого преобразования предназначены подпрограммы FOUT (3425h) - неформатный вывод и PUFOUT (3426h) - форматный вывод; Эти подпрограммы обрабатывают число, находящееся в DAC. Адрес полученной строки символов (уменьшенный на 1) записывается в HL.

В регистре A записывается формат преобразования. Содержимое его битов определяет следующее:

```
bit 7: если 1, то вывод осуществляется по формату;
bit 6: если 1, то через каждые 3 цифры вставляются запятые;
bit 5: если 1, то первые нули нужно заменить на символ "*";
bit 4: если 1, то перед числом вставить символ "$";
bit 3: если 1, то число выводится всегда со знаком;
bit 2: если 1, то вставить знак после числа;
bit 1:     не используется;
bit 0: если 0, то число выводится с фиксированной точкой;
       если 1, то число выводится с плавающей точкой;
```

В регистре B должно быть количество цифр перед точкой +2.

В регистре C - количество цифр после точки +1.
Приведем пример программы.

```
                        Z80-Assembler   Page:    1
3426 =        FOUT    EQU   3426h
F663 =        VALTYP  EQU   0F663h
F7F6 =        DAC     EQU   0F7F6h
2993 =        Cos     EQU   2993h
00A2 =        ChPut   EQU   0A2h
                 ORG   8100h
        ; запись числа в DAC
8100 212781           LD    HL,data
8103 11F6F7           LD    DE,DAC
8106 010400           LD    BC,4
8109 EDB0             LDIR                ; переписали в DAC
810B 3E04             LD    A,4           ; вещественное число
810D 3263F6           LD    (VALTYP),A    ; установили тип
        ; вычисляем косинус
8110 CD9329           CALL  Cos
        ; преобразование числа (DAC) в строку
        ; по формату
8113 3E88             LD    A,10001000b   ; формат
8115 0603             LD    B,3           ; до точки
8117 0E05             LD    C,5           ; после точки
8119 CD2634           CALL  FOUT
        ; адрес строки - в (HL)+1
        ; выводим ее на экран
811C 23       Next:   INC   HL
811D 7E               LD    A,(HL)
811E B7               OR    A             ; код символа ?
811F 2805             JR    Z,Exit        ; если ноль - все !
8121 CDA200           CALL  ChPut         ; иначе - вывод
8124 18F6             JR    Next
8126 C9       Exit:   RET
8127 43123456 data:   DB    43h,12h,34h,56h
                END
```

Для числа-аргумента этой программы 123.456 будет вычислено и напечатано в качестве результата значение -0.5944.

16. Подпрограммы BDOS

При выполнении программ типа ".COM" ПЗУ интерпретатора языка MSX-BASIC обычно отключено. Однако операционная система MSX-DOS имеет свой набор стандартных функций (подпрограмм) BDOS BIOS, при помощи которых можно осуществлять операции ввода/вывода на экран, принтер, на диски, работать с клавиатурой и выполнять некоторые другие операции. Их общее количество - около пятидесяти.

Список системных функций BDOS приведен в книге "Архитектура микрокомпьютера MSX-2". Каждая функция имеет свой код.

Для вызова функции BDOS необходимо:

- загрузить код функции в регистр C;
- загрузить параметры в соответствующие регистры;
- вызвать подпрограмму по адресу 5.

Вызов функций BDOS поддерживается и интерпретатором Disk BASICa на машинах, где он установлен. При этом код функции тоже загружается в регистр C, но вызывать нужно адрес &hF37D.

Например, функция с кодом 2 выводит на экран символ, код которого записан в регистр E. Функция с кодом 9 выводит на экран строку. Причем адрес строки должен быть записан в регистровую пару DE, а конец строки обозначается символом "$".

Ниже приведен листинг программы, вызывающей эти функции.

```
            MSX.M-80  1.00  01-Apr-85  PAGE 1
                            .Z80
0005                BDOS    EQU     5
0002                PUTSCR  EQU     2
0009                PUTSTR  EQU     9
0000' 0E 02                 LD      C,PUTSCR
0002' 1E 41                 LD      E,65
0004' CD 0005               CALL    BDOS
0007' 0E 02                 LD      C,PUTSCR
0009' 1E 42                 LD      E,66
000B' CD 0005               CALL    BDOS
000E' 0E 09                 LD      C,PUTSTR
0010' 11 0017'              LD      DE,string
0013' CD 0005               CALL    BDOS
0016' C9                    RET
0017' 68 65 6C 6C   string: DB      "Hello, fellows!$"
001B' 6F 2C 20 66
001F' 65 6C 6C 6F
0023' 77 73 20 21 24
                            END
```

17. Сетевые функции

Локальная сеть КУВТ-2 имеет систему стандартных сетевых функций ввода/вывода (Net ROM BIOS), включающую в себя функции инициализации сети, проверки номера компьютера, передачи/приема программ и данных, чтения/записи из памяти своего компьютера или компьютера ученика и другие.

Сетевые функции могут быть вызваны как при работе в режиме MSX-BASIC, так и при работе в MSX-DOS. Список сетевых функций дан в книге "Архитектура микрокомпьютера MSX-2".

Для проверки, имеет ли система сетевое ПЗУ, можно посмотреть, хранится ли по вызываемому адресу "заглушка" RST30 (F7h) или записан ли идентификатор "RNT" в сетевом ПЗУ по адресам 4040h-4042h.

Для инициализации сети в стандартной MSX-DOS необходимо вызвать подпрограмму по адресу F98Eh.

Для вызова сетевой функции нужно поместить код функции (01h-1Ah) в регистр C, начальный адрес блока параметров в регистровую пару DE и вызвать подпрограмму по адресу F989h.

Для окончания работы с сетью вызывается подпрограмма по адресу F984h.

Посмотрите пример программы, работающей с локальной сетью в стандартной MSX-DOS.

```
MSX.M-80       1.00    01-Apr-85           PAGE 1
                         .Z80
  F98E             NETINIT EQU    0F98Eh
  F989             NETFUNC EQU    0F989h
  F984             NETEND  EQU    0F984h
  0005             BDOS    EQU    5
  0009             PUTSTR  EQU    9
                   ; === Netinit & Who ?
  0000'   CD F98E           CALL   NETINIT
  0003'   0E 06             LD     C,6       ; Who ?
  0005'   CD F989           CALL   NETFUNC
                   ; === Check computer number
  0008'   32 0048'          LD     (WHO),A
  000B'   B7                OR     A
                   ; jump, if not a teacher
  000C'   20 08             JR     NZ,putnum
                   ; === Send message to Pupils
  000E'   0E 0D             LD     C,0Dh
  0010'   11 0039'          LD     DE,message
  0013'   CD F989           CALL   NETFUNC
  0016'   CD F984  putnum:  CALL   NETEND
  0019'   0E 09             LD     C,PUTSTR
  001B'   11 003E'          LD     DE,number
  001E'   3A 0048'          LD     A,(WHO)
  0021'   C6 30             ADD    A,'0'
  0023'   32 0048'          LD     (WHO),A
  0026'   CD 0005           CALL   BDOS
  0029'   C9                RET
  002A'   48 65 6C  text:   DEFM   'Hello, fellows!'
  002E'   6C 6F 2C 20 66
```

```
0032'   65 6C 6C 6F
0036'   77 73 21
0039'   00      message: DB    0       ; всем
003A'   002A'            DEFW  text    ; адрес
003C'   000F             DEFW  15      ; длина
003E'   4E 75 6D number: DEFM  'Number is '
0042'   62 65 72 20 69
0046'   73 20
0048'   00       WHO:    DB    0
0049'   24               DB    '$'
                         END
```

Если Вы используете нестандартную операционную систему, то она может иметь другие точки входа в NET BIOS или вообще их не иметь. В этом случае можно обратиться непосредственно ко входным точкам функций локальной сети. Они находятся по адресам:

NETINIT - 33/401Ch,
NETFUNC - 33/4019h,
NETEND - 33/4016h.

Программа, приведенная выше, может быть с учетом этого переписана следующим образом:

```
       MSX.M-80  1.00  01-Apr-85          PAGE 1
                        .Z80
0005             BDOS   EQU     5
0009             PUTSTR EQU     9
0000'   CD 002A'         CALL    NETINIT
0003'   0E 06            LD      C,6       ; Who ?
0005'   CD 002F'         CALL    NETFUNC
                 ; === Check computer number
0008'   32 0057'         LD      (WHO),A
000B'   B7               OR      A
                 ; === Jump, if not a teacher
000C'   20 08            JR      NZ,putnum
                 ; ===  Send message to Students
000E'   0E 0D            LD      C,0Dh
0010'   11 0048'         LD      DE,message
0013'   CD 002F'         CALL    NETFUNC
0016'   CD 0034'  putnum:CALL    NETEND
0019'   0E 09            LD      C,PUTSTR
001B'   11 004D'         LD      DE,number
001E'   3A 0057'         LD      A,(WHO)
0021'   C6 30            ADD     A,'0'
0023'   32 0057'         LD      (WHO),A
0026'   CD 0005          CALL    BDOS
0029'   C9               RET
002A'   F7       NETINIT: RST    30h
002B'   8F               DB      8Fh
002C'   401C             DW      401Ch
002E'   C9               RET
002F'   F7       NETFUNC: RST    30h
```

```
0030'   8F              DB      8Fh
0031'   4019            DW      4019h
0033'   C9              RET
0034'   F7      NETEND: RST     30h
0035'   8F              DB      8Fh
0036'   4016            DW      4016h
0038'   C9              RET
                ; ---------------------
0039'   48 65 6C 6C text:DEFM   'Hello, fellows!'
003D'   6F 2C 20 66
0041'   65 6C 6C 6F
0045'   77 73 21
0048'   00        message:DB    0
0049'   0039'              DEFW  text
004B'   000F               DEFW  15
004D'   4E 75 6D  number: DEFM  'Number is '
0051'   62 65 72 20 69
0055'   73 20
0057'   00        WHO:    DB    0
0058'   24                DB    '$'
                          END
```

В заключение приведем листинг программы Host.mac. Эта программа состоит из двух частей. Первая часть, вызываемая оператором USR из MSX-BASICa по адресу &hDA00, инициализирует сеть; вторая часть, запускаемая по адресу &hDA03, после проверки сети функцией Check читает два байта из сетевой памяти компьютера ученика.

Номер компьютера ученика передается второй подпрограмме, вызов которой осуществляется следующим образом:

```
NC = ...            номер компьютера ученика
W = VARPTR( NC): W = USR(W)
IF W=0 THEN  ...    компьютер подключен, можно брать
                    содержимое ячеек по адресам &hF406,&hF407
IF NC>256 THEN ...  ученику разрешено передавать
                    сообщения другим ученикам, NC=NC-256
```

```
                Z80-Assembler   Page:    1
                  ORG   0DA00h
      ; === Вызовы двух частей программы:
DA00 C306DA         JP    Initial
DA03 C318DA         JP    ChkNet
401C =    NETINIT   EQU    401Ch
4019 =    NETFUNC   EQU    4019h
      ; === Инициирование сети
DA06 F7    Initial: RST    30h
DA07 8F             DB     8Fh
DA08 1C40           DW     NETINIT
      ; === Разрешение прерываний сети (INTON)
DA0A 0E01           LD    C,01
DA0C F7             RST   30h
```

```
DA0D 8F             DB    8Fh
DA0E 1940           DW    NETFUNC
       ; === Начало упорядоченного опроса (PON)
DA10 0E03           LD    C,03
DA12 F7             RST   30h
DA13 8F             DB    8Fh
DA14 1940           DW    NETFUNC
DA16 FB             EI
DA17 C9             RET
; ============================================================
; Программа проверки подключения к сети
; и чтения значений из сетевого ОЗУ ученика
; Вход:  сеть инициализирована
;        HL - адрес ячейки с номером компьютера NC
;             (при помощи передачи параметров 2F8Ah)
; Выход: IS_OFF - NC выключен
;        IS_ON  - NC включен
;        F406h,F407h - содержимое ячеек NRAM ученика
;        NC <= NC + 100h, если ученику можно работать в сети
; ============================================================
0000 = IS_ON        EQU   0
FFFF = IS_OFF       EQU   0FFFFh
7900 = FROM1        EQU   7900h
7901 = FROM2        EQU   7901h    ; сетевые адреса ученика
F406 = First        EQU   0F406h   ; RAM учителя
F407 = Second       EQU   0F407h
DA18 CD8A2F ChkNet: CALL  2F8Ah    ; взять аргумент, записать в HL
DA1B E5             PUSH  HL       ; запомнить адрес NC
    ; ===  Check: Кто подключен к сети ?
DA1C 0E17           LD    C,17h
DA1E F7             RST   30h
DA1F 8F             DB    8Fh
DA20 1940           DW    4019h
DA22 FB             EI
    ; === HL - подключены к сети, DE - разрешение работы
DA23 C1             POP   BC         ; адрес NC
DA24 C5             PUSH  BC
DA25 0A             LD    A,(BC)     ; A <-- NC
DA26 3D             DEC   A          ; A <-- NC-1
DA27 2821           JR    Z,ChkL     ; если NC=1
DA29 FE08           CP    8
DA2B FA43DA         JP    M,Chk2_7   ; если NC=2..7
DA2E D608           SUB   8          ; 0.6= NC #9..15
DA30 2807           JR    Z,ChkH     ; если NC=9
DA32 47             LD    B,A
DA33 CB3C NextB:    SRL   H
DA35 CB3A           SRL   D
DA37 10FA           DJNZ  NextB      ; сдвиги по NC
DA39 CB44 ChkH:     BIT   0,H        ; бит NC - нулевой
DA3B 201D           JR    NZ,C_OFF   ; компьютер отключен
DA3D CB42           BIT   0,D
DA3F 2020           JR    NZ,C_ON    ; диалог ученику запрещен
DA41 280F           JR    Z,ENACOM   ; диалог ученику разрешен
    ; === Проверка для компьютеров со 2-го по 7-й
```

```
DA43 47   Chk2_7:   LD    B,A         ; контроль NC = 2..7
DA44 CB3D           SRL   L
DA46 CB3B           SRL   E
DA48 10FA           DJNZ  Chk2_7+1    ; сдвиги по NC
DA4A CB45  ChkL:    BIT   0,L         ; бит NC - нулевой
DA4C 200C           JR    NZ,C_OFF    ; компьютер отключен
DA4E CB43           BIT   0,E
DA50 200F           JR    NZ,C_ON     ; диалог ученику запрещен
; === Установка флага "диалог разрешен": NC <= NC+100h
DA52 E1   ENACOM:   POP   HL
DA53 E5             PUSH  HL
DA54 23             INC   HL
DA55 3E01           LD    A,1
DA57 77             LD    (HL),A      ; флаг "работа разрешена"
DA58 1807           JR    C_ON        ; теперь - компьютер вкл.
    ;=== Компьютер отключен от сети
DA5A E1   C_OFF:    POP   HL
DA5B 21FFFF         LD    HL,IS_OFF
DA5E C3992F         JP    2F99h       ; возврат в MSX-BASIC
    ;=== Читаем значения из сетевых ячеек
DA61 E1   C_ON:     POP   HL          ; адрес NC
DA62 E5             PUSH  HL
DA63 7E             LD    A,(HL)
DA64 3298DA         LD    (Block),A     ; номер ученика
DA67 210079         LD    HL,FROM1      ; адрес 1-й ячейки,
DA6A 2299DA         LD    (Block+1),HL  ; откуда брать из NRAM
    ; === Вызов PEEK из NRAM ученика
DA6D 0E12           LD    C,12h
DA6F 1198DA         LD    DE,Block    ; адрес блока парам.
DA72 F7             RST   30h
DA73 8F             DB    8Fh
DA74 1940           DW    4019h
DA76 FB             EI
DA77 38E1           JR    C,C_OFF     ; если был сбой ввода/вывода
DA79 3206F4         LD    (First),A   ; записать в свою память
    ; ===  2-я ячейка
DA7C 210179         LD    HL,FROM2       ; адрес 2-й ячейки,
DA7F 2299DA         LD    (Block+1),HL   ; откуда брать из NRAM
    ; === Вызов PEEK из NRAM ученика
DA82 0E12           LD    C,12h
DA84 1198DA         LD    DE,Block       ; адрес блока параметров
DA87 F7             RST   30h
DA88 8F             DB    8Fh
DA89 1940           DW    4019h
DA8B FB             EI
DA8C 38CC           JR    C,C_OFF     ; если был сбой ввода/вывода
DA8E 3207F4         LD    (Second),A  ; записать в свою память
DA91 E1             POP   HL
DA92 210000         LD    HL,IS_ON    ; компьютер ученика включен
DA95 C3992F         JP    2F99h       ; возврат в MSX-BASIC
    ;=== Параметры сетевого вызова
DA98     Block:     DS    1,0    ; N ученика
DA99                DS    2,0    ; адрес ячейки
DA9B 0101           DB    1,1    ; сетевая память ученика
                    END
```

18. Работа с портами ввода/вывода

Перейдем к командам ввода/вывода. В процессоре Z-80 предусмотрен ряд команд, позволяющих осуществлять не только побайтовый ввод/вывод, но и ввод/вывод блока.

```
a) OUT   (порт),a        ; вывод в порт байта из A
b) IN    a,(порт)        ; ввод в A из порта
c) OUT   (c),r           ; вывести в порт, номер которого
                         ; в регистре C, содержимое регистра r
d) IN    r,(C)           ; ввести байт в регистр r из
                         ; порта, номер которого в регистре C
e) INI
     OUT (C),(HL)
     INC HL
     DEC B
f) IND
     OUT (C),(HL)
     DEC HL
     DEC B
g) INIR
     OUT (C),(HL)
     INC HL
     DEC B
     если B не равно 0, то повторить
h) INDR
     OUT (C),(HL)
     DEC HL
     DEC B
     если B не равно 0, то повторить
```

Перед работой с портами ввода/вывода рекомендуется отключать прерывания. Особенно это касается работы с портами видеопроцессора. Примеры работы с портами будут даны ниже.

19. Работа с видеорегистрами и видеопамятью

Вначале рассмотрим способы доступа к видеоинформации. Как уже говорилось, для записи информации в регистры видеопроцессора или видеопамять или чтения из них используются порты ввода/вывода - четыре для чтения и четыре для записи. Узнать номер первого порта для чтения можно в ячейке ПЗУ 00/0006, а для записи - в ячейке 00/0007.

Обычно для работы с VDP используются порты с номерами 98h..9Bh. При этом не забывайте в начале подпрограммы или перед ее вызовом отключать прерывания командой DI, а в конце работы с VDP - снова их активировать командой EI.

п.1. Порядок чтения и записи информации

Прямая запись в регистр видеопроцессора осуществляется в следующем порядке:

```
    ДАННЫЕ      ->  порт 99h
    10rr rrrr   ->  порт 99h  (r..r - номер регистра)
```

Например, запись в регистр VDP #2:

```
 MSX.M-80 1.00  01-Apr-85       PAGE    1
                .Z80
0000'   3E 03   LD      A,00000011b  ; PNT
0002'   D3 99   OUT     (99h),A
0004'   3E 82   LD      A,10000010b  ; VDP(2)
0006'   D3 99   OUT     (99h),A
0008'   C9      RET
                END
```

Еще один пример - подпрограмма записи в регистр VDP данных из регистра B, номер регистра VDP в регистре C:

```
                  Z80-Assembler   Page:    1
0000 F5  wrrvdp:PUSH    Af      ; сохранить A в стеке
0001 78         LD      A,b     ; выводим в порт 99h
0002 D399       OUT     (99h),A ; данные
0004 79         LD      A,c     ; выводим в порт 99h
0005 F680       OR      80h     ; номер регистра VDP
0007 D399       OUT     (99h),A ; выставив 7 бит в 1
0009 F1         POP     Af      ; вытаскиваем A из стека
000A C9         RET             ; возврат
                END
```

Косвенная запись в регистр видеопроцессора с автоматическим увеличением номера регистра осуществляется так:

```
    00rr rrrr           ->  R17         (r..r - номер регистра R)
    ДАННЫЕ для R        ->  порт 9Bh
    ДАННЫЕ для R + 1    ->  порт 9Bh
    ДАННЫЕ для R + 2    ->  порт 9Bh
            . . . .
```

Запись в регистр 17 осуществляется прямым способом, косвенный запрещен. Например, запись нуля в регистры 8,9:

```
      MSX.M-80 1.00   01-Apr-85       PAGE    1
                .Z80
0000'   3E 08     LD     A,8
0002'   D3 99     OUT    (99h),A
0004'   3E 91     LD     A,80h OR 17    ; VDP(17) <= 8
0006'   D3 99     OUT    (99h),A        ;
0008'   AF        XOR    A
0009'   D3 9B     OUT    (9Bh),A        ; VDP(8) <= 0
000B'   D3 9B     OUT    (9Bh),A        ; VDP(9) <= 0
000D'   C9        RET
                END
```

Косвенная запись в регистр видеопроцессора без автоматического увеличения номера регистра.

```
    10rr rrrr        ->  R17
    ДАННЫЕ для R     ->  порт 9Bh
    ДАННЫЕ для R     ->  порт 9Bh
      . . . .
```

Чтение из регистра статуса (состояния) видеопроцессора (0..9).

```
    0000 rrrr     ->  R15
    ДАННЫЕ        <-  порт 99h
```

После того как данные были прочитаны, необходимо записать в R#15 ноль и разрешить прерывания.

Например, чтобы узнать, было ли наложение двух спрайтов, можно проанализировать пятый бит регистра статуса #0:

```
      MSX.M-80 1.00   01-Apr-85       PAGE  1
                .Z80
0000'   AF        XOR   A
0001'   D3 99     OUT   (99h),A
0003'   3E 8F     LD    A,8Fh      ; VDP(15) <= 0
0005'   D3 99     OUT   (99h),A
0007'   DB 99     IN    A,(99h)
0009'   CB 6F     BIT   5,A        ; Было ли столкновение?
000B'   C9        RET
                END
```

Запись в регистры палитры

Регистры палитры (0..15) являются девятибитными. Поэтому запись в них осуществляется следующим образом:

```
 НОМЕР ПАЛИТРЫ   -> R16
 0rrr 0bbb       -> порт 9Ah   (rrr - КРАСНЫЙ, bbb - СИНИЙ)
 0000 0ggg       -> порт 9Ah   (ggg - ЗЕЛЕНЫЙ            )
```

После записи содержимое R#16 автоматически увеличивается на 1. Поэтому возможно простое обновление всех палитр.

Чтение/запись из/в видеопамяти VRAM/ERAM по двоичному адресу b bbhh hhhh cccc cccc.

а) Установить банк VRAM:

```
00.. ....    -> R45   (для VRAM)
01.. ....    -> R45   (для ERAM)  [ в MSX-2 отсутствует ]
```

Содержимое регистра R45 не меняется при обращении к памяти, поэтому нет необходимости каждый раз переопределять шестой бит.

б) Установить адрес видеопамяти:

```
  0000 0bbb   -> R14
  cccc cccc   -> порт 99h
  00hh hhhh   -> порт 99h    (для чтения из видеопамяти)
  01hh hhhh   -> порт 99h    (для записи в  видеопамять)
```

в) Писать в порт 98h последовательные байты данных или читать из этого порта в зависимости от выбранного режима. Адрес видеопамяти при этом автоматически увеличивается. Для доступа к VRAM можно также использовать соответствующие команды VDP.

Например, необходимо прочитать из видеопамяти 200 байт и записать их, начиная с адреса 0C000h; начальный адрес видеопамяти равен 0:

```
       MSX.M-80 1.00   01-Apr-85       PAGE    1
                     .Z80
       ; === Установка банки VRAM/ERAM
0000'   AF           XOR     A
0001'   D3 99        OUT     (99h),A
0003'   3E AD        LD      A,80h OR 45 ; VDP(45) <= 0
0005'   D3 99        OUT     (99h),A
0007'   AF           XOR     A
0008'   D3 99        OUT     (99h),A
000A'   3E 8E        LD      A,8Eh       ; VDP(14) <= 0
000C'   D3 99        OUT     (99h),A
       ; === Копируем
000E'   21 0000      LD      HL,0    ; начальный адрес VRAM
0011'   11 C000      LD      DE,0C000h ; начальный адрес RAM
0014'   06 C8        LD      b,200   ; длина блока
0016'   7D           LD      A,L     ; адрес начала памяти
0017'   D3 99        OUT     (99h),A ; младш. байт адреса VRAM
0019'   7C           LD      A,h
001A'   D3 99        OUT     (99h),A ; старший байт
001C'   DB 98 blreAd:IN      A,(98h) ; вводим байт из ук.адр.
001E'   12           LD      (DE),A  ; записываем в память
001F'   13           INC     DE      ; подгот. след.адрес RAM
```

```
0020'   10 FA        DJNZ    blreAd  ; b=b-1, если b<>0,
                                    ; то повторить blreAd
0022'   C9          RET
                    END
```

Эту подпрограмму можно написать и по другому:

```
                  Z80-Assembler   Page:    1
0000 210000        LD      HL,0    ; начальный адрес VRAM
0003 1100C0        LD      DE,0C000h ; начальный адрес RAM
0006 06C8          LD      b,200   ; длина блока
0008 EB            EX      DE,HL   ; обменять HL и DE
0009 7B            LD      A,e     ; адрес начала памяти
000A D399          OUT     (99h),A ; младш. байт адреса VRAM
000C 7A            LD      A,d
000D D399          OUT     (99h),A ; старший байт
000F 0E98          LD      c,98h   ; номер порта вв./вывода
0011 EDB2          INIR            ; ввести данные
0013 C9            RET
                   END
```

При изменении типа экрана часто требуется восстанавливать таблицу шаблонов (образов) символов PGT. Ее можно извлечь из ROM BIOS по адресу, который записан в системной области в трехбайтовой ячейке F91Fh (слот + адрес ROM PGT). Обычно адрес ROM PGT равен 00/1BBFh.

Теперь попробуем написать подпрограмму пересылки блока данных из ROM PGT в VRAM PGT. Ранее для подобных действий мы пользовались подпрограммами BIOS.

```
                Z80-Assembler   Page:    1
                ORG 9000h
   ; ==== ROM PGT => VRAM PGT
   ; [HL] - адрес ROM PGT
   ; [DE] - адрес VRAM PGT
   ; [bc] - длина блока
9000 21BF1B       LD     HL,1BBFh ; ROM PGT
9003 110010       LD     DE,1000h ; TEXT-2 PGT
9006 010008       LD     BC,2048  ; длина
9009 7B           LD     A,E      ; выбрасываем младший
900A D399         OUT    (99h),A  ; байт адреса VRAM
900C 7A           LD     A,D      ; выбрасываем старший
900D F640         OR     40h      ; байт адреса VRAM,
900F D399         OUT    (99h),A  ; установив 6 бит в 1
9011 7E   LDirmv: LD     A,(HL)   ; запис. по адресу VRAM
9012 D398         OUT    (98h),A  ; данные из (HL)
9014 23           INC    HL       ; следующий адрес RAM
9015 0B           DEC    BC       ; уменьшаем длину
9016 78           LD     A,B      ; если длина не равна 0,
9017 B1           OR     C        ;    то повторить
9018 20F7         JR     NZ,LDirmv
901A C9           RET
            END
```

В завершение параграфа приведем пример достаточно большой программы, устанавливающей 80-символьный текстовый режим. В верхней части экрана мигает блок с текстом "width 80". Выход из программы - по CTRL/STOP.

```
            Z80-Assembler   Page:    1
               ORG 9000h
9000 F3        DI                  ; отменяем прерывания
    ; === Установка текстового режима 80 символов
    ;     Регистры VDP 0,1,8,9
9001 3E04      LD      A,00000100b
9003 D399      OUT     (99h),A
9005 3E80      LD      A,10000000b    ; VDP(0)
9007 D399      OUT     (99h),A
9009 3E70      LD      A,01110000b
900B D399      OUT     (99h),A
900D 3E81      LD      A,10000001b    ; VDP(1)
900F D399      OUT     (99h),A
9011 AF        XOR     A
9012 D399      OUT     (99h),A
9014 3E88      LD      A,10001000b    ; VDP(8)
9016 D399      OUT     (99h),A
9018 AF        XOR     A
```

```
9019 D399      OUT     (99h),A
901B 3E89      LD      A,10001001b    ; VDP(9)
901D D399      OUT     (99h),A
    ; === Установка базовых адресов PNT, PGT, CT
901F 3E03      LD      A,00000011b    ; PNT
9021 D399      OUT     (99h),A
9023 3E82      LD      A,10000010b    ; VDP(2) <= 0
9025 D399      OUT     (99h),A        ;
9027 3E02      LD      A,00000010b    ; PGT
9029 D399      OUT     (99h),A
902B 3E84      LD      A,10000100b    ; VDP(4) <= 2 (* 800h)
902D D399      OUT     (99h),A        ;
902F AF        XOR     A              ; CT
9030 D399      OUT     (99h),A
9032 3E8A      LD      A,10001010b    ; VDP(10) <= 0
9034 D399      OUT     (99h),A        ;
9036 3E27      LD      A,00100111b    ; CT
9038 D399      OUT     (99h),A
903A 3E83      LD      A,10000011b    ; VDP(3) <= 27h
903C D399      OUT     (99h),A        ;
    ; === Установка цветов и мигания
903E 3EFC      LD      A,11111100b    ; цвета текста и фона
9040 D399      OUT     (99h),A
9042 3E87      LD      A,87h          ; VDP(7) <= 15,12
9044 D399      OUT     (99h),A        ;
9046 3E1D      LD      A,00011101b    ; цвета для мигания
9048 D399      OUT     (99h),A
904A 3E8C      LD      A,8Ch          ; VDP(12) <= 1,13
904C D399      OUT     (99h),A        ;
904E 3E77      LD      A,01110111b    ; время вкл/выкл мигания
9050 D399      OUT     (99h),A
9052 3E8D      LD      A,8Dh          ; VDP(13)
9054 D399      OUT     (99h),A        ;
    ; === Установка банки VRAM/ERAM
9056 AF        XOR     A
9057 D399      OUT     (99h),A
9059 3EAD      LD      A,80h OR 45    ; VDP(45) <= 0
905B D399      OUT     (99h),A        ;
905D AF        XOR     A
905E D399      OUT     (99h),A
9060 3E8E      LD      A,8Eh          ; VDP(14) <= 0
9062 D399      OUT     (99h),A        ;
    ; ==== ROM PGT => VRAM PGT
    ; [HL] - адрес ROM PGT
    ; [DE] - адрес VRAM PGT
    ; [bc] - длина блока
9064 21BF1B    LD      HL,1BBFh
9067 110010    LD      DE,1000h
906A 010008    LD      BC,2048
906D 7B        LD      A,e     ; выбрасываем младший
906E D399      OUT     (99h),A ; байт адреса VRAM
9070 7A        LD      A,d     ; выбрасываем старший
9071 F640      OR      40h     ; байт адреса VRAM,
9073 D399      OUT     (99h),A ; установив 6 бит в 1
9075 7E   LDirmv: LD   A,(HL)  ; запис. по адресу VRAM
```

```
9076 D398      OUT     (98h),A ; данные из (HL)
9078 23        INC     HL      ; следующий адрес RAM
9079 0B        DEC     bc      ; уменьшаем длину
907A 78        LD      A,b     ; если длина не равна 0,
907B B1        OR      C       ;    то повторить
907C 20F7      JR      NZ,LDirmv
    ; ==== Очистить VRAM CT нулем (нет мигания)
    ; [DE] - адрес VRAM
    ; [bc] - длина блока
907E 110008    LD      DE,800h
9081 010E01    LD      BC,270
9084 7B        LD      A,E        ; выбрасываем младший
9085 D399      OUT     (99h),A    ; байт адреса VRAM
9087 7A        LD      A,D        ; выбрасываем старший
9088 F640      OR      40h        ; байт адреса VRAM,
908A D399      OUT     (99h),A    ; установив 6 бит в 1
908C AF   LDirCT: XOR  A          ; запис. по адресу VRAM
908D D398      OUT     (98h),A    ;    ноль
908F 0B        DEC     BC         ; уменьшаем длину
9090 78        LD      A,B        ; если длина не равна 0,
9091 B1        OR      C          ;    то повторить
9092 20F8      JR      NZ,LDirCT
    ; ==== Мерцание блока
9094 3E0E      LD      A,14    ; выбрасываем младший
9096 D399      OUT     (99h),A ; байт адреса VRAM
9098 3E08      LD      A,08h   ; выбрасываем старший
909A F640      OR      40h     ; байт адреса VRAM,
909C D399      OUT     (99h),A ; установив 6 бит в 1
909E 3E1F      LD      A,1Fh   ; запис. по адресу VRAM
90A0 D398      OUT     (98h),A ;    ноль
90A2 3E0F      LD      A,15    ; выбрасываем младший
90A4 D399      OUT     (99h),A ; байт адреса VRAM
90A6 3E08      LD      A,08h   ; выбрасываем старший
90A8 F640      OR      40h     ; байт адреса VRAM
90AA D399      OUT     (99h),A ; установив 6 бит в 1
90AC 3EF8      LD      A,0F8h  ; запис. по адресу VRAM
90AE D398      OUT     (98h),A ;    ноль
    ; ==== Пробел (20h) => VRAM PNT
    ; [DE] - адрес VRAM
    ; [bc] - длина блока
90B0 110000    LD      DE,0
90B3 018007    LD      BC,1920
90B6 7B        LD      A,E        ; выбрасываем младший
90B7 D399      OUT     (99h),A    ; байт адреса VRAM
90B9 7A        LD      A,D        ; выбрасываем старший
90BA F640      OR      40h        ; байт адреса VRAM,
90BC D399      OUT     (99h),A    ; установив 6 бит в 1
90BE 3E20 LDiPNT: LD   A,20h      ; запис. по адресу VRAM 20h
90C0 D398      OUT     (98h),A
90C2 0B        DEC     BC         ; уменьшаем длину
90C3 78        LD      A,B        ; если длина не равна 0,
90C4 B1        OR      C          ; то повторить
90C5 20F7      JR      NZ,LDiPNT
```

```
    ; ==== Надпись из RAM => VRAM PNT
    ; [HL] - адрес RAM
    ; [DE] - адрес VRAM
    ; [BC] - длина блока
90C7 21F190    LD      HL,tit
90CA 117400    LD      DE,116
90CD 019808    LD      BC,0898h
90D0 7B        LD      A,E     ; выбрасываем младший
90D1 D399      OUT     (99h),A ; байт адреса VRAM
90D3 7A        LD      A,D     ; выбрасываем старший
90D4 F640      OR      40h     ; байт адреса VRAM
90D6 D399      OUT     (99h),A ; установив 6 бит в 1
90D8 EDB3      OTIR            ; переписываем блок
    ; === Ширина экрана  - 80 символов
90DA 3E50      LD      A,80
90DC 32B0F3    LD      (0F3B0h),A
    ; === Локализуем курсор в (1,1)
90DF 3E01      LD      A,1
90E1 32A9FC    LD      (0FCA9h),A
90E4 210101    LD      HL,0101h
90E7 CDC600    CALL    0C6h
    ; === Ждем нажатия CTRL/STOP
90EA CDB700 Again:  CALL 0B7h
90ED 30FB      JR      NC,Again
90EF FB        EI
90F0 C9        RET
         tit: DB   'Width 80'
90F1 57696474
90F5 68203830
               END
```

п.2. Использование команд видеопроцессора

MSX-VIDEO выполняет основные графические операции, которые называются командами VDP. Они доступны в режимах VDP GRAPHIC 4..7, а для работы с ними используются специальные регистры VDP.

При работе с командами VDP используется особая координатная сетка. В ней нет деления на страницы, доступны все 128 KB VRAM. Она приведена на рис.19.1.

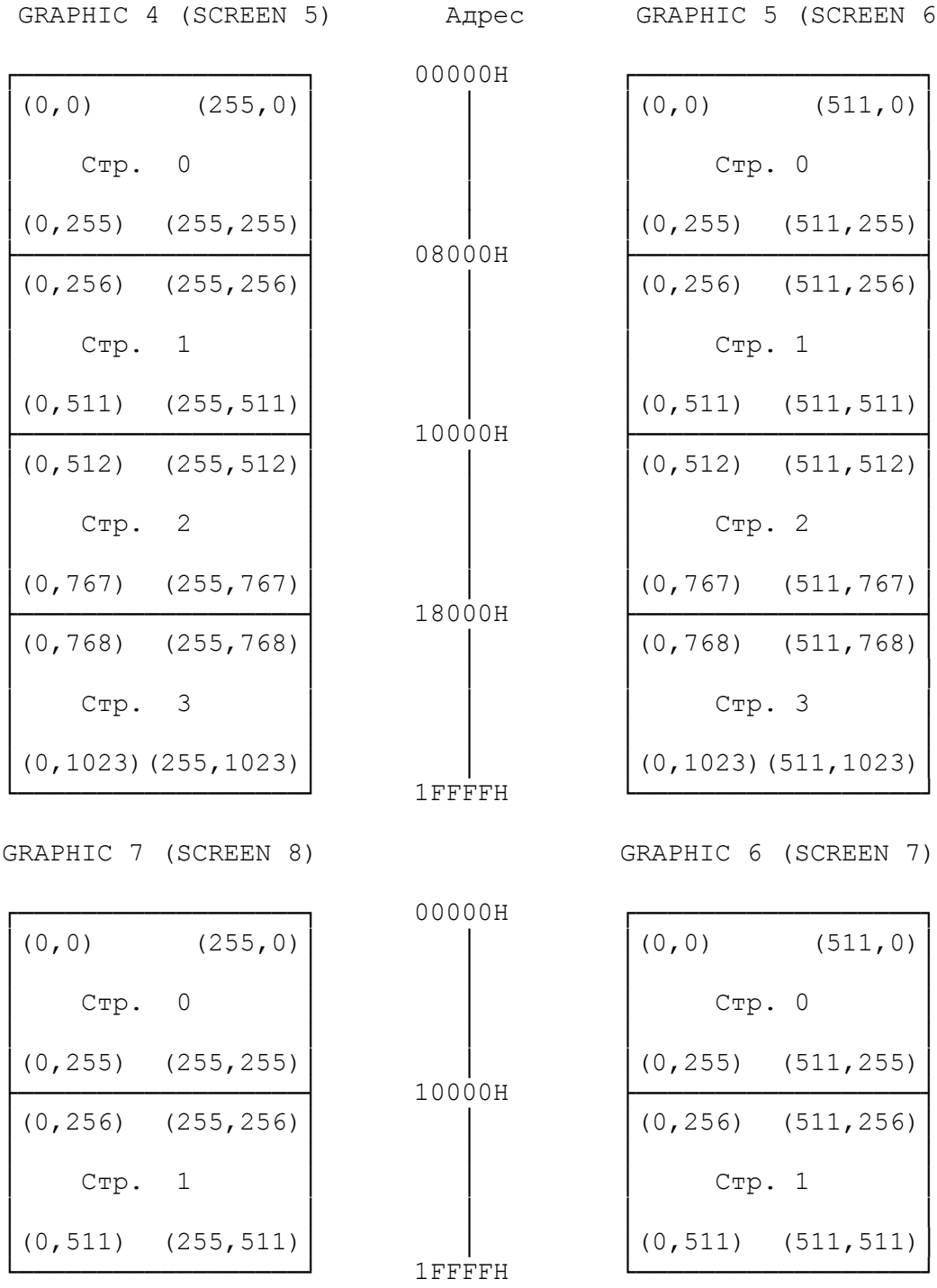


Рис.19.1. Координатная система VRAM

Имеется 12 типов команд VDP. Они были описаны в приложении к книге "Архитектура микрокомпьютера MSX-2". Параметры для выполнения команды записываются в регистры с 32-го по 45-й. Команда начинает выполняться после установки в 1 нулевого бита регистра статуса #2. После того как выполнение команды закончится, этот бит сбрасывается в ноль.

Для прекращения выполнения текущей команды можно выполнить команду VDP STOP.

Для ускорения выполнения команд VDP рекомендуется на время запрещать отображение спрайтов путем установки первого бита регистра #8. Можно также отключать при начальной загрузке изображение на экране.

Приведем пример программы, выполняющей команду HMMC VDP быстрой пересылки RAM => VRAM.

```
     MSX.M-80 1.00   01-Apr-85       PAGE  1
                     .Z80
               ; === Выполнение команды VDP HMMC
C000           ABegin EQU    0C000h
0000'                 ASEG
0000 FE               DB      0FEh      ; файл типа Obj
0001 C000             DW      ABegin    ; загрузочный адрес
0003 C08E             DW      AEnd      ; конечный адрес
0005 C000             DW      AStart    ; стартовый адрес
                      .Phase ABegin
C000           Astart EQU    $
0000           XK     EQU    0
0000           YK     EQU    0
0064           XS     EQU    100
0064           YS     EQU    100
               ; === Вызов HMMC
C000 DD 21 C07F       LD      ix,DataForVram
C004 21 0000          LD      HL,XK*100h+YK
C007 11 6464          LD      DE,XS*100h+YS
               ; === Пересылка RAM [IX] => VRAM (H,L)-(D,E)
C00A CD C00E          CALL    HMMC
C00D C9               RET
               ; === Команда VDP HMMC
C00E F3       HMMC:   DI              ; запрет прерываний
C00F CD C071          CALL    WaitVDP ; ожидание конца работы команды
C012 3E 24            LD      A,36    ; номер первого регистра
C014 D3 99            OUT     (99h),A
C016 3E 91            LD      A,17+80h
C018 D3 99            OUT     (99h),A ; VDP(17) <= 36
C01A 0E 9B            LD      c,9Bh   ; C=9Bh для косвенного доступа
C01C AF               XOR     A       ;    к регистрам VDP
C01D ED 61            OUT     (c),H   ; X младший байт
C01F ED 79            OUT     (c),A   ; X старший байт
C021 ED 69            OUT     (c),L   ; Y младший байт
C023 ED 79            OUT     (c),A   ; Y старший байт
C025 ED 51            OUT     (c),D
```

```
C027 ED 79        OUT    (c),A    ; NX
C029 ED 59        OUT    (c),E
C02B ED 79        OUT    (c),A    ; NY
C02D DD 66 00     LD     h,(IX)
C030 ED 61        OUT    (c),H    ; первое данное
C032 ED 79        OUT    (c),A    ; регистр аргумента
C034 3E F0        LD     A,11110000b
C036 ED 79        OUT    (c),A    ; приказ выполнить команду HMMC
C038 3E AC        LD     A,44 or 80h
C03A D3 99        OUT    (99h),A
C03C 3E 91        LD     A,17 OR 80h
C03E D3 99        OUT    (99h),A  ; VDP(17) <= 44
C040 3E 02  LOOP: LD     A,2
C042 CD C05D      CALL   GetStatus
C045 CB 47        BIT    0,A      ; проверить бит CE
C047 28 0D        JR     z,Exit   ; конец
C049 CB 7F        BIT    7,A      ; проверить бит TR
C04B 28 F3        JR     z,LOOP
C04D DD 23        INC    ix
C04F DD 7E 00     LD     A,(ix)
C052 D3 9B        OUT    (9Bh),A
C054 18 EA        JR     LOOP
C056 3E 00  Exit: LD       A,0
C058 CD C05D      CALL   GetStatus ; берем статус
C05B FB           EI               ; выход
C05C C9           RET
C05D    GetStatus:
C05D D3 99        OUT    (99h),A   ; регистр статуса
C05F 3E 8F        LD     A,8Fh
C061 D3 99        OUT    (99h),A
C063 E5           PUSH   HL
C064 E1           POP    HL
C065 DB 99        IN     A,(99h)
C067 F5           PUSH   Af
C068 AF           XOR    A
C069 D3 99        OUT    (99h),A
C06B 3E 8F        LD     A,8Fh
C06D D3 99        OUT    (99h),A
C06F F1           POP    Af
C070 C9           RET
C071     WaitVDP:
C071 3E 02        LD     A,2       ; ждать, пока VDP не готов
C073 CD C05D      CALL   GetStatus
C076 E6 01        AND    1
C078 20 F7        JR     NZ,WaitVDP
C07A AF           XOR    A
C07B CD C05D      CALL   GetStatus
C07E C9           RET
C07F  DataForVram EQU $
C07F 00 11 22 33  DB     00,11h,22h,33h,44h,55h,66h,77h,88h,99h
C083 44 55 66 77
C087 88 99 AA
C08A BB CC DD EE  DB     0AAh,0BBh,0CCh,0DDh,0EEh,0FFh
C08E FF
C08E       AEnd    EQU    $-1
```

```
                .DePhase
                END
```

20. Программирование шумов и музыки

В первой главе мы уже говорили о возможностях работы с программируемым звуковым генератором PSG . Здесь мы покажем Вам примеры программ на языке ассемблера, работающих с PSG.

Напомним, что для задания номера регистра PSG используется порт A0h, а для записи значения для этого регистра - порт A1h.

Приведем листинг программы, воспроизводящей звук летящего бомбардировщика. В этой программе запись в регистры PSG производится в цикле, в обратном порядке.

```
          MSX.M-80 1.00 01-Apr-85 PAGE 1
                    .Z80
    ; === Звук летящего бомбардировщика
0000' 3E 0D         LD      A,13        ; запись в регистры
0002' 21 0021'      LD      HL,BombSnd  ; в обратном порядке
0005' F3      Next: DI                  ; необх. запрет прерываний
0006' D3 A0         OUT     (0A0h),A    ; номер регистра
0008' F5            PUSH    AF          ; запоминаем A
0009' 7E            LD      A,(HL)      ; значение для регистра PSG
000A' D3 A1         OUT     (0A1h),A    ; записываем данные
000C' FB            EI                  ; можно восст. прерывания
000D' F1            POP     AF          ; восстанавливаем A
000E' 2B            DEC     HL          ; новый адрес
000F' D6 01         SUB     1           ; следующий регистр PSG
0011' 30 F2         JR      NC,Next     ; повторить
0013' C9            RET
; ====       регистры NN: ── 0 ─ 1─ 2 ─ 3─ 4 ─ 5 6 ──────
0014' C8 0E DC 0E   DefB    200,14,220,14,240,14,0,
0018' F0 0E 00 B8   ; NN: ────── 7 ──── 8  9 10 11 12 13 ─
001C' 0F 0F 0F 00   DB      10111000b,15,15,15, 0, 0, 0
0020' 00 00
0021'    BombSnd    EQU     $-1
                    END
```

Еще один пример - листинг программы, воспроизводящей звук сирены:

```
┌──────────────────────────
            MSX.M-80 1.00 01-Apr-85 PAGE 1
; === Звук сирены
                    .Z80
0000' F3            DI
0001' AF            XOR   a             ; запись 255 => 0 рег.
0002' D3 A0         OUT   (0A0h),a      ; номер регистра
0004' 3E FF         LD    a,255
0006' D3 A1         OUT   (0A1h), a     ; данные
0008' 3E 01         LD    a,1           ; запись 0 => 1 рег.
000A' D3 A0         OUT   (0A0h),a      ; номер регистра
000C' AF            XOR   a
000D' D3 A1         OUT   (0A1h),a      ; данные
000F' 3E 08         LD    a,8           ; запись 8 => 8 рег.
0011' D3 A0         OUT   (0A0h),a      ; номер регистра
0013' D3 A1         OUT   (0A1h),a      ; данные
0015' 3E 07         LD    a,7           ; запись упр => 7 рег.
0017' D3 A0         OUT   (0A0h),a      ; номер регистра
0019' 3E 3E         LD    a,00111110b
001B' D3 A1         OUT   (0A1h),a      ; данные
001D' FB            EI
     ; === Подъем звука (257..170)
001E' 3E FE         LD    a,254         ; запись в регистр
0020' 3D   NextA:   DEC   a             ; уменьшить на 2
0021' 3D            DEC   a
0022' F5            PUSH  af            ; сохранить
0023' 21 0090       LD    HL,90h        ; задержка времени
00026' 2B   timer:  DEC   HL
0027' 7C            LD    a,h
0028' B5            OR    L
0029' 20 FB         JR    NZ,timer
002B' AF            XOR   a
002C' D3 A0         OUT   (0A0h),a      ; номер регистра = 0
002E' F1            POP   af            ; восстановить A
002F' D3 A1         OUT   (0A1h),a      ; данные
0031' FE AA         CP    170           ; проверка
0033' 20 EB         JR    NZ,NextA      ; повторить
     ; === Падение звука (170..252)
0035' 3E AA         LD    a,170         ; запись в регистр
0037' 3C   NextB:   INC   a             ; увеличить
0038' F5            PUSH  AF            ; сохранить
0039' 21 05A0       LD    HL,90h*10     ; задержка больше,
003C' 2B   timer1:  DEC   HL            ; чем для роста звука
003D' 7C            LD    a,h           ; в 10 раз
003E' B5            OR    L
003F' 20 FB         JR    NZ,timer1
0041' AF            XOR   a
0042' D3 A0         OUT   (0A0h),a      ; номер регистра = 0
0044' F1            POP   af            ; восстановить
0045' D3 A1         OUT   (0A1h),a      ; данные
0047' FE FC         CP    252           ; проверка
0049' 20 EC         JR    NZ,NextB      ; повторить
004B' C3 0020'      JP    NextA         ; все снова
                    END
```

└──────────────────────

Теперь приведем пример программы, проигрывающей несколько нот. В регистры звукогенератора записываются коды, соответствующие обозначениям нот и их октаве.

```
┌──────────────────────
        MSX.M-80 1.00 01-Apr-85 PAGE 1
                 .Z80
0000' 3E 01      LD    a,1         ; запись 0 => 1 рег
0002' D3 A0      OUT   (0A0h),a
0004' AF         XOR   a
0005' D3 A0      OUT   (0A0h),a
0007' 3E 07      LD    a,7         ; разрешить звук
0009' D3 A0      OUT   (0A0h),a    ; канала A
000B' 3E 3E      LD    a,00111110b
000D' D3 A1      OUT   (0A1h),a
000F' 3E 08      LD    a,8         ; макс. звук для A
0011' D3 A0      OUT   (0A0h),a
0013' 3E 0F      LD    a,15
0015' D3 A1      OUT   (0A1h),a
0017' AF         XOR   a           ; запись 190 => 0
0018' D3 A0      OUT   (0A0h),a    ; нота D, октава 5
001A' 3E BE      LD    a,190
001C' D3 A1      OUT   (0A1h),a
001E' CD 0057'   CALL  timer
0021' AF         XOR   a           ; запись 214 => 0
0022' D3 A0      OUT   (0A0h),a    ; нота C, октава 5
0024' 3E D6      LD    a,214
0026' D3 A1      OUT   (0A1h),a
0028' CD 0057'   CALL  timer
002B' AF         XOR   a           ; запись 227 => 0
002C' D3 A0      OUT   (0A0h),a    ; нота B, октава 4
002E' 3E E3      LD    a,227
0030' D3 A1      OUT   (0A1h),a
0032' CD 0057'   CALL  timer
0035' CD 0057'   CALL  timer
0038' AF         XOR   a           ; запись 254 => 0
0039' D3 A0      OUT   (0A0h),a    ; нота A, октава 4
003B' 3E FE      LD    a,254
003D' D3 A1      OUT   (0A1h),a
003F' CD 0057'   CALL  timer
0042' CD 0057'   CALL  timer
0045' AF         XOR   a           ; запись 29  => 0
0046' D3 A0      OUT   (0A0h),a    ; нота G, октава 4
0048' 3E 1D      LD    a,29
004A' D3 A1      OUT   (0A1h),a
004C' 3E 01      LD    a,1         ; запись 1   => 1
004E' D3 A0      OUT   (0A0h),a
0050' D3 A1      OUT   (0A1h),a
0052' F7         RST   30h         ; BEEP, чистка регистров
0053' 00         DB    0
0054' 00C0       DW    0C0h
0056' C9         RET
    ; === Задержка звучания ноты
```

```
0057' 21 6000 timer: LD HL,6000h    ; задержка
005A' 2B  again:  DEC   HL
005B' 7C          LD    A,H
005C' B5          OR    L
005D' 20 FB       JR    NZ,again
005F' 3E 08       LD    A,8         ; гашение звука
0061' D3 A0       OUT   (0A0h),A
0063' 3E 00       LD    A,0
0065' D3 A1       OUT   (0A1h),A
0067' 21 1000     LD    HL,1000h    ; задержка, пауза
006A' 2B  again1: DEC   HL          ; перед следующей нотой
006B' 7C          LD    A,H
006C' B5          OR    L
006D' 20 FB       JR    NZ,again1
006F' 3E 08       LD    A,8         ; макс. звук для A
0071' D3 A0       OUT   (0A0h),A
0073' 3E 0F       LD    A,15
0075' D3 A1       OUT   (0A1h),A
0077' C9          RET
                  END
```

21. Управление памятью

Для управления логической памятью и размещением страниц в слотах и вторичных слотах используются порт A8h и ячейка RAM с адресом FFFFh. Для управления физической памятью изменяется содержимое портов ввода/вывода FCh...FFh. Эти механизмы были описаны в книге "Архитектура микрокомпьютера MSX-2", и их программирование не требует больших усилий.

Более мобильное управление памятью осуществляется при помощи специальных подпрограмм BIOS (их поддерживает и MSX-DOS):

WRSLT (0014h) Запись значения по указанному адресу в указанном слоте.

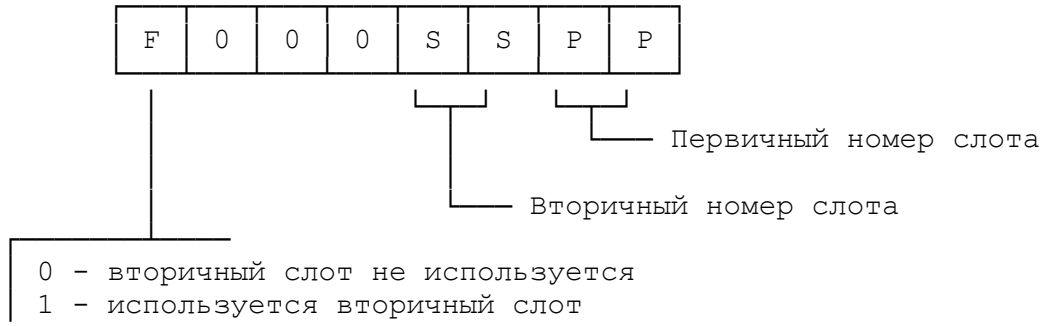
RDSLT (000Ch) Чтение значения по указанному адресу из указанного слота.

ENASLT (0024h) Выбор и активация слота.

Запись числа в RAM любого слота может быть осуществлена следующим образом:

```
LD   A,указатель-слота
LD   HL,адрес-ячейки
LD   E,значение-для-записи
CALL WRSLT
```

Указатель слота имеет вид:



Аналогично выглядит чтение значения:

```
        LD   A,указатель-слота
        LD   HL,адрес-для-чтения
        CALL RDSLT
        LD   (адрес-результата), A
```

Для активации слота используется подпрограмма ENASLT (0024h). Ее вызов имеет вид:

```
        LD   A,указатель-слота
        LD   HL,начальный-адрес
        CALL ENASLT
```

Такой вызов удобен для активации нулевой и первой страницы памяти.

Помните, что обычные команды записи LD, LDIR работают быстрее, чем межслотовая запись. Поэтому иногда лучше переключить слоты, переписать данные и восстановить исходное состояние слотов.

п.1. Работа с кассетами (картриджами)

Компьютер MSX обычно имеет по крайней мере один внешний слот. Та аппаратура (hardware), которая к нему подключается, называется кассетой, или картриджем (cartridge). Существуют кассеты ROM для прикладных программ и игр, дискового ввода/вывода, интерфейса RS232C, расширения памяти RAM и слотов расширения.

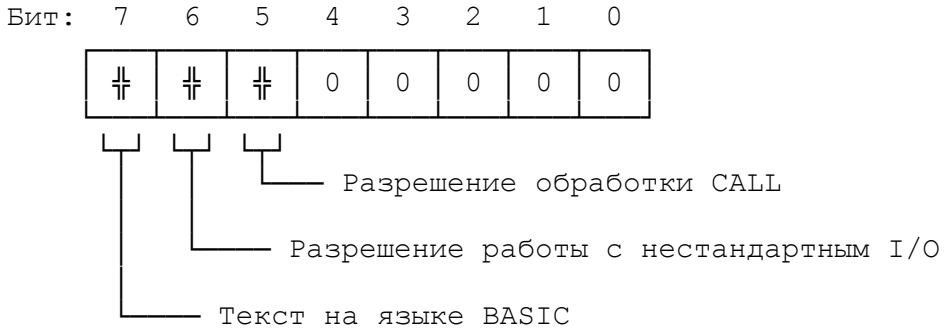
В кассету может быть аппаратно записано (ROM) программное обеспечение на языке BASIC или на языке ассемблера. Кроме этого, в подходящем слоте RAM можно создать "псевдокартридж" программным способом.

В рабочей области, начиная с адреса FCC9h, находится участок памяти, отвечающий за каждую страницу памяти, находящейся в некотором слоте. Адрес байта рабочей области, отвечающего за некоторую страницу памяти, вычисляется по формуле:

```
Addr = FCC9h + 16*SLTNUM + 4*EXTSLT + PageNmb
```

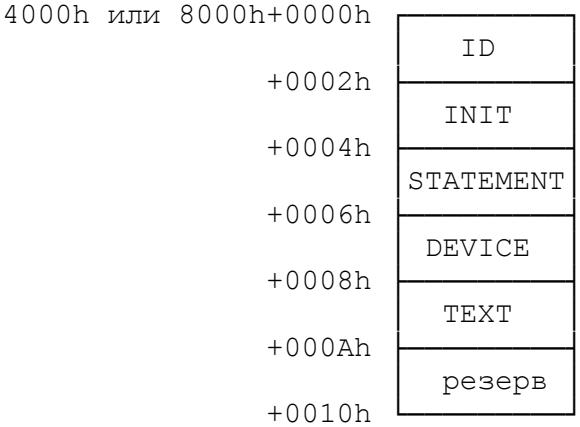
где SLTNUM - номер первичного слота;
EXTSLT - номер вторичного слота (слота расширения);
PageNmb - номер логической страницы памяти.

По этому адресу содержится информация о том, работу каких устройств могут поддерживать программы, размещенные в соответствующей странице памяти. Информация кодируется побайтно следующим образом:



Таким образом, например, ячейка RAM с адресом FD02h отвечает за страницу 1 слота 3-2, т.е. за страницу RAM по адресу 4000h. Запись числа 32 в ячейку FD02h означает разрешение обработки расширенного оператора CALL MSX-BASICa подпрограммами RAM по адресу 4000h.

MSX-BASIC просматривает все слоты (включая вторичные) по адресам с 4000H по 0BFFFH для нахождения ID устройства, начинающего каждую страницу. Формат заголовка кассеты, содержащего ID, приведен ниже.



ID - это двухбайтовая строка, при помощи которой можно отличить картридж ROM или SUB-ROM от пустой страницы. Картридж ROM обозначается строкой "AB" (41h,42h), а картридж SUB-ROM - строкой "CD".

INIT содержит адрес процедуры инициализации этого картриджа. Ноль записывается, если такой процедуры нет. Программы, которые нуждаются в связи с интерпретатором языка BASIC, возвращают управление командой Z-80 "RET". Все регистры, за исключением указателя стека SP, могут быть изменены. Для некоторых программ (например, для игр) соблюдать соглашение о вызове INIT не нужно, поэтому игры могут запускаться процедурой инициализации.

STATEMENT содержит адрес обработки расширенного оператора CALL, если его обработка в картридже предусмотрена. Ноль записывается в том случае, если обработки оператора CALL нет.

Когда BASIC встречает оператор 'CALL', то он записывает его имя в PROCNM (FD89h), в регистр HL - указатель на текст, следующий за CALL (список параметров), и вызывает адрес STATEMENT.

Картридж может быть расположен по адресам с 4000H по 7FFFH.

Синтаксис оператора расширения CALL:

CALL <statement_name> [(<arg>[,<arg>] ...)]

Слово CALL может быть заменено на символ подчеркивания (_).

Имя оператора CALL записывается в системную память и заканчивается нулевым кодом. Так как буфер PROCNM имеет фиксированную длину 16 байт, то имя оператора может иметь длину не более 15 символов.

Если обработчика требуемого оператора CALL в данном картридже не содержится, то устанавливается флаг C и управление возвращается в BASIC. Содержимое HL должно быть возвращено неизмененным.

В этом случае интерпретатор языка BASIC пытается вызвать другой слот расширения. Если ни один слот "не отзовется", генерируется сообщение об ошибке - "Syntax error".

Если обработчик для конкретного оператора CALL содержится в картридже, то можно его обработать (выполнить), после чего необходимо [HL] установить на конец оператора CALL. Обычно это нулевой код, означающий конец строки, или код ':', означающий конец оператора. Флаг C должен быть сброшен. Все регистры, за исключением SP, могут быть изменены.

DEVICE содержит адрес подпрограммы обработки устройства расширения, если она есть в этом картридже, в противном случае в DEVICE хранится ноль.

Картридж может иметь адреса в диапазоне с 4000H по 7FFFH и до четырех логических имен устройств.

Когда BASIC встречает имя устройства (например, OPEN"OPT:"...), то он записывает его в PROCNM (FD89h), код FFh - в аккумулятор и передает управление на картридж с наименьшим номером слота.

Если обработка устройства с этим именем в картридже не предусмотрена, то устанавливается флаг C и происходит возврат в BASIC. Если все картриджи возвратили флаг C, генерируется ошибка "Bad file name".

Если подпрограмма обработки устройства содержится в картридже, то она выполняется, затем ID устройства (от 0 до 3) записывается в аккумулятор, сбрасывается флаг C, и выполняется возврат. Все регистры могут быть изменены.

Когда выполняются реальные операции ввода/вывода, BASIC-интерпретатор записывает ID устройства (0-3) в ячейку DEVICE (FD99h), записывает запрос к устройству в регистр A (см.табл.21.1) и вызывает подпрограмму расширения устройства в картридже. Эта подпрограмма и должна правильно обработать запрос.

Регистр A	Запрос
0	OPEN
2	CLOSE
4	Прямой доступ
6	Последовательный вывод
8	Последовательный ввод
10	Функция LOC
12	Функция LOF
14	Функция EOF
16	Функция FPOS
18	Символ поддержки

Табл.21.1. Запросы к устройству

TEXT - это указатель на текст программы на языке BASIC, если эта BASIC-программа в картридже должна автоматически запускаться при перезагрузке. В противном случае там хранится ноль. Размер программы должен быть не более 16 К, по адресам с 8000h по BFFFh.

Интерпретатор языка BASIC проверяет содержимое поля TEXT заголовка картриджа после инициализации (INIT) и после того как стартует система. Если там не ноль, то по указателю TEXT запускается BASIC-программа. Она должна храниться в промежуточном коде и ее начало обозначается кодом ноль.

п.2. Создание CALL-подпрограмм пользователем

Приведем пример программы создания следующих подпрограмм пользователя, вызываемых из MSX-BASIC оператором CALL:

CALL RUSON - включение русских букв;
CALL RUSOFF - выключение русских букв;
CALL CAPSON - включение прописных букв;
CALL CAPSOFF - выключение прописных букв.

Эти операторы формируются в псевдо-ROM по адресу 4000h. Для правильной трансляции ссылок используются директивы .PHASE и .DEPHASE, которые описаны ниже.

Перед записью значений в ячейку управления вторичными слотами (FFFFh) они должны инвертироваться. Это нужно помнить и при запоминании текущего состояния слотов.

Хотя для трансляции мы воспользовались ассемблером M80 и сборщиком L80, полученный файл реально имеет тип OBJ. Это обеспечивают первые 7 байт текста программы.

```
       MSX.M-80 1.00       01-Apr-85
                  .Z80
9000 Load         EQU   9000h        ; адрес загрузки
4000 CallROM      EQU   4000h        ; адрес переписывания
FD02 RAMCF        EQU   0FD02h       ; страница 1 слот 3-2
FD89 IdCall       EQU   0FD89h       ; сюда BASIC записывает
                                     ; имя оператора CALL
0F1F RUSSWCH      EQU   0F1Fh        ; вкл/выкл. RUS
FCAB CAPST        EQU   0FCABh       ; статус CAPS
FCAC KANAST       EQU   0FCACh       ; статус RUS
0004 CallNmb      EQU   4            ; количество наших CALL
0000'             ASEG
0000  FE          DB    0FEh         ; Obj-файл
0001  9000        DW    Load         ; адрес загрузки
0003  90F2        DW    Load+Length  ; конечный адрес
0005  9000        DW    Start        ; стартовый адрес
; =======================================
                  .PHASE  Load
9000  F3  Start: DI
; ===  Установка вторичного слота
9001  3A FFFF    LD    A,(0FFFFh)   ; текущее полож. слотов
9004  2F         CPL                ; инверсия
9005  F5         PUSH  AF           ; запись в стек
9006  CB DF      SET   3,A          ; страница 1, втор.сл. 2
9008  CB 97      RES   2,A
900A  32 FFFF    LD    (0FFFFh),A
; ===  Установка первичного слота
900D  DB A8       IN    A,(0A8h)     ; первичный слот
900F  F5          PUSH  AF
9010  F6 0C       OR    00001100b    ; стр. 1, слот 3
9012  D3 A8       OUT   (0A8h),A
; === Заполняем псевдо-ROM
9014  21 902E     LD    HL,PrgEnd                ; откуда
9017  11 4000     LD    DE,CallROM               ; куда
901A  01 00C4     LD    BC,Length-(PrgEnd-Load) ; сколько
901D  ED B0       LDIR                           ; пересылка
; === Восстановление конфигурации BASIC
901F  F1          POP   AF
9020  D3 A8       OUT   (0A8h),A
9022  F1          POP   AF
9023  32 FFFF     LD    (0FFFFh),A
9026  FB          EI
; === Заполнение буфера SLTATR - FD02h, т.е.
; === Разрешение CALL для слота 3-2, первая страница (4000h)
9027  3E 20       LD    A,32
9029  32 FD02     LD    (RAMCF),A
902C  3F          CCF
902D  C9          RET              ; возврат в BASIC
902E      PrgEnd EQU   $
                  .DEPHASE
; === Заголовок псевдо-ROM
                  .PHASE CallROM
4000  41 42       DB     'AB'       ; ID картриджа
4002  0000         DW     0          ; адрес инициализации ROM
```

```
4004  4011       DW     CallBeg   ; адрес обработки CALL
4006  0000       DW     0         ; адрес обработки нестанд. I/O
4008  0000       DW     0         ; адрес текста BASIC в ROM
400A             DS     7,0       ; резерв
; === Начало обработки оператора CALL
4011      CallBeg:
4011  37         SCF              ; флаг "Syntax error"
4012  E5         PUSH HL          ; сохраним HL
4013  06 04      LD   B,CallNmb   ; цикл сравнений
4015  21 4053    LD   HL,IdBlock
4018      NewComp:
4018  E5         PUSH HL
4019  CD 4028    CALL CompBlock     ; сравниваем имена
401C  E1         POP  HL
401D  30 1C      JR   NC,AddrBlock  ; если нашли имя, переход
401F  11 0010    LD   DE,010h       ; на следующее имя, +16 байт
4022  19         ADD  HL,DE         ; увеличиваем адрес
4023  10 F3      DJNZ NewComp       ; повторяем поиск
4025  E1         POP  HL            ; возврат, имени CALL нет
4026  37         SCF                ; "Syntax error"
4027  C9         RET
; === Сравнение имен
4028      CompBlock:
4028  11 FD89     LD   DE,IdCall ; адрес имени CALL
402B  1A  NextS:  LD   A,(DE)
402C  A7          AND  A         ; ноль ?
402D  28 07       JR   Z,EndNm
402F  BE          CP   (HL)      ; сравнить с псевдоROM
4030  37          SCF
4031  C0          RET  NZ        ; выход, не равны
4032  23          INC  HL        ; сравнить следующие символы
4033  13          INC  DE
4034  18 F5       JR   NextS
4036  BE  EndNm:  CP   (HL)      ; тоже ноль ?
4037  37          SCF
4038  C0          RET  NZ        ; выход, если длиннее
4039  3F          CCF            ; имена совпали !
403A  C9          RET
; === Выбираем адрес нашего CALL и переходим
403B      AddrBlock:
403B  3E 04      LD   A,CallNmb
403D  90         SUB  B           ; номер имени CALL
403E  21 404B    LD   HL,AddrCall
4041  87         ADD  A,A         ; смещение в табл.адресов
4042  16 00      LD   D,0
```

```
4044  5F         LD   E,A
4045  19         ADD  HL,DE       ; адрес в таблице - в HL
4046  5E         LD   E,(HL)      ; адрес подпр. CALL - в DE
4047  23         INC  HL
4048  56         LD   D,(HL)
4049  EB         EX   DE,HL       ; адрес - в HL
404A  E9         JP   (HL)        ; переход на наш CALL !
; === Таблица адресов подпрограмм CALL
404B     AddrCall:
404B  4093        DW   RUSON
404D  409D        DW   RUSOFF
404F  40A8        DW   CAPSON
4051  40B7        DW   CAPSOFF
; === Таблица имен операторов CALL, по 16 байт на имя
4053      IdBlock:
4053  52 55 53 4F DEFM  'RUSON'
4057  4E
4058              DEFS  11,0
4063  52 55 53 4F DEFM  'RUSOFF'
4067  46 46
4069              DEFS  10,0
4073  43 41 50 53 DEFM  'CAPSON'
4077  4F 4E
4079              DEFS  10,0
4083  43 41 50 53 DEFM  'CAPSOFF'
4087  4F 46 46
408A              DEFS  9,0
; === Включить RUS
4093  AF   RUSON: XOR   A
4094  32 FCAC     LD    (KANAST),A
4097  F7          RST   30h
4098  00          DEFB  0
4099  0F1F        DEFW  RUSSWCH
409B  E1          POP   HL
409C  C9          RET
; === Выключить RUS
         RUSOFF:
409D  3E FF       LD    A,0FFh
409F  32 FCAC     LD    (KANAST),A
40A2  F7          RST   30h
40A3  00          DEFB  0
40A4  0F1F        DEFW  RUSSWCH
40A6  E1          POP   HL
40A7  C9          RET
; === Включить CAPS
         CAPSON:
40A8  3E FF       LD    A,0FFh
40AA  32 FCAB     LD    (CAPST),A
40AD  F3          DI
40AE  DB AA       IN    A,(0AAh)
40B0  E6 BF       AND   0BFh
40B2  D3 AA       OUT   (0AAh),A
40B4  FB          EI
40B5  E1          POP   HL
```

```
40B6  C9          RET
; === Выключить CAPS
         CAPSOFF:
40B7  AF          XOR A
40B8  32 FCAB     LD  (CAPST),A
40BB  F3          DI
40BC  DB AA       IN  A,(0AAh)
40BE  F6 40       OR  40h
40C0  D3 AA       OUT (0AAh),A
40C2  FB          EI
40C3  E1          POP HL
40C4  C9          RET
                  .DEPHASE
00F2      Length  EQU  $-1-7
                  END
```

22. Работа с файлами

При работе с внешними устройствами в традиционном программировании используются два понятия: набор данных и файл.

Набором данных называют логически связанную совокупность информации, размещаемую на внешних запоминающих устройствах и устройствах ввода/вывода. Таким образом, набор данных имеет физический смысл - информация, хранимая на ленте, диске, бумаге и т.п.

Файл - это абстракция (структура, описание) набора данных в программе на некотором языке программирования. Программист описывает файл и выполняет операции ввода/вывода над файлом, возможно, не зная точно, какой конкретно набор данных будет сопоставлен файлу. Связь файла с набором данных обычно осуществляет оператор открытия файла.

В последнее время понятие "файл" часто используется вместо понятия "набор данных". Файл становится и логическим, и физическим понятием.

С файлами (наборами данных) можно работать на двух уровнях - "низком" и "высоком". В первой главе была описана организация хранения информации на диске. Хорошо разобравшись в структуре директория, DPB, FAT, FCB, можно написать программу, которая ищет файл в директории, затем в FAT, читает или пишет информацию в соответствующие сектора диска и затем обновляет директорий и FAT. Это и есть "низкий" уровень, требующий очень кропотливой и аккуратной работы.

На "высоком" уровне программист берет на себя минимум забот - задает FCB и буфер ввода/вывода, а всю остальную работу выполняют системные функции BDOS MSX-DOS. Такая работа более надежна, но предоставляет меньше возможностей.

п.1. Абсолютное чтение/запись

Под абсолютным чтением/записью здесь понимается чтение/запись логических секторов диска, в том числе BOOT-сектора, секторов директория, таблиц FAT, секторов данных.

Для выполнения таких действий можно использовать системные функции BDOS 1Ah (установка адреса буфера), 1Bh (получение информации о драйвере), 2Fh (абсолютное чтение секторов), 30h (абсолютная запись секторов диска) и другие.

В качестве примера приведем программу восстановления директория, если были случайно уничтожены несколько файлов. Как Вы помните, при этом в первом байте соответствующей записи директория появляется код E5h.

Программа находит такие записи в директории и ждет ввода программистом первой литеры стертого файла. В конце работы восстановленный директорий записывается назад на диск. Обратите внимение, что программа не востанавливает FAT, записи которого при уничтожении файла обнуляются.

```
      MSX.M-80 1.00   01-Apr-85       PAGE   1
                      .Z80
0005          BDOS      EQU 5
0001          CONS_INP  EQU 1
0002          CONS_OUT  EQU 2
001A          SET_DMA   EQU 1Ah
001B          GET_ALLOC EQU 1Bh
002F          ABS_READ  EQU 2Fh
0030          ABS_WRT   EQU 30h
     ; === Установка адреса буфера для ввода директория
0000' 11 009B'          LD   DE,Dir
0003' 0E 1A             LD   C,SET_DMA
0005' CD 0005           CALL BDOS
     ; === Информация о драйвере
0008' 1E 00             LD   E,0          ; текущий дисковод
000A' 0E 1B             LD   C,GET_ALLOC
000C' CD 0005           CALL BDOS
000F' FE FF             CP   0FFh         ; ошибка ?
0011' C8                RET  Z            ; тогда - выход
     ; ===  Берем информацию о диске
0012' DD 7E 0B          LD   A,(IX+11)    ; макс.кол-во файлов
0015' 32 009A'          LD   (MaxF),A     ;  в директории
0018' DD 56 12          LD   D,(IX+18)    ; номер первого сектора
001B' DD 5E 11          LD   E,(IX+17)    ;  директория
001E' D5                PUSH DE           ; запоминаем его в стеке
     ; === Читаем директорий
001F' 26 07             LD   H,7          ; кол-во секторов
0021' 2E 00             LD   L,0          ; текущий драйвер
0023' 0E 2F             LD   C,ABS_READ
0025' CD 0005           CALL BDOS
     ; === Ищем уничтоженные файлы
0028' 21 009B'          LD   HL,Dir       ; нач.адр. буфера для дир.
002B' 16 00             LD   D,0
002D' 3A 009A'          LD   A,(MaxF)     ; для цикла по макс.кол-ву
0030' 5F                LD   E,A          ; файлов в директории
```

```
0031' D5              PUSH DE
0032' E5              PUSH HL           ; начальный адрес записи
0033' 7E       Again: LD   A,(HL)       ; берем первый байт записи
0034' B7              OR   A            ; если ноль - выход
0035' 28 56           JR   Z,Finish     ;   выход
0037' FE E5           CP   0E5h         ; уничтожен ?
0039' 20 44           JR   NZ,Next      ; если нет - то следующий
     ; ===  Печатаем строку - звездочка, имя файла
003B' 1E 0A           LD   E,0Ah        ; вниз на след. строку
003D' 0E 02           LD   C,CONS_OUT
003F' CD 0005         CALL BDOS
0042' 1E 0D           LD   E,0Dh        ; в начало строки
0044' 0E 02           LD   C,CONS_OUT
0046' CD 0005         CALL BDOS
0049' 1E 2A           LD   E,'*'        ; выводим звездочку
004B' 0E 02           LD   C,CONS_OUT
004D' CD 0005         CALL BDOS
     ; ===  Печатаем остаток имени файла (без первой буквы)
0050' E1              POP  HL           ; восстановили HL
0051' E5              PUSH HL
0052' 06 0A           LD   B,10         ; выводим 10 символов
0054' C5   NextCh:    PUSH BC           ; имени файла,
0055' 23              INC  HL           ; сохраняя нужные
0056' E5              PUSH HL           ; регистры в стеке
0057' 5E              LD   E,(HL)
0058' 0E 02           LD   C,CONS_OUT
005A' CD 0005         CALL BDOS         ; вывод на экран
005D' E1              POP  HL
005E' C1              POP  BC
005F' 10 F3           DJNZ NextCh       ; след. символ
     ; ===  Возвращаемся назад на экране на 11 символов
0061' 06 0B           LD   B,11
0063' C5       Back:  PUSH BC
0064' 1E 1D           LD   E,1Dh        ; стрелка  "<-"
0066' 0E 02           LD   C,CONS_OUT
0068' CD 0005         CALL BDOS
006B' C1              POP BC
006C' 10 F5           DJNZ Back
     ; ===  Вводим одну букву с отображением на экране
006E' 0E 01           LD   C,CONS_INP
0070' CD 0005         CALL BDOS
0073' FE 0D           CP   13           ; если нажат ВВОД,
0075' 28 08           JR   Z,Next       ;  то на след. файл
0077' FE 20           CP   20h          ; если не знак,
0079' FA 007F'        JP   M,Next       ;  то на след. файл
007C' E1              POP  HL           ; восстановили  HL
007D' E5              PUSH HL
007E' 77              LD   (HL),A       ; восстанавливаем букву
     ; === Переходим к следующей записи директория (файлу)
007F' E1       Next:  POP  HL           ; след. 32 байта директ.
0080' 11 0020         LD   DE,32
0083' 19              ADD  HL,DE
0084' D1              POP  DE           ; количество просм. файлов
0085' 1B              DEC  DE           ; стало меньше на 1 файл
0086' D5              PUSH DE
```

```
0087' E5              PUSH HL
0088' 7A              LD   A,D          ; директорий исчерпан ?
0089' B3              OR   E
008A' C2 0033'        JP   NZ,Again     ; если нет - повторим поиск
     ; ===  Записываем директорий назад на диск
008D' E1      Finish: POP  HL           ; восстанавл. параметры
008E' D1              POP  DE
008F' D1              POP  DE
0090' 26 07           LD   H,7          ; 7 секторов
0092' 2E 00           LD   L,0          ; на текущий диск
0094' 0E 30           LD   C,ABS_WRT    ; записываем директорий
0096' CD 0005         CALL BDOS
0099' C9              RET               ; все !
009A' 00        MaxF: DB   0            ; макс.кол-во файлов дир.
009B'           Dir:  DS   3584,0       ; 7 секторов по 512 байт
                      END
```

п.2. Использование системных функций

Для правильной работы с файлами необходимо различать тип организации файла (набора данных) и метод доступа к файлу (набору данных). По сути, первое определяет набор данных, а второе - файл в том смысле, о котором было ранее сказано.

Набор данных (файл) может быть потокоориентированным или записеориентированным.

Потокоориентированный набор данных (файл) состоит из числовых, символьных, булевских, битовых констант, разделяемых пробелами, запятыми, символами конца строк и т.п.

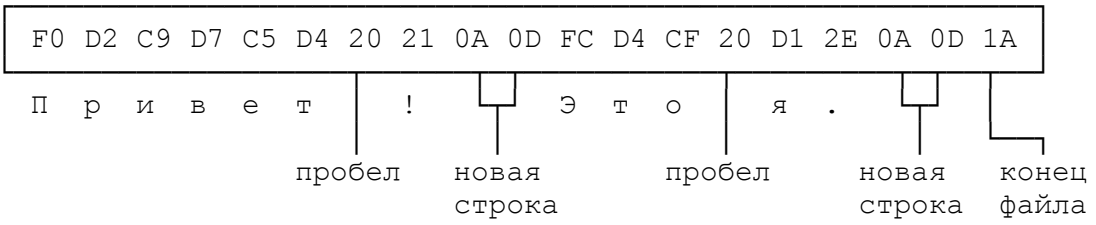
Такие файлы рассчитаны обычно либо на последовательный ввод всего или части файла, либо на последовательный вывод. К ним можно отнести текстовые файлы, BASIC-программы во внутреннем коде, объектные файлы (машинные коды и видеоинформация), промежуточные файлы трансляции, командные файлы (типа COM). Храниться потокоориентированные файлы могут как на магнитных лентах, так и на дисках.

Текстовые файлы состоят из последовательных кодов символов строк, разделяемых кодами 0Ah (вниз на строку) и 0Dh (в начало строки). В конце текстового файла всегда записывается код 1Ah, обозначающий конец файла - EOF.

Рассмотрим, например, как записан на диске следующий текст:

```
Привет !
Это я.
```

В файле он будет представлять собой следующие коды:



Объектные файлы отличаются тем, что имеют заголовок из семи байт: первый байт - код FEh, затем по два байта - загрузочный адрес, конечный адрес, стартовый адрес в INTELовском формате.

BASIC-программы во внутреннем коде имеют в качестве первого байта код FFh.

Записеориентированный набор данных (файл) состоит из записей, обычно фиксированной длины. В таких файлах удобно хранить различные таблицы во внутреннем формате. Каждая строка таблицы соответствует одной записи файла.

Если такие таблицы предназначены для последовательного чтения/записи, то они могут храниться и на лентах, и на дисках, а если иногда необходим ввод/вывод одной строки по ее номеру, то лучше хранить такие файлы на диске и использовать прямые методы доступа.

Метод доступа определяет, каким образом осуществляется доступ к информации, хранящейся в наборе данных (файле). Доступ может быть последовательным, прямым, телекоммуникационным и другим.

Прямой доступ возможен, только когда набор данных размещен на устройстве прямого доступа, например, на магнитном диске. При этом набор данных может иметь практически любую организацию.

Например, текстовый файл можно читать блоками по 1024 байт в любом порядке (в том числе и последовательно), если использовать прямой метод доступа.

Как уже говорилось, для работы с файлами имеются стандартные функции ввода/вывода BDOS. Они вызываются по адресу 0005h (дисковая операционная система) или F37Dh (дисковый BASIC). При этом надо занести в регистр C номер вызываемой функции, а через остальные регистры передать необходимые параметры.

Для того, чтобы открыть файл, необходимо подготовить FCB - блок управления файлом. Его можно создать в любом месте оперативной памяти (при работе в MSX-DOS) и в области 8000h-FFFFh (при работе в BASIC).

Как Вы можете убедиться, номер функции "открыть файл" равен 0Fh. Поэтому для обращения к этой функции в регистр C надо занести 0Fh, в регистровую пару DE - адрес FCB и затем обратиться по указанному адресу BDOS.

Ниже приводится пример программы, которая читает загрузочный, конечный и стартовый адреса файла VALLEY.GM (игра "King's Valley"). Для этого необходимо открыть файл, прочитать из него 7 первых байт и закрыть файл. Перед чтением информации должен быть установлен адрес DMA (буфера обмена с диском). Именно с адреса начала DMA будут записываться все читаемые данные.

```
'files-fcb'       Z80-Assembler   Page:    1
                  ORG   9000h
                  TITLE   'files-fcb'
    ; === Пытаемся открыть файл и заполнить FCB
9000 113090       LD    de,fcb      ; адрес FCB
9003 0E0F         LD    c,0Fh       ; открыть файл
9005 CD7DF3       CALL  0F37Dh      ; вызов BDOS (BASIC)
9008 B7           OR    a           ; смогли открыть ?
9009 37           SCF               ; установить флаг ошибки
900A C0           RET   nz          ; если нет, то возврат
    ; === Устанавливаем адрес буфера для чтения
900B 1100A0       LD    de,0A000H   ; загрузить в DE адрес DMA
900E 0E1A         LD    c,1Ah       ; установить DMA
9010 CD7DF3       CALL  0F37Dh      ; обратиться к DISK-BASIC
    ; === Пытаемся прочитать 7 первых байт файла в DMA
9013 210700       LD    hl,7        ; длина 1 записи
9016 223E90       LD    (fcb+14),hl; записать ее в FCB
9019 113090       LD    de,fcb      ; адрес FCB
901C 210100       LD    hl,1        ; читать 1 запись
901F 0E27         LD    c,27h       ; код функ. чтение блока
9021 CD7DF3       CALL  0F37Dh      ; считать
9024 B7           OR    A           ; смогли считать ?
9025 37           SCF               ; установить флаг ошибки
9026 C0           RET   NZ          ; если не смогли, то возврат
    ; === Закрываем файл
9027 113090       LD    DE,fcb      ; адрес FCB
902A 0E10         LD    C,10h       ; код функ. закрыть файл
902C CD7DF3       CALL  0F37Dh      ; выполнить
902F C9           RET               ; вернуться
    ; === Блок данных
9030 00     fcb:  DB 0              ; номер дисковода (акт.)
9031 56414C4C     DB 'VALLEY  GM ' ; имя (8 байт имя, 3 - тип)
9035 45592020
9039 474D20
903C              DS 28,0           ; все остальные данные - нули
                  END
```

Оттранслируем эту программу в файл с именем "TMP.OBJ" и выполним следующую программу на языке MSX-BASIC:

```
 10 CLEAR 200,&H9000
 20 BLOAD"TMP.OBJ",R
 30 PRINT HEX$(PEEK(&HA000))
 40 PRINT HEX$(PEEK(&HA001)+256*PEEK(&HA002))
 50 PRINT HEX$(PEEK(&HA003)+256*PEEK(&HA004))
 60 PRINT HEX$(PEEK(&HA005)+256*PEEK(&HA006))
 70 END
```

При ее запуске дважды произойдет обращение к диску (для загрузки файла "TMP.OBJ" и для чтения из файла "VALLEY.GM" 7 байт). На экран будет выведено следующее:

```
run
FE
9000
D080
9000
Ok
```

23. Ошибки програмирования и правонарушения, связанные с компьютерами

Программирование на языке ассемблера требует большого внимания и аккуратности. Наиболее простой тип допускаемых ошибок - синтаксические, то есть ошибки в записи команд ассемблера. Такие ошибки локализуются самим ассемблером. Трудность иногда может состоять только в том, чтобы понять, какая именно ошибка допущена.

Однако бывают такие ошибки, которые ассемблер не обнаруживает. Например, если Вы случайно напишете INC H вместо INC HL, то возможно, не скоро найдете причину плохой работы программы.

Другая часто встречающаяся ошибка - загрузка адреса вместо загрузки значения (опускание скобок). Вы должны четко осознавать разницу в смысле команд LD HL,(data) и LD HL,data.

С компьютерами связан целый ряд правонарушений - нарушения прав человека, внедрение для развлечения в информационные системы, кража информации, ее изменение или уничтожение, незаконные финансовые операции.

Некоторые из этих правонарушений совершаются при помощи специальных программ - вредоносного программного обеспечения (ВПО). Основными типами ВПО в настоящее время являются:

- компьютерные троянские кони (trojan horse);
- компьютерные вирусы (virus);
- компьютерные черви (worm).

п.1. Троянские кони

Троянский конь - это компьютерная программа, обычно располагающаяся на диске совершенно открыто, выполняющая некоторые полезные функции и в то же время скрыто наносящая вред.

Например, программа-игра может периодически портить FAT, уничтожать директорий или создавать на диске "сбойные" блоки. Троянский конь такого типа может быть и просто результатом ошибки программиста, разрабатывающего системную программу. Представьте себе, к примеру, программу обслуживания дисков DiskFxr с ошибкой.

Кроме этого троянский конь может быть переносчиком вирусов. После запуска троянского коня им активизируется содержащийся в нем вирус, который дальше действует самостоятельно.

п.2. Компьютерные вирусы

Компьютерный вирус - это программа, обычно скрыто располагающаяся на диске, обладающая возможностью к "саморазмножению" (копированию на другие диски или файлы) и имеющая некоторый вредоносный эффект, который проявляется через определенное время после заражения.

Таким образом, некоторое время вирус "размножается" и только потом начинает производить эффекты. В самых простых случаях это уничтожение директория и FAT. Более коварные вирусы слегка портят информацию или уничтожают ее не сразу, а незаметными порциями. Особенно подлым является незаметное изменение цифр в больших массивах числовых данных.

Три основных типа вирусов, разработаных в настоящее время для системы MSX - это:

- Бутовые вирусы;
- Вирусы MSX-DOS;
- Файловые вирусы.

Для понимания основных принципов их работы необходимо хорошо знать архитектуру компьютера и основы функционирования операционной системы MSX-DOS.

Рассмотрим процесс начальной загрузки системы MSX. Система MSX запускается следующим образом:

1. Сброс питания MSX-компьютера приводит к тому, что в процессе перезагрузки сначала проверяются все слоты, и если в вершине проверяемого слота записаны 2 байта 41H и 42H, слот интерпретируется как относящийся к определенной части ПЗУ. После этого выполняется программа INIT (инициализация), адрес которой установлен в верхней части ПЗУ. В случае использования программы INIT из ПЗУ дискового интерфейса в первую очередь определяется рабочая область для относящегося к нему дисковода.

2. Когда все слоты проверены, машина обращается к адресу FEDAh (H.STKE). Если содержимое этого адреса не равно C9h (т.е. если в этот хук не был записан вызов определенной программы при выполнении процедуры INIT), подготавливается конфигурация DISK-BASIC и управление передается на H.STKE.

3. Если же содержимое H.STKE равно C9h, во всех слотах ищется кассета со входом TEXT. В случае ее нахождения подготавливается конфигурация DISK-BASIC и выполняется BASIC-программа из этой кассеты.

4. Затем содержимое загрузочного сектора диска (логический сектор #0) передается в память на адреса с C000H по C0FFH. При этом, если возникает ошибка неготовности диска или ошибка чтения, или если значение первого байта этого сектора не равно ни EBh, ни E9h, вызывается DISK-BASIC.

5. Вызывается подпрограмма по адресу C01Eh, и происходит сброс флага C. При нормальной работе, поскольку по этому адресу записан код "RET NC", ничего не выполняется, и управление возвращается обратно. Любая записанная здесь на языке ассемблера программа запустится автоматически (первый вход в BOOT-программу).

6. Проверяется емкость ОЗУ (его содержимое при этом не разрушается). Если она менее 64Kb, вызывается DISK-BASIC.

7. Подготавливается конфигурация MSX-DOS и вызывается C01EH, на этот раз с установленным флагом C (второй вызов BOOT-программы). Загружается MSXDOS.SYS с адреса 100H, и на этот же адрес передается управление (т.е. начинает работать MSX-DOS). После этого MSX-DOS переносит себя на более высокий адрес. Если файл MSXDOS.SYS на диске отсутствует, вызывается DISK-BASIC.

8. MSXDOS.SYS загружает COMMAND.COM с диска по адресу 100H и выполняет переход на его начальный адрес. COMMAND.COM тоже переносит себя на более высокий адрес и запускается. Если COMMAND.COM отсутствует, появляется сообщение "INSERT DOS DISKETTE" (вставьте системный диск), и выполнение прерывается до тех пор, пока в дисковод не будет вставлена соответствующая дискета.

9. При первой загрузке MSX-DOS, если существует файл с именем "AUTOEXEC.BAT", он выполняется как обычный пакетный файл. Когда MSX-DOS не запущена и работает DISK-BASIC, и если на диске имеется файл "AUTOEXEC.BAS", то он будет автоматически запущен.

Бутовые вирусы

Бутовыми вирусами называют вирусы, которые размещают себя в BOOT-секторе диска и изменяют программу начальной загрузки таким образом, чтобы получать управление до начала работы "настоящей" BOOT-программы. Получив управление, вирус устанавливает ловушки так, что при записи информации на другой диск на него записывается и зараженный вирусом BOOT-сектор.

Вирусы MSX-DOS

Вирусом могут быть поражены и файлы операционной системы MSX-DOS - COMMAND.COM и MSXDOS.SYS. Вначале программист-хакер корректирует MSX-DOS, внедряя туда вирус, а затем такая "зараженная" MSX-DOS в ходе работы заменяет собой "чистые" версии системы на других дисках.

Файловые вирусы

Разработанные для MSX файловые вирусы, как правило, "живут" в файлах типа COM. В первых байтах зараженного файла обычно вирусом записывается команда перехода на основное тело вируса, которое дописывается к файлу. Таким образом, при запуске вирус первым получает управление, выполняет некоторые действия и затем запускает саму программу.

Ситуации, возможные при заражении вирусами

· По неизвестным причинам увеличился размер, изменилась дата и время создания файла типа COM, изменилась длина командного процессора (COMMAND.COM).

· Увеличилось количество файлов на диске.

· Появилось сообщение "1 file(s) copied".

· На диске появились "плохие" кластеры или уменьшился объем доступной дисковой памяти, хотя файлы не записывались и не удалялись.

· Загрузка или выполнение программы идет дольше, чем обычно.

· Аварийно завершаются ранее нормально функционировавшие программы.

· Зажигается лампочка обращения к дисководу, когда в этом нет очевидной необходимости.

· Машина находится в бесконечном цикле автоматического рестарта.

· Появились необъяснимые зависания или перезагрузки системы.

· Появилось сообщение о защите дискеты от записи при загрузке программ с защищенных от записи дискет ("Write protect error").

Помните, что сбои возможны и из-за неисправности оборудования - сбойные блоки на диске, конфликты адаптеров, несовместимость контроллеров и управляемого ими оборудования,сбои в питающей сети, повреждение на плате, большие колебания температуры, влажности, запыленность, и из-за неквалифицированного или неаккуратного обращения с компьютером.

п.3. Компьютерные черви

Компьютерные черви - это программы, пересылаемые по сетям ЭВМ, захватывающие все ресурсы зараженного компьютера для своей работы и/или крадущие информацию для своего "хозяина". Как правило, черви не содержат разрушающей компоненты.

Поскольку для системы MSX в СССР пока имеются только локальные сети, такой тип вредоносного программного обеспечения имеет чисто познавательный интерес.

п.4. Методы защиты информации

Можно выделить следующие методы защиты информации:

· Технологические и технические методы

· Организационные методы

· Правовые методы

Технологические методы включают в себя методы разработки и применения различных антивирусных программ - программ, которые могут обнаружить, ликвидировать или предупредить заражение.

Однако, к сожалению, большинство имеющихся антивирусных программ могут обнаруживать только изученные к моменту разработки антивирусной программы вирусы. Вновь разработанные вирусы не детектируются.

Кроме этого, широко применяются методы шифрования информации, для защиты от тех, кому не разрешено ей пользоваться.

Нужно также иметь в виду, что электронные устройства излучают радиацию, которая может быть обнаружена, и в случае простых "серийных сигналов" - восстановлена дешевым электронным подслушивающим устройством. Однако защита при помощи физических барьеров считается в сфере образования экономически неоправданной.

Правовые методы включают в себя нормы административного или уголовного права, применяемого к разработчикам вредоносных программ. Такие нормы у нас пока только разрабатываются.

Организационные методы включают в себя учет того, какая информация требуется, как она хранится, как долго, как она помечается, кто ей владеет, как она обрабатывается, кто и как может иметь к ней доступ.

Основные рекомендации и требования по защите информации

· Используйте программное обеспечение, полученное (приобретенное) только от лиц или организаций, которым Вы полностью доверяете. Приобретайте программы законным образом, поскольку защита от копирования может быть выполнена в виде вирусов, а украденные программы могут оказаться троянскими конями или программами, инфицированными вирусами.

· Не работайте с оригиналами дистрибутивных дисков. Делайте с них (если это разрешено поставщиком) копии и работайте с копиями. Лучше иметь не одну, а несколько копий. Под рукой всегда должна быть защищенная от записи дискета с "чистой" MSX-DOS, оболочкой ND, антивирусами и утилитами восстановления FIXER, DBG, VFY, DSKVER и т.п.

· Не пользуйтесь "чужими" дискетами на своем компьютере. Не запускайте с них операционную систему и программы. Не давайте свои дискеты для работы на других компьютерах. Не запускайте программы, назначение которых Вам точно неизвестно.

· В конце работы делайте копии того, что было сделано, на дисках-архивах, с которыми не ведется никакая другая работа, кроме записи на них копий файлов. Лучше всего копировать исходные файлы программ в текстовом виде (в том числе и программы на языке BASIC - в коде ASCII) . Перед копированием выключите ненадолго компьютер и затем загрузите "чистую" MSX-DOS или ND. Перед копированием можно просмотреть файл, но нельзя запускать никакие программы.

· Если Вы хотите поработать на уже включенном кем-то компьютере MSX, обязательно выключите его ненадолго и затем произведите загрузку со своей дискеты. Компьютер мог быть умышленно или случайно заражен вирусом, а его выключение, к счастью, уничтожает все вирусы.

· Если Вы поняли, что диск заражен вирусом, выключите компьютер, загрузите "чистую" систему с эталонного диска MSX-DOS или ND, перепишите файлы с зараженной дискеты, за исключением файлов типа COM, SYS, OBJ, GM (а лучше всего переписывать только текстовые файлы), на чистую дискету, отформатируйте и отверифицируйте зараженную дискету и восстановите на ней файлы.

ГЛАВА 3. МАКРОПРОГРАММИРОВАНИЕ

До сих пор созданием текстов на языке ассемблера (программированием) занимались мы сами, а ассемблер транслировал их в программы на машинном языке. Однако большинство ассемблеров могут кроме этого по определенным правилам сами генерировать команды на языке ассемблера из команд условной генерации и макрокоманд, написанных программистом.

Такие ассемблеры называют макроассемблерами. К ним относится и макроассемблер M80. Процесс трансляции макроассемблером может состоять из двух этапов:

· анализ программы и генерация текста на языке ассемблера;

· генерация программы в машинных кодах.

Таким образом, программирование на макроассемблере занимает промежуточное положение между программированием на языке ассемблера и программированием на языке высокого уровня типа Си, Паскаль, Ада.

Рассмотрим некоторые возможности макроассемблирования.

1. Генерация текста на языке ассемблера

Макроассемблер предоставляет различные возможности по автоматической генерации текста на языке ассемблера по заданным шаблонам.

п.1. Генерация текста несколько раз

Если некоторую группу команд нужно повторить несколько раз (подряд), можно использовать команду повторения REPT. Она имеет следующий вид:

```
REPT  выражение
команды-ассемблера
ENDM
```

Выражение задает количество повторений генерации команд на ассемблере до команды ENDM.

Например, напишем следующий исходный текст:

```
.Z80
LD    A,B
REPT  4
RLCA
ADD   a,3
ENDM
LD    B,A
AND   0fh
END
```

После трансляции макроассемблером M80 получим следующий листинг:

```
     MSX.M-80  1.00     01-Apr-85          PAGE         1
                                  .Z80
 0000'   78                       LD    A,B
                                  REPT  4
                                  RLCA
                                  ADD   a,3
                                  ENDM
 0001'   07              +        RLCA
 0002'   C6 03           +        ADD   a,3
 0004'   07              +        RLCA
 0005'   C6 03           +        ADD   a,3
 0007'   07              +        RLCA
 0008'   C6 03           +        ADD   a,3
 000A'   07              +        RLCA
 000B'   C6 03           +        ADD   a,3
 000D'   47                       LD    B,A
 000E'   E6 0F                    AND   0fh
                                  END
```

Обратите внимание, что макроассемблер отметил команды, которые он сам сгенерировал, знаком "+".

Кроме команд ассемблера в теле REPT могут стоять и некоторые директивы ассемблера, например, DB.

```
   MSX.M-80   1.00     01-Apr-85        PAGE          1
                                         .Z80
 0000'   3E 07'                          LD    a,data
 0002'   06 08'                          LD    b,data+1
 0004'   0E 09'                          LD    c,data+2
 0006'   C9                              RET
 0007'                           data    EQU   $
                                         REPT  3
                                         DB    1, 2
                                         DB    7
                                         ENDM
 0007'   01 02           +               DB    1, 2
 0009'   07              +               DB    7
 000A'   01 02           +               DB    1, 2
 000C'   07              +               DB    7
 000D'   01 02           +               DB    1, 2
 000F'   07              +               DB    7
                                         END
```

п.2. Генерация текста с параметрами

Иногда есть необходимость сгенерировать схожие в чем-то тексты, отличающиеся только некоторыми деталями. Для этого можно использовать одну из двух команд генерации:

```
IRP  параметр,<список>          IRPC  параметр,строка
команды-ассемблера              команды-ассемблера
ENDM                            ENDM
```

Параметр - это любое допустимое имя языка ассемблера. Ассемблер M80 допускает имена, содержащие знак "$". Их удобно использовать для обозначения параметров.

Команда IRP генерирует команды, каждый раз заменяя параметр в командах очередным значением из списка, а команда IRPC подставляет вместо параметра очередной символ строки.

Например, исходный текст:

```
                        .Z80
                        XOR   a
                        LD    a,(data)
                        RET
           data         EQU   $
                        IRP   $P,<1,2,4,7>
                        DB    $P
                        ENDM
                        END
```

После трансляции M80 получим:

```
MSX.M-80 1.00 01-Apr-85 PAGE 1
                                        .Z80
0000'   AF                              XOR   a
0001'   3A 0005'                        LD    a,(data)
0004'   C9                              RET
0005'                    data           EQU   $
                                        IRP   $P,<1,2,4,7>
                                        DB    $P
                                        ENDM
0005'   01          +                   DB    1
0006'   02          +                   DB    2
0007'   04          +                   DB    4
0008'   07          +                   DB    7
                                        END
```

Пример использования команды IRPC:

```
      MSX.M-80 1.00 01-Apr-85 PAGE 1
                                    .Z80
 0000'   AF                         XOR   a
                                    IRPC  $A,BCDE
                                    OR    $A
                                    CP    $A
                                    ENDM
 0001'   B0     +                   OR    B
 0002'   B8     +                   CP    B
 0003'   B1     +                   OR    C
 0004'   B9     +                   CP    C
 0005'   B2     +                   OR    D
 0006'   BA     +                   CP    D
 0007'   B3     +                   OR    E
 0008'   BB     +                   CP    E
 0009'   C9                         RET
                                    END
```

п.3.Условная генерация

Условная генерация - генерация в зависимости от некоторых условий различающихся или различных последовательностей команд ассемблера. Для условной генерации в системе DUAD и в M80 используются конструкции вида:

```
IF  условие                 IF  условие
команды-ассемблера-1        команды-ассемблера-1
ENDIF                       ELSE
                            команды-ассемблера-2
                            ENDIF
```

Команды-ассемблера-1 генерируются, если условие истинно, команды-ассемблера-2 генерируются, если условие ложно.

Команды условной генерации применяются обычно, когда одна и та же исходная программа должна быть настраиваемой на различные условия эксплуатации. Изменив несколько строк в начале программы и перетранслировав ее, можно получить объектный код, рассчитанный например, на другой тип машины или другую ее конфигурацию. Например, пусть исходный текст имеет следующий вид:

```
        ORG    9000h
MSX     EQU    0
MSX2    EQU    1
PRINTER EQU    1
        EX     DE,HL
IF      MSX
        CALL   4Dh
ELSE
        CALL   177h
ENDIF
        EX     DE,HL
IF      PRINTER
        CALL   0A8h
```

```
        JR      nz,round
        LD      A,65
        CALL    0A5h
round   EQU     $
ENDIF
        RET
        END
```

После трансляции ассемблером DUAD получим следующий листинг:

```
                    Z80-Assembler   Page:    1
                    ORG     9000h
0000 =    MSX       EQU     0
0001 =    MSX2      EQU     1
0001 =    PRINTER   EQU     1
9000 EB             EX      DE,HL
          IF        MSX
                    CALL    4Dh
          ELSE
9001 CD7701         CALL    177h
          ENDIF
9004 EB             EX      DE,HL
          IF        PRINTER
9005 CDA800         CALL    0A8h
9008 2005           JR      NZ,round
900A 3E41           LD      a,65
900C CDA500         CALL    0A5h
900F =    round     EQU     $
          ENDIF
900F C9             RET
                    END
```

Обратите внимание, что код, соответствующий MSX, не генерировался. Ниже приведен пример трансляции ассемблером M80 для других условий. Сгенерировано всего 6 байт.

```
     MSX.M-80  1.00    01-Apr-85      PAGE    1
                                       .Z80
  0001                          MSX     EQU     1
  0000                          MSX2    EQU     0
  0000                          PRINTER EQU     0
  0000'   EB                            EX      DE,HL
                                IF      MSX
  0001'   CD 004D                       CALL    4Dh
                                ELSE
                                        CALL    177h
                                ENDIF
  0004'   EB                            EX      DE,HL
                                IF      PRINTER
                                        CALL    0A8h
                                        JR      NZ,round
```

```
                                LD      A,65
                                CALL    0A5h
                        round   EQU     $
                                ENDIF

  0005'   C9                    RET
                                END

No Fatal error(s)
```

Для команд условной генерации обычно не допускается вложенность одного оператора IF в другой. Если же вложенность макроассемблером допускается, ELSE отвечает ближайшему IF, не имеющему ELSE.

2. Трансляция сегментов программ

При трансляции ассемблер использует текущее значение счетчика адреса памяти. Этим значением является адрес следующего байта, для которого транслятор генерирует код.

Однако адрес может быть как абсолютным, так и заданным относительно данных, кодов или общей памяти. Относительный адрес задает смещение к абсолютному стартовому адресу.

Тип адресации задается директивами ассемблеру - ASEG, CSEG, DSEG, COMMON.

Определение абсолютного сегмента

Директива ASEG задает абсолютный режим адресации. При этом генерируются абсолютные коды, жестко привязанные к одному участку памяти.

После директивы ASEG директива ORG должна использоваться с аргументом 103h или больше, причем она задает абсолютный адрес трансляции.

Определение сегмента относительно кодов

Директива CSEG задает режим трансляции относительно кодов. Относительные адреса в этом случае помечаются в листинге апострофом (') после адреса.

Если после CSEG не использована директива ORG, то значению счетчика адресов присваивается то значение, которое было последним в режиме CSEG (по умолчанию - 0).

Директива ORG в режиме CSEG задает не абсолютный адрес, а смещение к последнему значению адреса в режиме CSEG.

Если требуется в режиме CSEG установить абсолютный адрес, то для сборщика используется ключ /P.

Режим CSEG является стандартным режимом работы ассемблера.

Определение сегмента относительно данных

Для задания этого режима адресации используется директива DSEG. Признаком этого режима трансляции являются двойные кавычки после адреса (").

Как и в режиме CSEG, устанавливается то значение счетчика адреса, которое было последним в режиме DSEG, а директива ORG

задает относительное смещение адреса.

Для установки абсолютного адреса в сборщике используется ключ /D.

Определение блока общей области

Директива COMMON /[имя-блока]/ определяет некоторую общую область данных для всех блоков COMMON, известных редактору связей, и является неисполняемой директивой резервирования памяти.

Признак этого режима трансляции - восклицательный знак (!) после адреса. Как и раньше, директива ORG задает относительный адрес.

Через общие блоки с одним и тем же именем разные подпрограммы могут обмениваться данными и результатами.

Смещение

Иногда требуется временно хранить программу в одном месте для последующего переписывания и выполнения в другом. Для этого используется директива

.PHASE выражение.

Выражение должно иметь абсолютное значение.

Директива .DEPHASE используется для обозначения конца трансляции такого смещенного блока кодов.

Ниже приводится пример программы, использующей некоторые директивы управления адресами. Эта программа работает посредством обработки прерываний от таймера (60 раз в секунду). Напомним, что по этому прерыванию центральный процесор выполняет подпрограмму обработки прерывания, находящуюся по адресу 0038h.

Как и любая другая подпрограмма обработки прерывания, она начинается с сохранения регистров (путем засылки их в стек), затем вызывается ловушка этого прерывания (0FD9Ah), в которой вначале находится команда возврата (RET).

При инициализации наша программа перемещает свой код в область, начиная с адреса 4000h (которая интерпретатором языка BASIC не используется) и через ловушку прерывания устанавливает точку входа.

Суть самой программы заключается в том, что она два раза в секунду печатает системное время в правом верхнем углу экрана (SCREEN 0, WIDTH 80). Мы уже сказали, что используемое прерывание происходит 60 раз в секунду (во всей доступной авторам литературе указывается число 50), т.е. каждый тридцатый вызов этого прерывания указывает на то, что прошло 1/2 секунды.

Наша программа имеет счетчик, который увеличивается при каждом вызове подпрограммы обработки прерывания (поскольку сначала выполняется наша подпрограмма, а затем уже подпрограмма обработки прерывания), и если этот счетчик получает значение 29, то он обнуляется и выводится новое время.

Системное время считывается с микросхемы таймера при помощи стандартных функций BDOS.

Приводимая ниже программа написана в мнемонике INTEL 8080.

```
┌──────────────────────────────────────────────
   MSX.M-80   1.00   01-Apr-85    PAGE 1
;-------------------------------------------
;
;       (c) 1989 by Igor BOCHAROV.
;
;-------------------------------------------
; ИСПОЛЬЗУЕМЫЕ ФУНКЦИИ И ТОЧКИ ВХОДА
;
0006     CallF   EQU     0006h    ; МЕЖСЛОТОВЫЙ ВЫЗОВ
FD9A     H.KeyI  EQU     0FD9Ah   ; ОБРАБОТКА ПРЕРЫВАНИЙ
F37D     BDOS    EQU     0F37Dh   ; ВХОДНАЯ ТОЧКА BDOS
002C     GetTime EQU     2Ch      ; ФУНКЦИЯ ЧТЕНИЯ
                                  ; ВРЕМЕНИ В BDOS
9000     Load    EQU     9000h    ; ЗАГРУЗОЧНЫЙ АДРЕС
4010     Work    EQU     4010h    ; РАБОЧИЙ АДРЕС
;-------------------------------------------
                .8080        ; ИСПОЛЬЗУЕТСЯ МНЕМОНИКА Intel
0000'           ASEG         ; АБСОЛЮТНЫЙ СЕГМЕНТ ПРОГРАММЫ
0000 FE         DEFB    0FEh            ; OBJ - ФАЙЛ
0001 9000       DEFW    Load            ; АДРЕС ЗАГРУЗКИ
0003 9095       DEFW    Load+PrgEnd     ; АДРЕС КОНЦА
0005 9000       DEFW    TimeInit        ; АДРЕС ЗАПУСКА
             ;
             ;-------------------------------------------
             ; Инициализация
                .PHASE  Load      ; АДРЕС ТРАНСЛЯЦИИ ЗАГРУЗЧИКА
9000      TimeInit:
9000 F3         DI
9001 DB A8      IN      0A8h      ; ТЕКУЩЕЕ ПОЛОЖЕНИЕ СЛОТОВ
9003 F5         PUSH    psw
9004 3E AA      MVI     a,0AAh    ; УСТАНАВЛИВАЕМ, КАК НАМ НАДО
9006 32 FFFF    STA     0FFFFh
9009 3E FC      MVI     a,0FCh
900B D3 A8      OUT     0A8h
900D 21 9035    LXI     h,ForInit  ; АДРЕС БЛОКА ДЛЯ ЛОВУШКИ
9010 11 FD9A    LXI     d,H.KeyI   ; ЛОВУШКА ПРЕРЫВАНИЯ
9013 01 0005    LXI     b,5        ; ДЛИНА БЛОКА ДЛЯ ЛОВУШКИ
9016 CD 902A    CALL    Ldir       ; INTEL 8080 НЕ ИМЕЕТ LDIR
9019 21 903A    LXI     h,PrgBeg   ; ПЕРЕПИСЫВАЕМ ПОДПРОГРАММУ
901C 11 4010    LXI     d,Work
901F 01 005B    LXI     b,PrgEnd-(PrgBeg-Load)
9022 CD 902A    CALL    Ldir
9025 F1         POP     psw        ; ВОССТАНОВЛИВАЕМ СОСТОЯНИЕ
9026 D3 A8      OUT     0A8h
9028 FB         EI
9029 C9         RET
902A 7E    Ldir: MOV    a,m        ; ПЕРЕПИСЫВАНИЕ БЛОКА
902B 12         STAX    d          ; в ловушку
902C 23         INX     h
```

```
902D 13          INX     d
902E 0B          DCX     b
902F 78          MOV     A,B
9030 B1          ORA     c
9031 C2 902A     JNZ     ldir
9034 C9          RET
9035      ForInit:
9035 F7          RST     CallF
9036 8B          DEFB    8Bh         ; СЛОТ RAM
9037 4010        DEFW    TimeH       ; АДРЕС НАШЕЙ ПОДПРОГРАММЫ
9039 C9          RET                 ; ВОЗВРАТ ИЗ ЛОВУШКИ
903A    PrgBeg   EQU   $
                 .DEPHASE
;-----------------------------------------
; ПЕЧАТЬ АСТРОНОМИЧЕСКОГО ВРЕМЕНИ
; ВИСИТ НА ПРЕРЫВАНИИ im2 (rst 38)
;-----------------------------------------
; СОБСТВЕННО САМА ПРОГРАММА
;
                 .PHASE  Work    ; ОСНОВНАЯ ПРОГРАММА
4010 21 405F TimeH:LXI   h,MyJiffy       ; TIME
4013 34          INR     m
4014 7E          MOV     a,m     ; 30 ТИКОВ = 1/2 СЕК
4015 D6 1D       SUI     29
4017 C0          RNZ             ; ВОЗВРАТ, НЕ ПРОШЛО 1/2 СЕК.
4018 36 07       MVI     m,a     ; ОБНУЛЯЕМ СЧЕТЧИК СЕКУНД
401A 21 4062     LXI     h,Timer+2       ; ":" ИЛИ " "
401D 3E 1A       MVI     a,':' XOR ' ' ; ИНВЕРТИРУЕМ ':' НА ' '
401F AE          XRA     M          ; ' '=>':', ':'=>' '
4020 77          MOV     M,A        ; СОХРАНЯЕМ НОВОЕ СОСТОЯНИЕ
4021 0E 2C       MVI     c,GetTime  ; ФУНКЦИЯ BDOS:
4023 CD F37D     CALL    BDOS       ; СЧИТАТЬ ВРЕМЯ!
4026 11 4060     LXI     d,Timer    ; АДРЕС СТРОКИ-ШАБЛОНА
4029 7C          MOV     A,H        ; ЧАСЫ
402A CD 404D     CALL    DaaDig     ; ПРЕОБРАЗУЕМ В ДЕСЯТ.ВИД
402D 13          INX     d          ; ПРОПУСТИТЬ ДВОЕТОЧИЕ
402E 7D          MOV     A,L        ; МИНУТЫ
402F CD 404D     CALL    DaaDig     ; ТОЖЕ ПРЕОБРАЗУЕМ
; === С ТЕКУЩИМ ВРЕМЕНЕМ
4032 F3          DI
4033 3E 4B       MVI     a,75     ; КУДА ВЫВОДИТЬ (МЛАДШИЙ БАЙТ)
4035 D3 99       OUT     99h
4037 3E 00       MVI     A,00     ; СТАРШИЙ БАЙТ
4039 F6 40       ORI     40h      ; ФЛАГ: ЗАПИСЬ ВО VRAM
403B D3 99       OUT     99h
403D E3          XTHL             ; ЗАДЕРЖКА
403E E3          XTHL
403F 21 4060     LXI     h,Timer  ; АДРЕС СТРОКИ-ШАБЛОНА
4042 06 05       MVI     B,5      ; СКОЛЬКО
4044 7E          MOV     A,M      ; ВЫВЕСТИ!
4045 D3 98       OUT     98h
4047 23          INX     H
4048 05          DCR     B
4049 C2 4044     JNZ     $-5
404C C9          RET              ; ВОЗВРАТ
```

```
;-------------------------------------------
; ПОЛУЧЕНИЕ ДЕСЯТИЧНОГО ЧИСЛА В КОДЕ ASCII
; ВХОД:   [a] - ЧИСЛО, [de] - КУДА ЕГО ЗАПИСАТЬ
; ВЫХОД:  (de) = ДЕСЯТИЧНОЕ ЧИСЛО,
;          [de] = [de] + 3
;----------------------
404D 06 2F DaaDig:MVI    b,'0'-1 ; ПОЛУЧЕНИЕ ДЕСЯТИЧ. ЧИСЛА
404F 04          INR     b            ; ПО АДРЕСУ В [de] В ASCII,
4050 D6 0A       SUI     10           ; ЧИСЛО В [a]
4052 D2 404F     JNC     DaaDig+2
4055 C6 3A       ADI     '9'+1
4057 4F          MOV     C,A          ; ЧИСЛО В [b] И [c]
4058 78          MOV     A,B          ; СТАРШИЙ РАЗРЯД ЧИСЛА
4059 12          STAX    D
405A 13          INX     D
405B 79          MOV     A,C          ; МЛАДШИЙ РАЗРЯД
405C 12          STAX    D
405D 13          INX     D
405E C9          RET
;-------------------------------------------
; РАБОЧАЯ ОБЛАСТЬ
;
405F 00   MyJiffy:DEFB   0          ; СЧЕТЧИК СЕКУНД
4060 3F3F3A3F Timer:  DEFB  "??:??" ; ШАБЛОН ДЛЯ ВЫВОДА
4064 3F                             ;     ВРЕМЕНИ
;-------------------------------------------
                 .DEPHASE
0095     PrgEnd  EQU   $-1  ; ОТНОСИТЕЛЬНЫЙ АДРЕС КОНЦА
                 end
```

3. Макрокоманды

Еще одна возможность макрогенерации - использование макрокоманд. В этом случае группе команд дается имя, и каждое использование этого имени в тексте будет означать подстановку соответствующей группы команд в текст. Описание макрокоманды называют макроопределением. Оно выглядит следующим образом:

```
имя     MACRO   параметры
        команды-ассемблера
        ENDM
```

Список параметров может отсутствовать. Тогда использование имени макрокоманды в тексте будем обозначать просто подстановку вместо него соответствующей группы команд.

Например, имеется исходный текст:

```
                .Z80
   SHIFT        MACRO
                LD      A,B
                RLCA
                RLCA
                RLCA
                LD      B,A
```

```
                        ENDM
                        Ld      b,76
                        SHIFT

                        LD      b,34h
                        SHIFT
                        RET
                        END
```

После его трансляции M80 получим:

```
      MSX.M-80   1.00   01-Apr-85   PAGE 1
                                .Z80
                   SHIFT        MACRO
                                LD      A,B
                                RLCA
                                RLCA
                                RLCA
                                LD      B,A
                                ENDM
0000' 06 4C                     LD      b,76
                                SHIFT
0002' 78           +            LD      A,B
0003' 07           +            RLCA
0004' 07           +            RLCA
0005' 07           +            RLCA
0006' 47           +            LD      B,A
0007' 06 34                     LD      b,34h
                                SHIFT
0009' 78           +            LD      A,B
000A' 07           +            RLCA
000B' 07           +            RLCA
000C' 07           +            RLCA
000D' 47           +            LD      B,A
000E' C9                        RET
                                END
```

Если в заголовке макрокоманды были указаны параметры, то вместо них в тексте будут подставлены те значения, которые были использованы при вызове макрокоманды. Например,

```
   'bcd-hex convrt' MSX.M-80 1.00 01-Apr-85 PAGE  1
                          .Z80
                          TITLE 'bcd-hex convrt'
                ; === conversion of BCD-nmb to binary
A000            BCDARG    EQU     0A000h
A001            HEXRES    EQU     0A001h
                MPLY10    MACRO   $reg,$wreg
                          LD      $wreg,$reg  ; копируем
                          SLA     $reg        ; умнож. на 8
                          SLA     $reg
                          SLA     $reg
                          ADD     $reg,$wreg  ; прибавить 2
                          ADD     $reg,$wreg  ; раза
                          ENDM
                ; === берем аргумент
0000' 21 A000             LD    HL,BCDARG ; arg. address
0003' 46                  LD    B,(HL)    ; arg. in  B
0004' AF                  XOR   A         ; clear A
0005' ED 6F               rld
                ; === умножаем десятки на 10
                          MPLY10  A,C     ; * 10
0007' 4F    +             LD      C,A     ; копируем
0008' CB 27 +             SLA     A       ; умнож. на 8
000A' CB 27 +             SLA     A
000C' CB 27 +             SLA     A
000E' 81    +             ADD     A,C     ; прибавить 2
000F' 81    +             ADD     A,C     ; раза
0010' 4F                  LD      C,A
                 ; === прибавляем единицы
0011' 78                  LD      A,B     ; restore argument
0012' E6 0F               AND     0Fh     ; mask
0014' 81                  ADD     A,C     ; add & get result
                 ; === возврат
0015' 32 A001             LD   (HEXRES),A ;
0018' C9                  RET
                          END
```

Локальные метки макрокоманды

Возможны случаи, когда в теле макрокоманды нужно использовать метки. Использовать обычную метку нельзя, потому что при втором вызове макрокоманды появится ошибка "повторно определенная метка". Макроассемблер позволяет обойти это ограничение при помощи использования локальных меток макрокоманды. Вместо таких меток макроассемблер подставляет свои метки особого вида в диапазоне ..0000 Ў ..FFFF.

Рассмотрим пример с использованием локальных макрометок.

```
      MSX.M-80   1.00   01-Apr-85   PAGE 1
                                    .Z80
00A5                         PUTchr  EQU     0A5h
                             PUT     MACRO   $SY
                             LOCAL   $D
                                     LD      ($D),A
                                     LD      A,$SY
                                     CALL    PUTchr
                                     JP      $D+1
                             $D:     DS      1,0
                                     ENDM
                                     PUT     66
0000' 32 000B'          +            LD      (..0000),A
0003' 3E 42             +            LD      A,66
0005' CD 00A5           +            CALL    PUTchr
0008' C3 000C'          +            JP      ..0000+1
000B'                   +    ..0000: DS      1,0
                                     PUT     42
000C' 32 0017'          +            LD      (..0001),A
000F' 3E 2A             +            LD      A,42
0011' CD 00A5           +            CALL    PUTchr
0014' C3 0018'          +            JP     ..0001+1
0017'                   +    ..0001: DS     1,0
0018' C9                             RET
                                     END
```

Дополнительные возможности макрокоманд

Во время компиляции можно использовать так называемые переменные времени компиляции. Для присваивания значения такой переменной используется директива SET:

имя SET выражение.

Для управления печатью листинга макроассемблера можно использовать директивы:

- LALL - выводит полный текст макрорасширения;
- SALL - только объектный код расширения без текста;
- XALL - выводит те строки, которые генерируют текст.

Операции:

& - связывание метки и параметра, например, ERROR&X;

;; - макрокомментарий;

! - означает, что за ним - литерал. Например, "!;" означает символ точка с запятой.

% - преобразование выражения в число. Например, %X+Y.

ЗАКЛЮЧЕНИЕ

На этом мы заканчиваем изучение команд и директив языков ассемблера и макроассемблера для микропроцессора ZILOG-80 в среде MSX-2. В ограниченном объеме книги не удалось подробно осветить некоторые тонкие вопросы программирования, но авторы надеются, что некоторое представление об архитектуре MSX-2 и управлении устройствами этой системы внимательный читатель все же получил.

Желаем Вам успехов в программировании и надеемся, что эта книга предоставила Вам ответы на многие вопросы, касающиеся системы MSX-2. Авторы будут благодарны за все замечания и предложения по содержанию книги.

ЛИТЕРАТУРА

1. YAMAHA personal computer YIS 503IIR/IIIR. Techn. Summary. YAMAHA Corp. Hamamatsu, Japan. 07.87. 101 p.

2. YAMAHA. Справочное руководство по языку программирования MSX-BASIC для комплектов учебной вычислительной техники на базе персональных компьютеров "Ямаха MSX-2". 03.88. YAMAHA Corp. Japan. 474 с.

3. YAMAHA. Локальная сеть: верс.3.0. Руководство. YAMAHA Corp. Japan. 02.88. 120 с.

4. Бедрековский М.А., Кручинкин Н.С., Подолян В.А. Микропроцессоры. - М., Радио и связь, 1981.

5. Радио. 1982. NN 9 - 12.

6. Радио. 1988. NN 11,12.

7. Радио. 1989. NN 1-4.

8. МикроЭВМ. - М.: Энергоиздат, 1982.

9. Schiller E. Computerwissen fuer alle.- Leipzig: VEB Fachbuchverlag. 1988. 232 S.

10. Рафикузаман М. Микропроцессоры и машинное проектирование микропроцессорных систем. Кн.1. М.: Мир. 1988.

11. MSX-2 Technical Handbook. ASCII Corporation. 400 p.

12. MSX Interface Cartridge for NL-10. Users Manual.- Star Micronics Co., Ltd. Japan. 134 p.

13. Буреев Л.Н. и др. Простейшая микро-ЭВМ: Проектирование. Наладка. Использование.- М.: Энергоатомиздат. 1989.- 216 с.

14. Черемных С.В. и др. От микропроцессоров к персональным ЭВМ.- М.: Радио и связь. 1988.- 288 с.

15. Батурин Ю.М. Компьютерное преступление - что за этим понятием // Интерфейс. 1990. N 1. С.36-37.

16. Николаев А. Осторожно - вирус! // Компьютер Пресс. 1990. N 6, С.3-16.

17. Чижов А.А. Некоторые соображения по поводу компьютерных вирусов // В мире персональных компьютеров. 1988. N 1. С.121-124.

18. Abel H., et al Datensicherungsma╬nehmen beim Einsatz von Arbeitsplatzcomputern //Datenschutz und Datensicher. 1989. N 10. S.498-504.

19. Wiener D.P. When a virus makes Your PC sneeze // US News and World Rept. 1990. Vol.108, N8, p.62.

20. Greenberg R.M. Know thy viral enemy //BYTE, 1989. Vol.14, N 6. p.275-280.

ПРИЛОЖЕНИЕ 1. СИСТЕМА КОМАНД МИКРОПРОЦЕССОРА Z-80

В приложении приводится полный список команд микропроцессора Z-80 фирмы Zilog. Дана мнемоника команд Zilog и соответствующая мнемоника для микропроцессора INTEL 8080 фирмы INTEL, описано действие команд, время выполнения команд.

Принятые сокращения и обозначения

A	- регистр A;
I	- регистр I контроля вектора прерываний;
R	- регистр памяти R;
r	- регистр A,B,C,D,E,H или L;
r1	- то же, что и r;
HL	- регистровая пара HL;
·	- состояние флага не изменяется;
0	- флаг сбрасывается в 0;
1	- флаг устанавливается в 1;
X	- флаг не определен;
?	- состояние флага зависит от результата операции;
V	- если было переполнение - 1, нет - 0
P	- если количество единичных битов результата четно (или ноль) - 1, нечетно - 0
d	- смещение;
n	- 8-ми битная константа;
nn	- 16-ти битная константа или адрес;
IFF	- триггер разрешения прерывания;
dd	- регистр BC,DE,HL,SP;
qq	- регистр BC,DE,HL,AF;
pp	- регистр BC,DE,IX,SP;
rr	- регистр BC,DE,IY,SP;
b	- бит номер 0,1,2,3,4,5,6 или 7;
cc	- условие NZ,Z,NC,C,PO,PE,P или M;
p	- адрес рестарта 0,8h,10h,18h,20h,28h, 30h или 38h;
e	- смещение при относительной адресации;
s/b	- указывает на бит b ячейки s;
CY	- триггер (флаг) переноса C;
()	- косвенная адресация по содержимому операнда;
ЧЦ	- число циклов;
ЧТ	- число тактов;
Дл	- длина команды в байтах.

Группа команд 8-разрядной загрузки

Это наиболее многочисленная группа команд. С их помощью производится обмен данными между внутренними регистрами микропроцессора, а также между внутренними регистрами и ячейками памяти.

Мнемокод	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
LD r,r1	r <= r1	·	·	·	·	·	·	1	1	4	MOV R,R'
LD A,I	A<=I,PV=IFF	·	?	?	?	0	0	2	2	9	—
LD A,R	A <= R	·	·	·	·	·	·	2	2	9	—
LD I,A	I <= A	·	·	·	·	·	·	2	2	9	—
LD R,A	R <= A	·	·	·	·	·	·	2	2	9	—
LD r,n	r <= n	·	·	·	·	·	·	2	2	7	MVI R,N
LD r,(HL)	r <= (HL)	·	·	·	·	·	·	1	2	7	MOV R,M
LD r,(IX+d)	r <= (IX+d)	·	·	·	·	·	·	3	5	19	—
LD r,(IY+d)	r <= (IY+d)	·	·	·	·	·	·	3	5	19	—
LD A,(BC)	A <= (BC)	·	·	·	·	·	·	1	2	7	LDAX B
LD A,(DE)	A <= (DE)	·	·	·	·	·	·	1	2	7	LDAX D
LD A,(nn)	A <= (nn)	·	·	·	·	·	·	3	4	13	LDA NN
LD (HL),r	(HL) <= r	·	·	·	·	·	·	1	2	7	MOV M,R
LD (IX+d),r	(IX+d) <= r	·	·	·	·	·	·	3	5	19	—
LD (IY+d),r	(IY+d) <= r	·	·	·	·	·	·	3	5	19	—
LD (BC),A	(BC) <= A	·	·	·	·	·	·	1	2	7	STAX B
LD (DE),A	(DE) <= A	·	·	·	·	·	·	1	2	7	STAX D
LD (nn),A	(nn) <= A	·	·	·	·	·	·	3	4	13	STA NN
LD (HL),n	(HL) <= n	·	·	·	·	·	·	2	3	10	MVI M,N
LD (IX+d),n	(IX+d)<=n	·	·	·	·	·	·	4	5	19	—
LD (IY+d),n	(IY+d)<=n	·	·	·	·	·	·	4	5	9	—

Группа команд 16-разрядной загрузки

Мнемокод	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
LD dd,nn	dd<=nn	·	·	·	·	·	·	3	3	10	LXI DD,NN
LD IX,nn	IX<=nn	·	·	·	·	·	·	4	4	14	—
LD IY,nn	IY<=nn	·	·	·	·	·	·	4	4	14	—
LD dd,(nn)	dd<=(nn)	·	·	·	·	·	·	4	6	20	—
LD HL,(nn)	HL<=(nn)	·	·	·	·	·	·	3	5	16	LHLD NN
LD IX,(nn)	IX<=(nn)	·	·	·	·	·	·	4	6	20	—
LD IY,(nn)	IY<=(nn)	·	·	·	·	·	·	4	6	20	—
LD (nn),HL	(nn)<=HL	·	·	·	·	·	·	3	5	16	SHLD NN
LD (nn),dd	(nn)<=dd	·	·	·	·	·	·	4	6	20	—
LD (nn),IX	(nn)<=IX	·	·	·	·	·	·	4	6	20	—
LD (nn),IY	(nn)<=IY	·	·	·	·	·	·	4	6	20	—
LD SP,HL	SP<=HL	·	·	·	·	·	·	1	1	6	SPHL
LD SP,IX	SP<=IX	·	·	·	·	·	·	2	2	10	—
LD SP,IY	SP<=IY	·	·	·	·	·	·	2	2	10	—
PUSH qq	SP<= SP-2 (SP) <= qq	·	·	·	·	·	·	1	3	11	PUSH PSW, B,D,H
PUSH IX	SP<= SP-2 (SP) <= IX	·	·	·	·	·	·	2	4	15	—
PUSH IY	SP<= SP-2 (SP) <= IY	·	·	·	·	·	·	2	4	15	—
POP qq	qq<=(SP) SP <= SP+2	· ?	· ?	· ?	· ?	· ?	· ?	1	3	10	POP PSW, B,D,H
POP IX	IX<=(SP) SP <= SP+2	·	·	·	·	·	·	2	4	14	—
POP IY	IY<=(SP) SP <= SP+2	·	·	·	·	·	·	2	4	14	—

Команда POP AF делает содержимое регистра признаков AF равным значению регистра F из стека.

Команды 8-разрядной арифметики

Мнемокод	Символич. описание	Флаги						ДЛ	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
INC r	r <= r+1	·	?	V	?	0	?	1	1	4	INR R
INC (HL)	(HL)<=(HL)+1	·	?	V	?	0	?	1	3	11	INR M
INC (IX+d)	(IX+d) <= (IX+d)+1	·	?	V	?	0	?	3	6	23	—
INC (IY+d)	(IY+d) <= (IY+d)+1	·	?	V	?	0	?	3	6	23	—
DEC r	r <= r-1	·	?	V	?	1	?	1	1	4	DEC R
DEC (HL)	(HL)<=(HL)-1	·	?	V	?	1	?	1	3	11	DEC M
DEC (IX+d)	(IX+d) <= (IX+d)-1	·	?	V	?	1	?	3	6	23	—
DEC (IY+d)	(IY+d) <= (IY+d)-1	·	?	V	?	1	?	3	6	23	—
ADD A,r	A<=A+r	?	?	V	?	0	?	1	1	4	ADD R
ADD A,n	A<=A+n	?	?	V	?	0	?	2	2	7	ADI N
ADD A,(HL)	A<=A+(HL)	?	?	V	?	0	?	1	2	7	ADD M
ADD A,(IX+d)	A<=A+(IX+d)	?	?	V	?	0	?	3	5	19	—
ADD A,(IY+d)	A<=A+(IY+d)	?	?	V	?	0	?	3	5	19	—
ADC A,r	A<=A+r+CY	?	?	V	?	0	?	1	1	4	ADC R
ADC A,n	A<=A+n+CY	?	?	V	?	0	?	2	2	7	ACI N
ADC A,(HL)	A<=A+(HL)+CY	?	?	V	?	0	?	1	2	7	ADC M
ADC A,(IX+d)	A<=A+(IX+d)+CY	?	?	V	?	0	?	3	5	19	—
ADC A,(IY+d)	A<=A+(IY+d)+CY	?	?	V	?	0	?	3	5	19	—
SUB r	A<=A-r	?	?	V	?	1	?	1	1	4	SUB R
SUB n	A<=A-n	?	?	V	?	1	?	2	2	7	SUI N
SUB A,(HL)	A <= A-(HL)	?	?	V	?	1	?	1	2	7	SUB M
SUB A,(IX+d)	A <= A-(IX+d)	?	?	V	?	1	?	3	5	19	—
SUB A,(IY+d)	A <= A-(IY+d)	?	?	V	?	1	?	3	5	19	—
SBC A,r	A<=A-r-CY	?	?	V	?	1	?	1	1	4	SBB R
SBC A,n	A<=A-n-CY	?	?	V	?	1	?	2	2	7	SBI N

SBC A,(HL)	A<=A-(HL)-CY	?	?	V	?	1	?	1	2	7	SBB M
SBC A,(IX+d)	A<=A-(IX+d)-CY	?	?	V	?	1	?	3	5	19	—
SBC A,(IY+d)	A<=A-(IY+d)-CY	?	?	V	?	1	?	3	5	19	—
NEG	A=0-A	?	?	V	?	1	?	2	2	8	—
DAA	Десятичная коррекция	?	?	P	?	·	?	1	1	4	DAA

Группа команд информационного обмена

Эта группа команд позволяет производить обмен данными между регистровыми парами, содержимым стека и регистровой парой; производить смену текущего набора регистров.

Мнемокод	Символическое описание	Флаги						ДЛ	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
EX DE,HL	DE <=> HL	·	·	·	·	·	·	1	1	4	XCHG
EX AF,AF'	AF <=> AF'	?	?	?	?	?	?	1	1	4	—
EXX	BC <=> BC' DE <=> DE' HL <=> HL'	·	·	·	·	·	·	1	1	4	—
EX (SP),HL	H <=> (SP+1) L <=> (SP)	·	·	·	·	·	·	1	5	19	XTHL
EX (SP),IY	IY <=> (SP)	·	·	·	·	·	·	2	6	23	—
EX (SP),IX	IX <=> (SP)	·	·	·	·	·	·	2	6	23	—

Команды 16-разрядной арифметики

Мнемокод	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
INC dd	dd<=dd+1	·	·	·	·	·	·	1	1	6	INX DD
INC IX	IX<=IX+1	·	·	·	·	·	·	2	2	10	—
INC IY	IY<=IY+1	·	·	·	·	·	·	2	2	10	—
DEC dd	dd<=dd-1	·	·	·	·	·	·	1	1	6	DCX DD
DEC IX	IX<=IX-1	·	·	·	·	·	·	2	2	10	—
DEC IY	IY<=IY-1	·	·	·	·	·	·	2	2	10	—
ADD HL,dd	HL<=HL+dd	?	·	·	·	0	X	1	3	11	DAD DD
ADC HL,dd	HL<=HL+dd+CY	?	?	V	?	0	X	2	4	15	—
SBC HL,dd	HL<=HL-dd-CY	?	?	V	?	1	X	2	4	15	—
ADD IX,pp	IX<=IX+pp	?	·	·	·	0	X	2	4	15	—
ADD IY,rr	IY<=IY+rr	?	·	·	·	0	X	2	4	15	—

Логические команды

Мнемокод	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
AND r	A <=A & r	0	?	P	?	0	1	1	1	4	ANA R
AND n	A <=A & n	0	?	P	?	0	1	2	2	7	ANI N
AND (HL)	A <=A & (HL)	0	?	P	?	0	1	1	2	7	ANA M
AND (IX+d)	A<=A &(IX+d)	0	?	P	?	0	1	3	5	19	-
AND (IY+d)	A<=A &(IY+d)	0	?	P	?	0	1	3	5	19	-
OR r	A <=A V r	0	?	P	?	0	0	1	1	4	ORA R
OR n	A <=A V r	0	?	P	?	0	0	2	2	7	ORI N
OR (HL)	A <=A V (HL)	0	?	P	?	0	0	1	2	7	ORA M
OR (IX+d)	A<=A V(IX+d)	0	?	P	?	0	0	3	5	19	-
OR (IY+d)	A<=A V(IY+d)	0	?	P	?	0	0	3	5	19	-
XOR r	A <= A ╫ r	0	?	P	?	0	0	1	1	4	XRA R
XOR n	A <= A ╫ n	0	?	P	?	0	0	2	2	7	XRI N
XOR (HL)	A <= A ╫(HL)	0	?	P	?	0	0	1	2	7	XRA M
XOR (IX+d)	A<=A ╫(IX+d)	0	?	P	?	0	0	3	5	19	-
XOR (IY+d)	A<=A ╫(IY+d)	0	?	P	?	0	0	3	5	19	-
CP r	A ? r	?	?	V	?	1	?	1	1	4	CPM R
CP n	A ? n	?	?	V	?	1	?	2	2	7	CPI N
CP (HL)	A ? (HL)	?	?	V	?	1	?	1	2	7	CPM M
CP (IX+d)	A ? (IX+d)	?	?	V	?	1	?	3	5	19	-
CP (IY+d)	A ? (IY+d)	?	?	V	?	1	?	3	5	19	-
CPL	Инверсия битов аккумул. 0 <=> 1	·	·	·	·	1	1	1	1	4	CMA
CCF	Инвертировать флаг C	?	·	·	·	0	0	1	1	4	CMC
SCF	Установка C	1	·	·	·	0	0	1	1	4	STC

Команды для работы с отдельными разрядами

Мнемокод	Символич. описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
BIT b,r	0=r/b ?	·	?	X	X	0	1	2	2	8	—
BIT b,(HL)	0=(HL)/b ?	·	?	X	X	0	1	2	3	12	—
BIT b,(IX+d)	0=(IX+d)/b?	·	?	X	X	0	1	4	5	20	—
BIT b,(IY+d)	0=(IY+d)/b?	·	?	X	X	0	1	4	5	20	—
SET b,r	r/b<=1	·	·	·	·	·	·	2	2	8	—
SET b,(HL)	(HL)/b<=1	·	·	·	·	·	·	2	4	15	—
SET b,(IX+d)	(IX+d)/b<=1	·	·	·	·	·	·	4	6	23	—
SET b,(IY+d)	(IY+d)/b<=1	·	·	·	·	·	·	4	6	23	—
RES b,r	r/b<=0	·	·	·	·	·	·	2	2	8	—
RES b,(HL)	(HL)/b<=0	·	·	·	·	·	·	2	4	15	—
RES b,(IX+d)	(IX+d)/b<=0	·	·	·	·	·	·	4	6	23	—
RES b,(IY+d)	(IY+d)/b<=0	·	·	·	·	·	·	4	6	23	—

Команды работы с портами ввода/вывода

Микропроцессор INTEL8080 имеет всего одну команду ввода и одну команду вывода. Это соответственно IN и OUT. По команде OUT содержимое аккумулятора записывается в порт, номер которого указывается непосредственно в команде. Команда IN позволяет ввести байт из порта ввода/вывода и занести его в аккумулятор.

К системе команд микропроцессора ZILOG-80 добавлены команды ввода/вывода блока (как пошаговые, так и автоматические), и ввода/вывода в порт, косвенно адресуемый по содержимому регистра C содержимого любого из основных регистров микропроцессора.

Команды INI и IND устанавливают флаг Z, если B=0.

Команды IN ?,(C) и OUT (C),(HL) не имеют обрабатываемой ассемблером мнемоники, но их можно ввести по машинному коду ED 70 и ED 71 соответственно.

Мнемокод	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
IN A,(n)	A <= порт(n)	·	·	·	·	·	·	2	3	11	IN N
IN r,(C)	r <= порт(C)	·	?	P	?	0	?	2	3	12	—
IN ?,(C)	Уст. флагов как у IN r	·	?	P	?	0	?	2	3	12	—
INI	(HL)<=порт(C) B=B-1 HL=HL+1	X	1 ?	X	X	1	X	2	4	16	—
INIR	(HL)<=порт(C) B=B-1 HL=HL+1 Повторить пока B<>0	X	1	X	X	1	X	2 2	5 4	21 16	—
IND	(HL)<=порт(C) B=B-1 HL=HL-1	X	1 ?	X	X	1	X	2	4	16	—
INDR	(HL)<=порт(C) B=B-1 HL=HL-1 Повторить пока B<>0	X	1	X	X	1	X	2 2	5 4	21 16	—
OUT (n),A	порт(n)<= A	·	·	·	·	·	·	2	3	11	OUT N
OUT (C),r	порт(C)<= r	·	·	·	·	·	·	2	3	12	—
OUT (C),(HL)	Уст.флаги	·	·	·	·	·	·	2	3	12	—
OUTI	порт(C)<=(HL) B=B-1 HL=HL+1	X	0 ?	X	X	1	X	2	4	16	—
OTIR	порт(C)<=(HL) B=B-1 HL=HL+1 Повторить пока B<>0	X	1	X	X	1	X	2 2	5 4	21 16	—
OUTD	порт(C)<=(HL) B=B-1 HL=HL-1	X	?	X	X	1	X	2	4	16	—
OTDR	порт(C)<=(HL) B=B-1 HL=HL-1 Повторить пока B<>0	X	1	X	X	1	X	2 2	5 4	21 16	—

Команды перехода

Мнемокод Z80	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Мнемокод Intel 8080
		C	Z	V	S	N	H				
JP nn	PC <=nn	·	·	·	·	·	·	3	3	10	JMP NN
JP cc,nn	Если условие cc истинно PC= nn, иначе продолжить программу	·	·	·	·	·	·	3	3	10	JNZ, JZ, JNC, JC, JPO, JPE, JP, JM NN
JP (HL)	PC <=HL	·	·	·	·	·	·	1	1	4	PCHL
JP (IX)	PC <=IX	·	·	·	·	·	·	2	2	8	-
JP (IY)	PC <=IY	·	·	·	·	·	·	2	2	8	-
JR e	PC<=PC+e	·	·	·	·	·	·	2	3	12	-
JR C,e	Если C=0,продолжить программу, иначе PC<=PC+e	·	·	·	·	·	·	2 2	2 3	7 12	-
JR NC,e	Если C=1,продолжить программу, иначе PC<=PC+e	·	·	·	·	·	·	2 2	2 3	7 12	—
JR Z,e	Если Z=0,продолжить программу, иначе PC<=PC+e	·	·	·	·	·	·	2 2	2 3	7 12	—
JR NZ,e	Если Z=1,продолжить программу, иначе PC<=PC+e	·	·	·	·	·	·	2 2	2 3	7 12	—
DJNZ e	B=B-1 Если B=0 продолжить программу, иначе PC<=PC+e	·	·	·	·	·	·	2 2	2 3	8 13	—

Группа команд сдвига и циклического сдвига

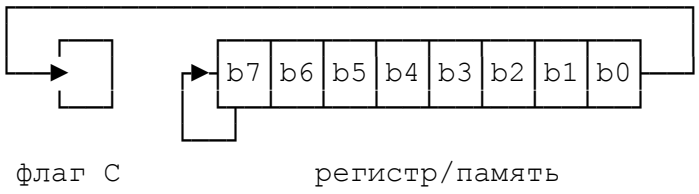
Мнемокод	Флаги						Длина в байт.	Число машин. циклов	Число тактов	Intel 8080
	C	Z	V	S	N	H				
SLA r	?	?	P	?	0	0	2	1	4	-
SLA (HL)	?	?	P	?	0	0	2	4	15	—
SLA (IX+d)	?	?	P	?	0	0	4	6	23	—
SLA (IY+d)	?	?	P	?	0	0	4	6	23	—
SRA r	?	?	P	?	0	0	2	1	4	-
SRA (HL)	?	?	P	?	0	0	2	4	15	—
SRA (IX+d)	?	?	P	?	0	0	4	6	23	—
SRA (IY+d)	?	?	P	?	0	0	4	6	23	—
SRL r	?	?	P	?	0	0	2	1	4	-
SRL (HL)	?	?	P	?	0	0	2	4	15	—
SRL (IX+d)	?	?	P	?	0	0	4	6	23	—
SRL (IY+d)	?	?	P	?	0	0	4	6	23	—
RLA	?	·	·	·	0	0	1	1	4	RAL
RL r	?	?	P	?	0	0	2	1	4	—
RL (HL)	?	?	P	?	0	0	2	4	15	—
RL (IX+d)	?	?	P	?	0	0	4	6	23	—
RL (IY+d)	?	?	P	?	0	0	4	6	23	—
RRA	?	·	·	·	0	0	1	1	4	RAR
RR r	?	?	P	?	0	0	2	2	8	—
RR (HL)	?	?	P	?	0	0	2	4	15	—
RR (IX+d)	?	?	P	?	0	0	4	6	23	—
RR (IY+d)	?	?	P	?	0	0	4	6	23	—
RLCA	?	·	·	·	0	0	1	1	4	RLC
RLC r	?	?	P	?	0	0	2	2	8	—

RLC (HL)	?	?	P	?	0	0	2	4	15	—
RLC (IX+d)	?	?	P	?	0	0	4	6	23	—
RLC (IY+d)	?	?	P	?	0	0	4	6	23	—
RRCA	?	·	·	·	0	0	1	1	4	RRC
RRC r	?	?	P	?	0	0	2	2	8	—
RRC (HL)	?	?	P	?	0	0	2	4	15	—
RRC (IX+d)	?	?	P	?	0	0	4	6	23	—
RRC (IY+d)	?	?	P	?	0	0	4	6	23	—
RLD	·	?	P	?	0	0	2	5	18	—
RRD	·	?	P	?	0	0	2	5	18	—

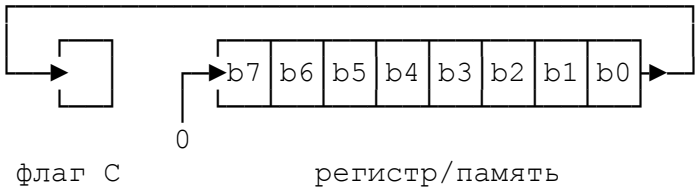
Действие команд сдвига поясняется рисунком:

SLA (сдвиг влево арифметический):

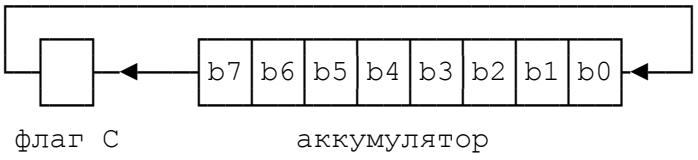
SRA (сдвиг вправо арифметический):



SRL (сдвиг вправо логический):

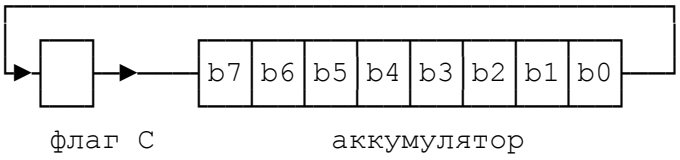


RLA (сдвиг влево через перенос):



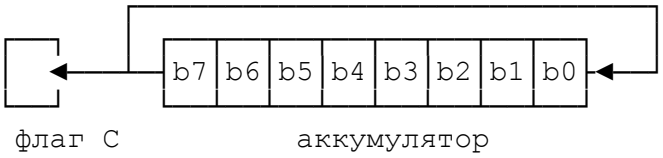
Команда RL выполняется аналогично над регистром или памятью.

RRA (сдвиг вправо через перенос):



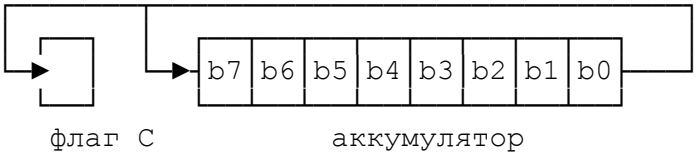
Команда RR выполняется аналогично над регистром или памятью.

RLCA (циклический сдвиг влево):



Команда RLC выполняется аналогично над регистром или памятью.

RRCA (циклический сдвиг вправо):



Команда RRC выполняется аналогично над регистром или памятью.

RLD (обмен полубайтов влево):

аккумулятор память, адресуемая (HL)

RRD (обмен полубайтов вправо):

аккумулятор память, адресуемая (HL)

Группа команд пересылки блока

Мнемокод	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
LDI	(DE)<= (HL) DE=DE+1 HL=HL+1 BC=BC-1	·	·	? 1	·	0	0	2	4	16	—
LDIR	(DE)<= (HL) DE=DE+1 HL=HL+1 BC=BC-1 Повторяется пока BC<>0	·	·	0	·	0	0	2	4	16	—
LDD	(DE)<= (HL) DE=DE-1 HL=HL-1 BC=BC-1	·	·	? 1	·	0	0	2	4	16	—
LDDR	(DE)<= (HL) DE=DE-1 HL=HL-1 BC=BC-1 Повторяется пока BC<>0	·	·	0	·	0	0	2	4	16	—

Группа команд поиска

Мнемокод	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
CPI	A = (HL) ? HL<=HL+1 BC<=BC-1; P/V=0,if BC=0 P/V=1,if <>0	·	?	?	?	1	?	2	4	16	-
CPIR	A = (HL) ? HL<=HL+1 BC<=BC-1 Повторяется пока BC<>0	·	?	?	?	1	?	2	4	16	-
CPD	A = (HL) ? HL=HL-1 BC=BC-1 P/V=0,if BC=0 P/V=1,if <>0	·	?	?	?	1	?	2	4	16	-
CPDR	A = (HL) ? HL=HL-1 BC=BC-1 Повторяется пока BC<>0	·	? 2	? 1	?	1	?	2	4	16	-

Команды обращения к подпрограмме

Мнемокод	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
CALL nn	SP <= SP-2 (SP) <= PC PC = nn	·	·	·	·	·	·	3	5	17	CALL NN
CALL cc,nn	Если условие cc ложно,продолжить программу, иначе так же, как и для CALL nn	·	·	·	·	·	·	3 3	3 5	10 17	CNZ, CZ, CNC, CC, CPO, CP, CPE, CM
RET	PC <= (SP) SP <= SP+2	·	·	·	·	·	·	1	3	10	RET
RET cc	Если условие cc ложно,продолжить программу, иначе так же, как и для RET	·	·	·	·	·	·	1 1	1 3	5 11	RNZ, RZ, RNC, RC, RPO, RP, RPE, RM
RETI	Возврат после прерывания	·	·	·	·	·	·	2	4	14	—
RETN	Возврат после немаскируемого прерывания	·	·	·	·	·	·	2	4	14	—
RST p	SP <= SP-2 (SP)<= PC PC = p	·	·	·	·	·	·	1	3	11	RST N

В системе команд микропроцессора имеется восемь однобайтовых команд RST 0 - RST 7 вызова подпрограмм, расположенных по фиксированным адресам. Ниже приведена таблица соответствия между этими командами и шестнадцатеричными адресами ячеек памяти, куда передается управление при их выполнении.

В мнемонике Z-80 (в отличие от мнемоники INTEL 8080) команда записывается с указанием непосредственного адреса обращения к подпрограмме. Например, RST 7 записывается как RST 38h.

Команда	Адрес начала подпрограммы	Команда	Адрес начала подпрограммы
RST 0	0000	RST 4	0020
RST 1	0008	RST 5	0028
RST 2	0010	RST 6	0030
RST 3	0018	RST 7	0038

Группа команд управления центральным процессором

Команда NOP этой группы не производит никаких операций, однако т.к. она выполняется за определенный отрезок времени, ее можно использовать в программах для задания временных интервалов.

Появление в программе команды HALT ведет к останову выполнения программы. Продолжить выполнение программы можно только подачей сигнала СБРОС или ЗАПРОС ПРЕРЫВАНИЯ на соответствующие входы микропроцессора. В режиме ожидания команда схожа с командой NOP.

Команда IM служит для установки вектора прерываний. Команды DI и EI - для запрещения и разрешения маскируемых прерываний.

Команда IM 0 устанавливает режим прерывания, в котором прерывающее устройство может вставить какую-нибудь команду в шину данных или для выполнения CPU.

Команда IM 1 устанавливает режим прерывания, в котором процессор будет реагировать на прерывание, выполняемое командой RST 38H.

Команда IM 2 устанавливает режим прерывания, в котором разрешается непрямой вызов какой-нибудь ячейки памяти. В этом режиме CPU формирует 16-битный адрес памяти. Восемь верхних битов содержит регистр I контроля вектора прерывания.

Мнемокод	Символическое описание	Флаги						Дл	ЧЦ	ЧТ	Intel 8080
		C	Z	V	S	N	H				
NOP	Нет операции	·	·	·	·	·	·	1	1	4	NOP
HALT	Останов	·	·	·	·	·	·	1	1	4	HLT
DI	IFF=0	·	·	·	·	·	·	1	1	4	DI
EI	IFF=1	·	·	·	·	·	·	1	1	4	EI
IM 0	Установка режима прерываний	·	·	·	·	·	·	2	2	8	—
IM 1		·	·	·	·	·	·	2	2	8	—
IM 2		·	·	·	·	·	·	2	2	8	—

ПРИЛОЖЕНИЕ 2. ЛИСТИНГ ПРОГРАММЫ "ОКЕАН"

Ниже приводится листинг программы, рисующей заставку с надписью "ОКЕАН". Программа оттранслирована ассемблером DUAD.

```
                TITLE  Ocean
                ORG    9000h
9000 CDC793     CALL   ocean@
9003 C9         RET
;---------------------------------------------------------------
0010 = nosprz   EQU   16  ;номера спрайтов для заставки (и +1)
;---------------------------------------------------------------
  ; Подпрограмма записи данных в регистр VDP
  ; [b] - данные, [c] - номер регистра
9004 78   wrrvdp: LD      A,B       ; грузим данные
9005 D399         OUT     (99H),A   ; выкидываем в порт VDP
9007 79           LD      A,C       ; теперь номер регистра VDP
9008 F680         OR      80H       ; устанавливаем 7 бит в 1
900A D399         OUT     (99H),A   ; выкидываем в порт VDP
900C C9           RET               ; возвращаемся
;---------------------------------------------------------------
; данные-"кирпичики" для рисования картинок типа "Океан"
900D FFFFC0C0 block1: DB 0FFh,0FFh,0C0h,0C0h,0C0h,0C0h,080h,0FFh
9011 C0C080FF
9015 91919898         DB 91h, 91H, 98h, 98h, 98h, 98h, 96h, 61h
9019 98989661
901D FFFFFFFF block2: DB 0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
9021 FFFFFFFF
9025 91918181         DB 91h, 91h, 81h, 81h, 81h, 81h, 61h, 61h
9029 81816161
902D FFFEFCFC block3: DB 0FFh,0FEh,0FCh,0FCh,0FCh,0FCh,0FFh,0FFh
9031 FCFCFFFF
9035 91968686       DB 91h, 96h, 86h, 86h, 86h, 86h, 61h, 61h
9039 86866161
;---------------------------------------------------------------
; Подпрограмма подготовки графического режима
; данные для "Шум моря"
903D                DEFS    6
9043 1EB71000       DEFB    30,183,16,0,0,0,90,14
9047 00005A0E
904B CD2894 draw:   CALL    EXchg

; -------- шум моря
904E F7             RST     30h
904F 00             DEFB    0
9050 C000           DEFW    0c0h
9052 214A90         LD      HL,draw-1 ; адрес байта данных
                                      ; для 13 регистра PSG
9055 3E0D           LD      a,13      ; кол-во регистров PSG
9057 5E             LD      e,(HL)    ; загрузить данные
9058 F7             RST     30h
9059 00             DEFB    0
905A 9300           DEFW    93h       ; записать в регистр
905C 2B             DEC     HL        ; следующий байт данных
```

```
905D 3D             DEC     a          ; след.номер регистра PSG
905E F25690         JP      p,draw+1   ; если не -1, повт.
; -------- colOR 4=040
9061 011004         LD      BC,410H    ; записать в 16 регистр VDP
9064 CD0490         CALL    wrrvdp     ; номер регистра палитры 4
9067 3E04           LD      A,4        ; переделать палитру
9069 D39A           OUT     (9AH),A    ; номер 4
906B 3E00           LD      A,0
906D D39A           OUT     (9AH),A
; -------- colOR 15,1,1
906F 210F01         LD      HL,10Fh    ; загр. номера цветов
9072 22EAF3         LD      (0F3EAH),HL; записать в сист.яч.
9075 22E9F3         LD      (0F3E9h),HL
; -------- screen 2,2
9078 21E0F3         LD      HL,0F3E0h  ; адрес регистра 1 VDP
907B CBCE           SET     1,(HL)     ; размер спрайта 16x16
907D CB86           RES     0,(HL)     ; без увеличения
907F F7             RST     30h
9080 00             DEFB    0
9081 7200           DEFW    72H        ; screen 2
; -------- clear the sprites
9083 F7             RST     30h
9084 00             DEFB    0
9085 6900           DEFW    69H
; -------- Установка начальной позиции экрана
9087 011758         LD      BC,5817H   ; начальная строка экрана
                                       ; и регистр 23
908A CD0490         CALL    wrrvdp     ; записать в 23 регистр
908D C5             PUSH    BC         ; записать данные для
                                       ; движения экрана
908E CD2894         CALL    EXchg
; ---------------------
; Подпрограмма рисования изображения из кирпичиков (напр: Океан)
; [HL] - адрес картинки, [DE] - x,y, [BC] - размер Y*X
; ---------------------
9091 ED534994       LD      (const),DE ; сохр. координаты X,Y
9095 79             LD      a,c        ; сохр. размер по X
9096 324B94         LD      (const+2),a
9099 78     l03:    LD      A,B        ; загруз.размер по Y
909A 82             ADD     A,D        ; добавить коорд. Y
909B 57             LD      D,A        ; записать в коорд. Y
909C D5     l04:    PUSH    DE         ; сохранить координаты
909D 7E             LD      a,(HL)     ; загрузить очередной байт
                                       ; (блок) картинки
909E CD2894         CALL    EXchg
90A1 D1             POP     DE         ; считать координаты
90A2 210D90         LD      HL,block1  ; загрузить адрес 1 кирп.
90A5 FE31           CP      '1'        ; надо его рисовать ?
90A7 280E           JR      z,wrvdm    ; если да, рисуем
90A9 211D90         LD      HL,block2  ; загрузить адрес 2 кирп.
90AC FE32           CP      '2'        ; надо его рисовать ?
90AE 2807           JR      z,wrvdm    ; если да, рисуем
90B0 212D90         LD      HL,block3  ; загрузить адрес 2 кирп.
90B3 FE33           CP      '3'        ; надо его рисовать ?
90B5 2010           JR      nz,ewrvdm  ; если нет, переходим
```

```
90B7 D5     wrvdm:  PUSH    DE          ; сохраняем координаты
90B8 0608           LD      b,8         ; грузим длину кирпичика
90BA CD8092         CALL    ldirvm      ; переписываем в VRAM
                                        ; по адресу [DE]
90BD D1             POP     DE          ; считываем координаты
90BE D5             PUSH    DE          ; опять сохраняем
90BF 7A             LD      A,D         ; загр.ст.байт адр.VRAM
90C0 C620           ADD     a,32        ; делаем из адреса
                                        ; шаблонов адрес цветов
90C2 57             LD      D,A         ; записыв. в адрес VRAM
90C3 CD8092         CALL    ldirvm      ; перепис.в цвета данные
90C6 D1             POP     DE          ; считываем адрес VRAM
90C7 3E08   ewrvdm: LD      a,8         ; грузим длину блока
90C9 83             ADD     A,E         ; добавл. к младшему
                                        ; байту адреса VRAM
90CA 5F             LD      E,A
90CB D5             PUSH    DE          ; сохраняем адрес VRAM
90CC CD2894         CALL    EXchg
90CF D1             POP     DE          ; т.е. передаем его
                                        ; осн. группе регистров
90D0 23             INC     HL          ; следующий адрес рисунка
90D1 0D             DEC     c           ; уменьшаем размер по X
90D2 20C8           JR      nz,l04      ; если <> 0, то повт.рисов.
90D4 ED5B4994       LD      DE,(const)  ; загр. исходн.координаты
90D8 3A4B94         LD      a,(const+2); востановить размер по X
90DB 4F             LD      C,A
90DC 10BB           DJNZ    l03         ; если не все, то повт.
90DE C1             POP     BC          ; иначе счит.рег.управления
                                        ; экраном 23
;-------------------------------------------------------------
; Движение экрана [b] - текущее состояние 23 регистра VDP
; [c] - равен 23 (номер регистра)
90DF CD0490 move:   CALL    wrrvdp      ; сдвинуть экран
90E2 113001         LD      DE,130H     ; загр. размер задержки
90E5 CD2294         CALL    time        ; задержка
90E8 04             INC     b           ; след. позиция экрана
90E9 20F4           JR      nz,move     ; если не 0, то повт.
90EB CD0490         CALL    wrrvdp      ; иначе посл.раз сдвин.
  ; ----------------------
  ; рисуем кораблик
  ; ----------------------
90EE 212291  ship:  LD      HL,shipdt   ;адрес блока данных
90F1 11D010         LD      DE,10D0h    ;координаты Y и X
90F4 010628         LD      BC,2806h    ; размеры XxY
90F7 C5             PUSH    BC          ; сохранить размеры
90F8 CD5A92         CALL    draw1       ; перебр. данные в шабл.
90FB C1             POP     BC          ; считать размеры
90FC 11D030         LD      DE,30D0h    ; адрес цветов
90FF 3E71           LD      a,71h       ; цвет изображения и фона
9101 CD6492         CALL    draw2       ; заполнить цвета
  ; ----------------------
  ; пишем спрайты # Nosprz
  ; ----------------------
```

```
9104 3E10   SETspr: LD      a,nosprz   ; узнаем адрес
9106 F7             RST     30h
9107 00             DEFB    0
9108 8400           DEFW    84h        ; шаблона номер
                                       ; Nosprz
910A 111292         LD      DE,sprz    ; грузим адрес данных
                                       ; для шаблонов
910D EB             EX      DE,HL
910E 0640           LD      b,64       ; и длину 2 шаблонов
9110 CD8092         CALL    ldirvm     ; переписываем в VRAM
  ; вывод спрайтов # Nosprz
9113 3E10           LD      a,nosprz   ; узнаем адрес
9115 F7             RST     30h
9116 00             DEFB    0
9117 8700           DEFW    87h        ; таблицы атрибутов
                                       ; плоскости Nosprz
9119 115292         LD      DE,xyspr   ; грузим адрес
                                       ; блока данных
911C EB             EX      DE,HL
911D 0608           LD      b,8        ; и длину блока
911F C38092         JP      ldirvm     ; переписываем в VRAM
  ;-----
9122 00000000 shipdt: DEFB    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
9126 00000000           0,0,0,0,0,0,0,0,0,0
912A 0000000000000000
9132 00000000000000000000
913C 0102060C         DEFB    1,2,6,12,28,78H
9140 1C78
9142 00000000         DEFB    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
9146 00000000           0,0,0,0,0,0,0,0,0,0
914A 0000000000000000
9152 00000000000000000000
915B 01020408         DEFB    1,2,4,8,16,32,64,248,248,56,68H,
915F 102040F8           68H,78H,216,232
9163 F838686878D8E8
916A 00000000         DEFB    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
916E 00000000           0
9172 00000000000000000000
917B 01020408         DEFB    1,2,4,8,16,48,64,81H,1,3,3,3,3,
917F 10304081           7,7,232,98H,88H,232,98H,8,252,4
9183 01030303030707E8
918B 9888E89808FC04
9192 00000000         DEFB    0,0,0,0,0,0,0,0,0
9196 0000000000
919B 01010206         DEFB    1,1,2,6,15,0,0,80H,80H,0,0,0,0,
919F 0F000080           192,48,6,7,7,7,7,7,7,7
91A3 80000000000C03006
91AB 07070707070707
91B2 7C82027E         DEFB    7CH,82H,2,7EH,81H,63,68,87H
91B6 813F4487
91BA 000000000000     DEFB    0,0,0,0,0,0
91C0 80C00000         DEFB    80H,192,0,0,0,0,1,0,0,0,12,2,1,
91C4 00000100           0,255,15,2,1,0,0,0,80H,255,255,0
91C8 00000C020100FF0F
```

```
91D0 020100000080FFFF00
91D9 007C7830         DEFB    0,7CH,78H,48,32,255,255,1,242,0,
91DD 20FFFF01                   0,0,0,224,80H
91E1 F200000000E080
91E8 00000000         DEFB    0,0,0,0,0,0,0,0,0,0
91EC 000000000000
91F2 01061860         DEFB    1,6,24,96,0,0,0,0,241,16,27H
91F6 00000000F11027
91FD 0000000000       DEFB    0,0,0,0,0
9202 0C81F008         DEFB    12,81H,240,8,7,0,0,0,0,0,248
9206 070000000000F8
920D 0000000000       DEFB    0,0,0,0,0
  ;-----
  ; данные для спрайтов ДВГУ и МАТФАК
9212 89DAAA8B sprz:   DEFB    137,218,170,139,138,0,127,255,
9216 8A007FFF                   193,192,0,113,170,170,115,34
921A C1C00071AAAA7322
9222 9F4444C4         DEFB    159,68,68,196,68,0,3,131,255,
9226 44000383FFFE0091           254,0,145,82,92,210,81
922E 525CD251
9232 36555556         DEFB    54,85,85,86,85,85,85,246,0,0,0,
9236 555555F6                   48,121,255,207,134
923A 0000003079FFCF86
9242 75554545         DEFB    117,85,69,69,67,65,69,66,0,0,0,
9246 43414542                   195,231,255,60,24
924A 000000C3E7FF3C18

  ;-----
  ; координаты xy спрайтов # Nosprz
9252 6010400E xyspr:DEFB    60h,10h,nosprz*4,14,60h,0E0h,
9256 60E0440E                       (nosprz+1)*4,14
  ;-----------------------------------------------
  ; заполнение VRAM данными
  ; [HL] - откуда, [DE] - куда, [BC] - x,y
925A D5  draw1:     PUSH    DE          ; сохранить адрес VRAM
925B CD8092         CALL    ldirvm      ; переписать блок
925E D1             POP     DE          ; считать адрес VRAM
925F 14             INC     D           ; следующая строка
9260 0D             DEC     C           ; если не все,
9261 20F7           JR      NZ,draw1    ; то повторить
9263 C9             RET                 ; иначе возврат
  ;-----------------------------------------------
  ; заполнение VRAM const
  ; [DE] - куда, [BC] - x,y, [a] - const
9264 D5     draw2:  PUSH    DE          ; сохранить адрес
9265 C5             PUSH    BC          ; VRAM и размеры
9266 CD7292         CALL    fillvm      ; заполнить строку
9269 C1             POP     BC          ; считать размеры
926A D1             POP     DE          ; и адрес VRAM
926B 14             INC     D           ; следующая строка
926C 0D             DEC     C           ; размер Y=Y-1
926D 20F5           JR      NZ,draw2    ; если не все,то
926F C9             RET                 ; повторить
  ;-----------------------------------------------
  ; заполнение VRAM 255
```

```
9270 3EFF fvmFF:    LD      A,255
  ;-----------------------------------------------
  ; заполнение VRAM const [a], len [b], adr [DE]
9272 CD7992 fillvm: CALL    wrvram      ; записать байт во VRAM
9275 13             INC     DE
9276 10FA           DJNZ    $-4         ; повторить, если надо
9278 C9             RET
  ;-----------------------------------------------
  ; запись [a] во VRAM [DE] - adr
9279 EB     wrvram: EX      DE,HL
927A F7             RST     30h
927B 00             DEFB    0
927C 4D00           DEFW    4Dh         ;записать в VRAM
927E EB             EX      DE,HL       ; (для msx-2)
927F C9             RET
     ;-----------------------------------------------
     ; пересылка RAM в VRAM [HL] source, [DE] DEst, [b] length
9280 C5     ldirvm: PUSH    BC          ; сохранить регистровую пару
9281 7B             LD      A,E         ; выбросить младший байт
9282 D399           OUT     (99h),a     ; адреса VRAM
9284 7A             LD      A,D         ; затем старший байт
9285 F640           OR      40h         ; выставить 6 бит в 1
9287 D399           OUT     (99h),a
9289 0E98           LD      c,98h       ; загрузить номер порта VDP
928B EDB3           OTIR                ; вывести блок
928D C1             POP     BC          ; считать регистровую пару
928E C9             RET
         ;===============================================
  ; Подпрограмма 'OKEAH'
928F 20313232 okean:  DEFB  ' 1223  13  13 1223 13  13 13  13'
9293 33202031
9297 3320203133203132
929F 3233203133202031
92A7 3320313320203133
92AF 31332020         DEFB  '13  13 13 13  13   13  13 13  13'
92B3 31332031
92B7 3320313320203133
92BF 2020203133202031
92C7 3320313320203133
92CF 31332020         DEFB  '13  13 1223   13   122223 13  13'
92D3 31332031
92D7 3232332020203133
92DF 2020203132323232
92E7 3320313320203133
92EF 31332020         DEFB  '13  13 123    123  13  13 122223'
92F3 31332031
92F7 3233202020203132
92FF 3320203133202031
9307 3320313232323233
930F 31332020         DEFB  '13  13 1223   13   13  13 13  13'
9313 31332031
9317 3232332020203133
```

```
931F 2020203133202031
9327 3320313320203133
932F 31332020         DEFB  '13  13 13 13  13    1223  13  13'
9333 31332031
9337 3320313320203133
933F 2020202031323233
9347 2020313320203133
934F 20313232         DEFB  ' 1223  13  13 1223   13   13  13'
9353 33202031
9357 3320203133203132
935F 3233202020313320
9367 2020313320203133
  ;------
936F F7D3C5D3 pbyok:  DEFB    247,211,197,211,207,192,218,206,
9373 CFC0DACE                   217,202,32,208,201,207,206
9377 D9CA20D0
937B C9CFCE
937E C5D2D3CB         DEFB    197,210,211,203,201,202,32,204,
9382 C9CA20CC                   193,199,197,210,216,32,227,235
9386 C1C7C5D2D820E3EB
938E 20F7ECEB         DEFB    32,247,236,235,243,237,0
9392 F3ED00
9395 28632920 cfib:   DEFB    '(c) ',199,210,213,208,208,193,
9399 C7D2D5D0                   ' F&B, ',247,240,236,' "'
939D D0C12046
93A1 26422C20F7F0EC2022
93AA EFCBC5C1         DEFB    239,203,197,193,206,'", 1988',0
93AE CE222C203139383800
93B7 E2D5C8D4 emar:   DEFB    226,213,200,212,193,32,229,205,
93BB C120E5CD                   193,210,0
93BF C1D200
93C2 3139383800 year: DEFB    '1988',0
  ;-----------------------------------------
  ocean@:   ; Рисуем изображение
93C7 218F92         LD      HL,okean    ; адрес данных
93CA 012007         LD      BC,0720h    ; размер YxX
93CD 110002         LD      DE,0200h    ; начальный адрес VRAM
93D0 CD4B90         CALL    draw        ; рисуем кирпичиками
93D3 3E0F           LD      a,0Fh       ; цветом 15
93D5 116F93         LD      DE,pbyOK    ; надпись 'Всесоюзный ...'
93D8 210812         LD      HL,1208h    ; с таких координат
93DB CD0994         CALL    print2
93DE 21A261         LD      HL,61A2h    ; точно также написать
93E1 11B793         LD      DE,emar     ; 'Бухта Емар'
93E4 3E0F           LD      A,0Fh
93E6 CD0994         CALL    print2
93E9 21AC73         LD      HL,73ACh
93EC 11C293         LD      DE,year     ; и '1988'
93EF 3E0F           LD      A,0Fh
93F1 CD0994         CALL    print2
93F4 111837         LD      DE,3718h    ; зарисовать строку с
                                        ; этими координатами
93F7 06D0           LD      B,0D0h      ; такой длины
93F9 3EE4           LD      A,0E4h      ; цветом 4 (фон)
```

```
93FB CD7292         CALL    fillvm
93FE 3E0E           LD      A,14        ; установить цвет 14
9400 21B81C         LD      HL,1CB8h    ; и такие координаты
9403 119593         LD      DE,cfib     ; взять текст '(c)
                                        ; группа F&B...'
9406 C30994         JP      print2      ; и вывести его на
                                        ; экран
  ;-----------------------------------------------
  ; Подпрограмма плотной печати в режиме SCREEN 2
  ; [HL] - x,y, [DE] - адрес надписи, [a] - цвет
9409 32E9F3  print2:ld      (0F3E9h),a ; устанавливаем цвет [a]
940C D5             PUSH    DE          ; сохраняем адрес текста
940D EB             EX      DE,HL       ; заносим в DE координаты
940E 21B9FC         LD      HL,0FCB9h   ; загружаем адрес системных
                                        ; координат
9411 73             LD      (HL),E      ; записываем координату Y
9412 2B             DEC     HL          ; получаем адрес системной
                                        ; координаты X
9413 2B             DEC     HL
9414 72             LD      (HL),D      ; записываем координату X
9415 D1             POP     DE          ; считываем адрес текста
9416 1A      l08:   LD      A,(DE)      ; грузим очередной байт
                                        ; текста
9417 B7             OR      A           ; проверяем: последний ?
9418 C8             RET     Z           ; если да, то возврат
9419 F7             RST     30h
941A 00             DEFB    0
941B 8D00           DEFW    8Dh         ; иначе печатаем его
941D 35             DEC     (HL)        ; приращение X = 6,а не 8
                                        ; (как в системе)
941E 35             DEC     (HL)
941F 13             INC     DE          ; следующий символ
9420 18F4           JR      l08         ; повторить
  ;-----------------------------------------------
  ;   Подпрограмма задержки
  ; вход на TIME+3, [DE] - кол-во циклов
  ; вход на TIME, кол-во циклов = FFFF
9422 1B     time:   DEC     DE          ; уменьшить,
9423 7A             LD      A,D         ; если DE <> 0
9424 B3             OR      E
9425 20FB           JR      NZ,time     ; то повторить
9427 C9             RET                 ; иначе возврат
  ;-----------------------------------------------
9428 F5     EXchg:  PUSH    AF
9429 E5             PUSH    HL
942A 214D94         LD      HL,EXxd
942D 7E             LD      A,(HL)
942E 70             LD      (HL),B
942F 47             LD      B,A
9430 23             INC     HL
9431 7E             LD      A,(HL)
9432 71             LD      (HL),C
9433 4F             LD      C,A
9434 23             INC     HL
9435 7E             LD      A,(HL)
```

```
9436 72             LD      (HL),D
9437 57             LD      D,A
9438 23             INC     HL
9439 7E             LD      A,(HL)
943A 73             LD      (HL),E
943B 5F             LD      E,A
943C E1             POP     HL
943D D5             PUSH    DE
943E ED5B5194       LD      DE,(EXxd+4)
9442 225194         LD      (EXxd+4),HL
9445 EB             EX      DE,HL
9446 D1             POP     DE
9447 F1             POP     AF
9448 C9             RET
  ;-----------------------------------------------
9449        const:  DEFS    4
944D        EXxd:   DEFS    6
                    END
```

ПРИЛОЖЕНИЕ 3. ЛИСТИНГ ПРОГРАММЫ УПРАВЛЕНИЯ СПРАЙТОМ

Ниже приводится листинг программы управления спрайтом при помощи джойстика. Если у Вас его нет, можно воспользоваться MSX-мышью. Для работы мыши в режиме джойстика при включении компьютера или устанавливая мышь в разъем, держите нажатой левую кнопку мыши. Программа оттранслирована ассемблером системы DUAD.

```
'sprite ctrl'     Z80-Assembler   Page:    1
                        title   'sprite ctrl'
                ; константы
0001 =          nospr   EQU     1           ; номер активного спрайта
0001 =          step    EQU     1           ; шаг приращения координат
                        ORG     9000h       ;начальный адрес прог.
                ; вводим номер джойстика
9000 CDC000             CALL    0C0h        ;beep
9003 21AEF3             LD      HL,0F3AEh   ;разм. экр. в screen 0
9006 3628               LD      (HL),40     ;width 40
9008 210F01             LD      HL,010Fh
900B 22E9F3             LD      (0F3E9h),HL;colOR 15,1
900E 21E0F3             LD      HL,0F3E0h   ;vdp(1)
9011 CBCE               SET     1,(HL)      ;16*16
9013 CB86               RES     0,(HL)      ;норм. размер спрайта
9015 CD6C00             CALL    6Ch         ;screen 0
9018 CDCC00             CALL    0CCh        ;key off
901B 21F190             LD      HL,input    ;Введите номер джойс.
                ; вывод сообщения на экран
901E 7E                 LD      a,(HL)
901F B7                 OR      a           ;если код символа = 0,
9020 2806               JR      z,l02       ;то закончить вывод
9022 CDA200             CALL    0A2h        ;вывод символа (A)
                                            ;на экран
9025 23                 INC     HL          ;адрес след. символа
9026 18F6               JR      $-8         ;повторить вывод
9028 CD9F00     l02:    CALL    9Fh         ;ввести символ с клав.
902B 0600               LD      b,0         ;номер джойстика = 0
902D D630               SUB     '0'         ;проверяем:
902F 280A               JR      z,l01       ;нажато "0", переходим
9031 04                 INC     b           ;если нет, то джойс.=1
9032 3D                 DEC     a           ;нажата "1"
9033 2806               JR      z,l01       ;если да, то переходим
9035 04                 INC     b           ;джойстик = 2
9036 3D                 DEC     a           ;если нажата "2",
9037 2802               JR      z,l01       ;переходим
9039 18ED               JR      l02         ;иначе вводим снова
                ; создаем шаблон номер NoSpr
903B CD7200     l01:    CALL    72h         ;screen 2
903E C5                 PUSH    BC          ;сохр. номер джойстика
903F 3E01               LD      a,nospr     ;грузим номер спрайта
9041 CD8400             CALL    84h         ;узнаем адрес шаблона
9044 111191             LD      DE,sprdat   ;грузим адрес данных
9047 012000             LD      BC,32       ;длина данных
904A EB                 EX      DE,HL
904B CD5C00             CALL    5Ch         ;перепис. блок во VRAM
```

```
                ; выводим спрайт на экран
904E 3E01       putspr: LD      a,nospr   ;грузим номер спрайта
9050 CD8700             CALL    87h       ;узнаем адр.табл.атр.
9053 113191             LD      DE,y      ;адрес блока данных
9056 010400             LD      BC,4      ;длина блока
9059 EB                 EX      DE,HL
905A CD5C00             CALL    5Ch       ;пересылаем блок
                ; задержка
905D 110001             LD      DE,100h   ;кол-во пустых циклов
9060 1B                 DEC     DE
9061 7A                 LD      a,d
9062 B3                 OR      e
9063 20FB               JR      nz,$-3    ;если <> 0, то повт.
                ; вводим stick(b)
9065 C1         stick:  POP     BC        ;считыв. номер джойст.
9066 CDB700             CALL    0B7h      ;пров., не нажато ли
                                          ;ctrl+STOP
9069 D8                 RET     c         ;если да, то возврат
906A 78                 LD      a,b       ;A = номеру джойстика
906B C5                 PUSH    BC        ;опять сохр. номер дж.
906C CDD500             CALL    0D5h      ;вводим напр. джойс.
906F B7                 OR      a         ;если ничего не нажато,
9070 28F3               JR      z,stick   ;то ввод снова
9072 0E01               LD      c,step    ;загр. значение шага
                ; вверх
9074 3D                 DEC     a         ;нажато вверх ?
9075 2005               JR      nz,ur     ;если нет, то следующ.
9077 CDBD90             CALL    moveUP    ;иначе - уменьшение Y
907A 18D2               JR      putspr    ;и переходим на  спр.
                ; вверх/вправо
907C 3D  ur:            DEC     a         ;то же, что и выше
907D 2008               JR      nz,right
907F CDBD90             CALL    moveUP
9082 CDC890             CALL    moveRG
9085 18C7               JR      putspr
                ; вправо
9087 3D         right:  DEC     a
9088 2005               JR      nz,rd
908A CDC890             CALL    moveRG
908D 18BF               JR      putspr
                ; вправо/вниз
908F 3D         rd:     DEC     a
9090 2008               JR      nz,down
9092 CDC890             CALL    moveRG
9095 CDD690             CALL    moveDW
9098 18B4               JR      putspr
                ; вниз
909A 3D         down:   DEC     a
909B 2005               JR      nz,dl
909D CDD690             CALL    moveDW
90A0 18AC               JR      putspr
                ; вниз/влево
90A2 3D  dl:            DEC     a
90A3 2008               JR      nz,left
90A5 CDD690             CALL    moveDW
```

```
90A8 CDE490           CALL    moveLF
90AB 18A1             JR      putspr
                ; влево
90AD 3D       left:   DEC     a
90AE 2005             JR      nz,lu
90B0 CDE490           CALL    moveLF
90B3 1899             JR      putspr
                ; влево/вверх
90B5 CDE490   lu:     CALL    moveLF
90B8 CDBD90           CALL    moveUP
90BB 1891             JR      putspr

                ; уменьшение ячейки Y на значение шага (в регистре C)
90BD 3A3191   moveUP: LD      a,(y)     ;A = (Y)
90C0 91               SUB     c         ;A = A - шаг
90C1 DCEF90           CALL    c,lda0    ;если < 0, то A=0
90C4 323191           LD      (y),a     ;сохраняем A в (Y)
90C7 C9               RET
90C8 3A3291   moveRG: LD      a,(x)     ;A = (X)
90CB 81               ADD     a,c       ;A = A + шаг
90CC FEF0             CP      240       ;пров: A<240 ?
90CE 3802             JR      c,$+4     ;если нет,
90D0 3EEF             LD      a,239     ;то A=239
90D2 323291           LD      (x),a     ;сохраняем A в (X)
90D5 C9               RET
90D6 3A3191   moveDW: LD      a,(y)     ;A = (Y)
90D9 81               ADD     a,c       ;A = A + шаг
90DA FEB0             CP      176       ;пров: A<176
90DC 3802             JR      c,$+4     ;если нет,
90DE 3EAF             LD      a,175     ;то A=175
90E0 323191           LD      (y),a     ;сохраняем A в (Y)
90E3 C9               RET
90E4 3A3291   moveLF: LD      a,(x)     ;A = (X)
90E7 91               SUB     c         ;A = A - шаг
90E8 DCEF90           CALL    c,lda0    ;если < 0, то A=0
90EB 323291           LD      (x),a     ;сохраняем A в (X)
90EE C9               RET
90EF AF       lda0:   XOR     a         ;A = 0
90F0 C9               RET
90F1 F7D7C5C4 input:  DEFB  'Введите номер джойстика (0-2): ',0
90F5 C9D4C520
90F9 CECFCDC5D220C4D6
9101 CFCAD3D4C9CBC120
9109 28302D32293A2000
                ;таблица шаблона номер NOSPR
9111 F0818181 sprdat: DEFB  240,129,129,129,129,1,0,28,0,1,129
9115 8101001C
9119 000181
911C 818180F0         DEFB  129,129,128,240,0,30,2,2,2,2,0,0
9120 001E020202020000
9128 70000002         DEFB    112,0,0,2,2,2,2,30,0
912C 0202021E00
```

```
                ; таблица атрибутов спрайта номер NOSPR
9131 58         y:      DEFB    88          ;координата Y
9132 80         x:      DEFB    128         ;координата X
9133 04                 DEFB    nospr*4     ;номер шаблона спрайта
9134 0F         clrspr: DEFB    15          ;цвет спрайта
                        END
```

ПРИЛОЖЕНИЕ 4. ПРИМЕР ОРГАНИЗАЦИИ СВЯЗЕЙ С ЯЗЫКОМ MSX-BASIC. "Универсальное меню"

Приведенная ниже программа на языке MSX-BASIC вызывает подпрограмму на языке ассемблера и передает ей адрес строкового массива, используя функцию VARPTR. Массив представляет собой меню. Строка массива, начинающаяся с символа пробел, считается комментарием. Пустая строка ("") является признаком конца меню.

Подпрограмма на ассемблере рисует на экране (параметры SCREEN 0, WIDTH 80 - устанавливаются заранее) окно, размеры которого определяются программно, исходя из параметров переданного массива (Y - по количеству элементов массива, т.е. до пустой строки, X - по наиболее длинной строке). Координаты окна программа на языке MSX-BASIC передает функцией LOCATE X,Y, используя тот факт, что интерпретатор записывает значения X и Y в системные ячейки.

Строка-курсор устанавливается на первый возможный вариант выбора меню (комментарии пропускаются). Эту строку можно перемещать в пределах окна с помощью клавиш вверх/вниз. Выбор осуществляется нажатием клавиши ввод.

Выбранная строка отмечается галочкой, и программе на языке MSX-BASIC возвращается ее номер. Если в момент выбора были нажаты клавиши <CTRL>+<Stop>, возвращается ноль. Обратите внимание:

- Пользователь MSX-BASIC должен сам следить за размещением окна на экране.
- В меню должен быть хотя бы один возможный выбор.

1. BASIC-программа

```
 0 rem (c) 1990 И.Бочаров
10 CLEAR 200,&HDC00             ' Резервируем память для подпр.
20 DIM A$(17)                   ' Определяем массив-меню
30 BLOAD"wnd.obj"               ' Загружаем подпрограмму
40 VDP(13)=&HA4:COLOR = (4,0,0,4)' Устанавливаем цвета окна
50 VDP(14)=&HF0
60 DATA "     Select:"," ==============",Brief,Full,Info,Tree,
On/Off        ^F1," ---------------",Name,Extension,Size,Time,
UnsORted," --------------",Help infORmation,Exit to main menu,""
70 FOR I=0 TO 16                ' Считываем меню
80    READ A$(I)
90 NEXT
100 DEFUSR=&HDC00               ' Адрес подпрограммы-меню
110 DEFUSR1=&HDC03              ' Подпрограмма очистки экрана
130 :
140 LOCATE 2,1                  ' Устанавливаем координаты окна
150 GOSUB 230                   ' Вызываем подпрограмму
160 LOCATE 20,6
170 GOSUB 230
180 LOCATE 26,0
190 GOSUB 230
200 GOTO 140 ' Зацикливаемся
210 :
230 A=USR1(0) ' Очищаем цвета на экране
240 A=USR(VARPTR(A$(0))) ' Вызываем подпрограмму
250 IF A THEN LOCATE 75,22:PRINT A;:RETURN ELSE CLS:PRINT USR1(
```

```
"<Ctrl><Stop>"):END             ' Обрабатываем результат
260 rem The end.
```

2. Подпрограмма на языке ассемблера

```
;               (c) 1990 by IgOR BOCHAROV.
;
;--------------- Константы и адреса --------------------------
;
cr      EQU     13              ; Клавиша "Конец выбора"
Up      EQU     30              ; Клавиша вверх
Down    EQU     31              ; Клавиша вниз
TheBeg  EQU     0DC00h          ; Отсюда наша программа трансл.
GetHL   EQU     2F8Ah           ; Подпрограммы передачи значен.
PutHL   EQU     2F99h
Base1   EQU     0F3B5h          ; Здесь хран. адрес табл. цвет.
CsrX    EQU     0F3DDh          ; Сюда LOCATE пишет X
CsrY    EQU     0F3DCh          ; А сюда Y
RdVram  EQU     004Ah           ; Чтение из VRAM
WrVrHL  EQU     004Dh           ; Запись во VRAM
SetWrt  EQU     0053h           ; Установка VDP для записи
ChSns   EQU     009Ch           ; Опрос состояния буфера клав.
ChGet   EQU     009Fh           ; Взятие символа из буфера
BreakX  EQU     00B7h           ; Опрос <CTRL>+<Stop>
;-------------------------------------------------------------
        ORG     TheBeg
;------------------- Точки входа ----------------------------
        JP      Menu            ; Универсальное меню
        JP      ClMenu          ; Очищение таблицы цветов
;-------------- Вычисляем размер окна ------------------------
Menu:   CALL    GetHL           ; [HL] - адрес первого элемента
        PUSH    HL              ; Адрес первого элемента
        LD      IY,Variab       ; Адрес блока переменных
        XOR     a               ; Инициируем переменный
        LD      (IY+dy),a       ; Число строк
        LD      (IY+dx),a       ; dx
l001:   LD      a,(HL)          ; Длина очередной строки
        AND     a               ; Конец?
        JR      Z,l002          ; Если да, то выходим
        INC     (IY+dy)         ; Увеличиваем число строк
        INC     HL              ; + длина
        INC     HL              ; + адрес
        INC     HL
        CP      (IY+dx)         ; Больше dx?
        JR      C,l001          ; Если нет, то dx не меняем
        LD      (IY+dx),A
        JR      l001
;--------------- Вычисляем координаты окна -------------------
l002:   LD      HL,(CsrY)       ; x y
        DEC     H
        DEC     L
        LD      C,H             ; x
        LD      B,0
```

```
        LD      H,B
        ADD     HL,HL           ; * 2
        ADD     HL,HL           ; * 4
        ADD     HL,HL           ; * 8
        ADD     HL,HL           ; * 16
        LD      E,L
        LD      D,H
        ADD     HL,HL           ; * 32
        ADD     HL,HL           ; * 64
        ADD     HL,DE           ; * 80
        ADD HL,BC
        EX      DE,HL
        INC     (IY+dx)         ; dx
        INC     (IY+dx)         ; dx
        LD      b,(IY+dx)       ; dx
        LD      c,(IY+dy)       ; dy
        PUSH    DE              ; Для печати текста
;------------------- Рисуем окно -----------------------------
; [DE] - x + y*80, [b] - dx, [c] - dy
        PUSH    BC
        PUSH    DE              ; Для подсветки
; Рисование окантовки окна
        LD      a,17h           ;  ─
        LD      HL,1819h        ; ┌ ┐
        CALL    WndL01
WndL02:
        LD      a,' '           ; ' '
        LD      HL,1616h        ; │ │
        CALL    WndL01
        DEC     c
        JR      NZ,WndL02
        LD      a,17h           ;  ─
        LD      HL,1A1Bh        ; └ ┘
        CALL    WndL01
;--- Теперь подсвечиваем -------------
NEXtColORInWindow:
        POP     HL              ; x,y
        POP     DE              ; dx,dy
        INC     D
        INC     D
        INC     D               ; dx = dx + 4
        INC     D
        INC     E               ; dy = dy + 2
        INC     E
WndLp1:
        PUSH    DE
        PUSH    HL
        CALL    FillColORTable  ; Заполняем строку
        POP     HL
```

```
        LD      DE,80           ; [y] = [y] + 1
        ADD     HL,DE
        POP     DE
        DEC     E               ; [dy] = [dy] - 1
        JR      NZ,WndLp1
;------------------- Пишем текст в окне -----------------------
        POP     HL
        LD      DE,80*1+3       ; Откуда начать писать
        ADD     HL,DE
        LD      D,H
        LD      E,L
        EX      (SP),HL         ; Адрес первой строки
1003:   LD      A,(HL)          ; Длина очередной строки
        AND     A               ; Конец?
        JR      Z,1004          ; Если да, то выходим
        LD      B,A             ; Текущий dx
        PUSH    HL
        PUSH    DE
        DI
        LD      A,E             ; Младший байт адреса
        OUT     (99h),A
        LD      A,D             ; затем старший байт
        OR      40h             ; выставив 6 бит в 1
        OUT     (99h),A
        INC     HL
        LD      A,(HL)          ; Получаем адрес строки
        INC     HL
        LD      H,(HL)
        LD      L,A
        LD      C,98h           ; загрузить номер порта VDP
        OTIR                    ; вывести блок
        EI
        POP     HL
        LD      DE,80
        ADD     HL,DE
        EX      DE,HL           ; [DE] - следующий адрес
        POP     HL
        INC     HL
        INC     HL
        INC     HL
        JR      1003            ; Повторяем вывод
;------------------- Выбор в меню -----------------------------
1004:   POP     HL              ; Адрес первой строки
        LD      C,0             ; Текущее состояние
        LD      D,C
        LD      E,D
1007:   PUSH    HL
1008:   ADD     HL,DE
        CALL    RdVram
        CP      ' '             ; Пробел?
        JR      NZ,1006
        LD      DE,80
        JR      1008
1006:   POP     DE
```

```
        CP      17h             ; —?
        JR      NZ,l009
        EX      DE,HL
        DEC     C
l009:   INC     C
        PUSH    BC              ; Состояние
        PUSH    HL              ; Стираем засветку
        LD      D,(IY+dx)
        DEC     HL
        CALL    ClearColORTable
        POP     HL
        POP     BC
;-------------------
InputKey:
        CALL    BreakX
        JR      NC,Input
        LD      C,0
        JR      Exit
Input:  CALL    ChSns           ; Есть что-нибудь в буфере?
        JR      Z,Inputkey
        CALL    ChGet           ; Берем символ
;-------------------
        CP      cr
        JR      NZ,CheckUp
        LD      A,'√'
        DEC     HL
        CALL    WrVrHL
Exit:   LD      H,0
        LD      L,C             ; Текущее состояние
        JP      PutHL
;-------------------
CheckUp:
        CP      Up
        JR      NZ,CheckDown
        LD      A,C
        DEC     A               ; А можно ли наверх?
        JR      Z,InputKey
        LD      C,A
        PUSH    BC
        PUSH    HL
        LD      D,(IY+dx)
        DEC     HL
        CALL    FillColORTable  ; Стираем курсор
        POP     HL
        LD      DE,-80          ; Смещение на строку вверх
l010:   ADD     HL,DE           ; Новая позиция курсора
        CALL    RdVram
        CP      ' '
        JR      Z,l010          ; Сканируем пробелы
        LD      D,(IY+dx)
        PUSH    HL
        DEC     HL
        CALL    ClearColORTable
        POP     HL
        POP     BC
```

```
        JR      InputKey
;-------------------
CheckDown:CP    Down
        JR      NZ,InputKey
        PUSH    HL
        LD      D,(IY+dx)
        PUSH    BC
        DEC     HL
        CALL    FillColORTable  ; Стираем курсор
        POP     BC
        POP     HL
        LD      DE,80
        JR      l007
;---------------------------------------------------------------
WndL01:
        PUSH    BC
        PUSH    DE
        PUSH    AF
        PUSH    BC
        LD      A,' '
        CALL    WrVram
        LD      A,H
        CALL    WrVram
        POP     BC
        POP     AF
        CALL    FillVm
        LD      A,L
        CALL    WrVram
        LD      A,' '
        CALL    WrVram
        POP     HL
        LD      BC,80
        ADD     HL,BC
        EX      DE,HL
        POP     BC
        RET
;---------------------------------------------------------------
; Рисует цветную линию по координатам и размеру
; [HL] - y *80+ x; [d] - dx
; MoDIfy: AF, BC, DE, HL
FillColORTable:
        LD      E,10000000b
        CALL    GetMaskColOR    ; Получить адрес и маску
WndLp4: CALL    RdVram          ; считать текущий байт
WndLp3: OR      E
        DEC     D               ; [dx] = [dx] - 1
        JP      z,WrVrHL
        RRC     E               ; Следующая маска
        JR      NC,WndLp3       ; Если не вышли из байта, то не
                                ; пишем
        CALL    WrVrHL          ; Иначе записываем
```

```
        INC     HL              ; Следующий адрес VRAM
        JR      WndLp4
;---------------------------------------------------------------
; Очищает цветную линию
; [HL] - y *80+ x; [d] - dx
; MoDIfy: AF, BC, DE, HL
ClearColORTable:
        LD      E,01111111b
        CALL    GetMaskColOR    ; Получить адрес VRAM и маску
WndLp5: CALL    RdVram          ; считать текущий байт
WndLp6: AND     E
        DEC     D               ; [dx] = [dx] - 1
        JP      Z,WrVrHL
        RRC     E               ; Следующая маска
        JR      C,WndLp6        ; Если не вышли из байта, то не
                                ; пишем
        CALL    WrVrHL          ; Иначе записываем
        INC     HL              ; Следующий адрес VRAM
        JR      WndLp5
;---------------------------------------------------------------
; По координатам выдает адрес VRAM и маску
; [HL] - y*80 + x;
; [HL] - адрес Vram; [e] - маска
GetMaskColOR:
        LD      A,L             ; Получаем из коор. физ. адрес
        LD      B,3
        SRL     H               ; [HL] = [HL] / 8
        RR      L
        DJNZ    $-4
        LD      BC,(Base1)
        ADD     HL,BC           ; Физический адрес VRAM
        CPl                     ; [a] = not([a])
        AND     00000111b       ; Бит, который надо установить
        LD      B,A
        INC     B
        RLC     E
        DJNZ    $-2
        RET
;---------------------------------------------------------------
; Очистка цветной таблицы

ClMenu: LD      DE,(Base1)
        LD      B,24*10
        XOR     A
;---------------------------------------------------------------
; заполнение VRAM const [a], len [b], adr [DE]
FillVm: DI
        PUSH    AF
        LD      A,E
        OUT     (99h),A
        LD      A,D
        OR      40h
        OUT     (99h),A
        POP     AF
```

```
        EX      (SP),HL
        EX      (SP),HL
        OUT     (98h),A
        INC     DE
        djNZ    $-3             ; повторить, если надо
        RET
;-----------------------------------------------------------------
WrVram: EX      DE,HL
        CALL    WrVrHL
        EX      DE,HL
        INC     DE
        RET
;-----------------------------------------------------------------
Variab  EQU     $
dy      EQU     1
dx      EQU     2
        END
```

ОГЛАВЛЕНИЕ