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.

2.1. Директивы ассемблера

С помощью директив (псевдокоманд) программист даёт 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 имён 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 её эксплуатации. Пример:

  ```
  scrnum    EQU   2
  nospr     EQU   16
  dma       EQU   fcb+len*3
  argum     EQU   0A001h
  ```

- `.REQUEST` — просмотр неопределённых 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')` — даёт 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.

2.2. 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 даёт возможность достаточно просто умножать/делить на любую целую степень двойки.

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.

2.3. 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 приведённых 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, $, 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	≤
GT	>
GE	≥

Текущий адрес обозначается знаком «$» или «Ъ». Например, если текущий счётчик 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
```.

2.4. 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-разрядной загрузки, приведённые 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 её отличие от следующей за ней команды.

Приведём 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 пришёл, 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 произойдёт запись HL ⇒ BC, BC ⇒ DE, DE ⇒ HL.

2.5. 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 перекрёстных ссылок.
- `.XCREF` — прекращение выдачи файла перекрёстных ссылок.
- `SUBTTL текст` — печать подзаголовка титула.
- `*eject выражение` — новая страница размером «выражение».

2.6. Арифметические команды

К арифметическим командам Z80 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.

2.6.1. Представление операндов

При выполнении арифметических команд каждый операнд обычно представляется как 8-разрядное число со знаком в старшем разряде, в дополнительном двоичном коде.

Двоичное	Шестнадцатеричное	Десятичное
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	Шестнадцатеричное	Десятичное
0111 1111 1111 1111	7FFF	32767
0111 1111 1111 1110	7FFE	32766
…	…	…
0000 0000 0000 0011	0003	3

Двоичное	Шестнадцатеричное	Десятичное
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 одно–байтовых 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,

Двоичная запись	Шестнадцатеричная	Десятичная	Двоично–десятичная (BCD)
0011 0111	37	55	37
1001 0110	96	150	96
0001 1010	1A	26	-

2.6.2. Работа с восьмиразрядными числами

При выполнении команд один из операндов обычно должен быть помещён 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 задаётся 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 учёт значения бита заёма 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 ещё раз
```

Команда сложения или вычитания двух чисел, представленных в двоично-десятичном формате BCD, даёт 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 её 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 приведём 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 аккумуляторе, 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.

2.6.3. 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 четырёх 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
```

2.7. 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 Z80 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 (неэквивалентность): 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 определённых 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 определённые 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 аккумулятора, 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 приведём 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 Z80 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:

7	6	5	4	3	2	1	0

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
```

2.8. Команды перехода и условного перехода

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

Порядок выполнения программы может быть изменён, если занести в регистр счётчика 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 счётчика 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 определённого 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

Мнемоника	Условие	Флаг	Код CCC
PO (Parity Odd)	нечётный результат	P =0	100
PE (Parity Even)	чётный 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 вперёд/назад на -126..+129 ячеек памяти и используются для написания перемещаемых программ.

Хотя один байт обычно задаёт смещение -128..+127, здесь нужно учитывать, что счётчик 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 ещё 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 произойдёт 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 определённое 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 приведём 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
```

Приведём 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.

2.9. 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 Arithmetical, SRL — Shift Right Logical);
- циклический сдвиг (RLCA, RLC, RRCA, RRC);
- циклический сдвиг через флаг C (RLA, RL, RRA, RR);
- перестановка полубайт (RLD, RRD).

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

Схема работы команды SLA:

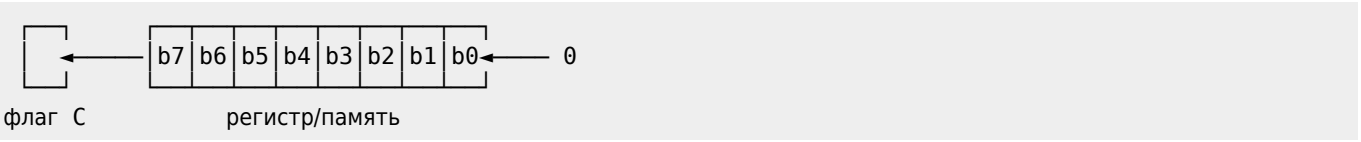


Схема работы команды SRA:

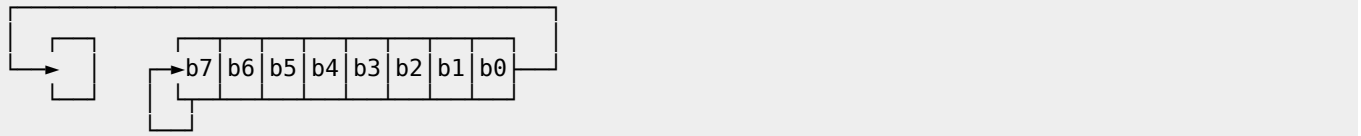
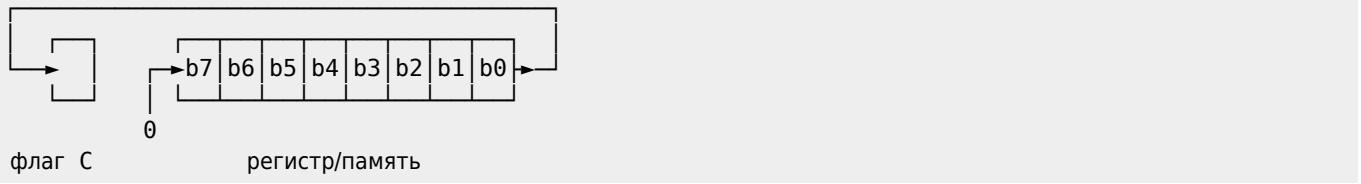


Схема работы команды SRL:



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

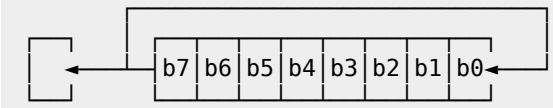
Арифметический сдвиг влево SLA можно использовать для умножения на степень двойки, а арифметический сдвиг вправо SRA — для деления на два без остатка (нацело).

Приведём 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
```

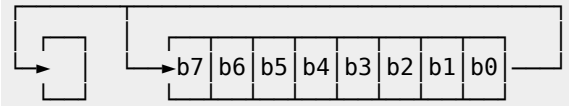
Команды циклического сдвига сдвигают содержимое регистра или байта памяти влево или вправо на один бит. При этом выдвинувшийся за разрядную сетку бит не теряется, а переносится на первое место с другого конца байта.

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.

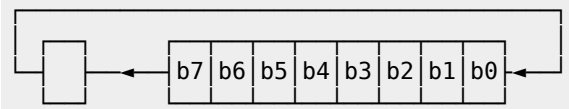
Схема работы команды RRCA:



<code>

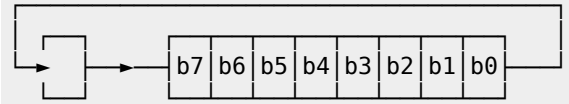
Команда ''RRC'' выполняется аналогично над регистром или памятью.

Кроме этого можно использовать команды циклического сдвига влево или вправо через флаг C.
Схема работы команды ''RLA'':
<code>



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 берёт 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:

Схема работы команды RRD:

Изучите приведённый 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
```

2.10. 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:

- `LDI` — пересылка байта с инкрементом выполняется так:

```
1) LD     (DE),(HL)
2) INC    HL
3) INC    DE
4) DEC    BC
```

- `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
```

- `LDD` — пересылка байта с декрементом:

```
1) LD     (DE),(HL)
2) DEC    HL
3) DEC    DE
4) DEC    BC
```

- `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 неё 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 разберём 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 её 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
```

2.11. 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:

- CPI — сравнение A с байтом памяти с инкрементом:

```
1) CP     A,(HL)
2) INC    HL
3) DEC    BC
```

- 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
```

- CPD — сравнение A с байтом с декрементом:

```
1) CP     A,(HL)
2) DEC    HL
3) DEC    BC
```

- 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 найдётся, 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
```

2.12. 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 счётчик команд PC микропроцессора загружается величина adr, т.е. адрес первой команды вызываемой подпрограммы. Таким образом, управление из основной программы передаётся 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 счётчик команд PC микропроцессора, и управление вновь передаётся 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 отсутствует, то вызов подпрограммы или возврат из неё 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.

В Приложении 1 приведена таблица соответствия между этими командами и шестнадцатеричными адресами ячеек памяти, куда передаётся управление при их выполнении.

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

В программе на языке ассемблера могут использоваться внешние переменные, то есть переменные, определённые 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 её 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
```

2.13. 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.

2.13.1. Клавиатура

Для работы с клавиатурой применяются следующие подпрограммы BIOS: чтение статуса строки матрицы клавиатуры (`SNSMAT, 0141H`), обнаружение нажатия клавиш CTRL+STOP при отключённых 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
```

2.13.2. 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
```

2.13.3. Графика

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 16×16 точек учтите, что номер шаблона определяется так же, как и для спрайтов размером 8×8, 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.

2.13.4. 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.

Приведённая ниже программа «щёлкает» 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.

2.13.5. 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 её 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
```

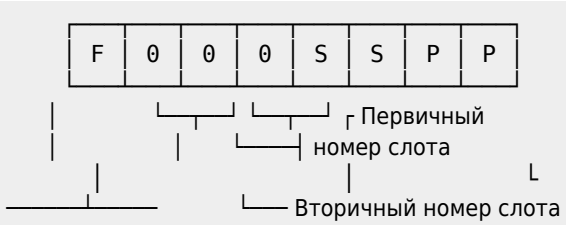
2.13.6. Межслотовые вызовы подпрограмм

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:



0 — Вторичный слот не используется
1 — Используется вторичный слот

При межслотовом вызове `CALSLT` и `CALLF` прерывания автоматически запрещаются. При возврате управления из этих подпрограмм в MSX-1 прерывания остаются запрещёнными, 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
```

2.13.7. 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 приведём 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
```

2.14. Ловушки

Работа компьютера MSX-2 практически состоит в выполнении набора определённых 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 начнётся работа стандартной подпрограммы.

Приведём 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 её 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.

2.14.1. 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 BASIC.

```
                  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 наберёте команду `FILES`, будет очищен экран, выведена надпись «Список файлов:» и сам список файлов.

2.14.2. 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+$ даёт 0).

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

Приведём 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 её 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 Ввод⏎. Нажатие клавиши STOP будет обрабатываться как нажатие двух клавиш — STOP+CTRL.

2.15. 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 BASIC нужно по мере достаточной необходимости.

Список входных точек и правила вызова операций и функций MSX BASIC приведены в книге «Архитектура микрокомпьютера MSX-2».

Не забывайте, что вызов подпрограмм MSX BASIC возможен, например, либо если Вы работаете в обычном режиме MSX BASIC (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 BASIC. 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.

2.15.1. 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
```

2.15.2. Работа с вещественными числами

При работе с вещественными 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:

- бит 7: если 1, то вывод осуществляется по формату;
- бит 6: если 1, то через каждые 3 цифры вставляются запятые;
- бит 5: если 1, то первые нули нужно заменить на символ «*»;
- бит 4: если 1, то перед числом вставить символ «$»;
- бит 3: если 1, то число выводится всегда со знаком;
- бит 2: если 1, то вставить знак после числа;
- бит 1: не используется;
- бит 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.

Приведём 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.

2.16. 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 приведён в книге «Архитектура микрокомпьютера MSX-2». Каждая функция имеет свой код.

Для вызова функции BDOS необходимо:

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

Вызов функций BDOS поддерживается и интерпретатором MSX Disk BASIC 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. Причём адрес строки должен быть записан в регистровую пару `DE`, а конец строки обозначается символом «$».

Ниже приведён 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
```

2.17. 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/приёма 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, приведённая выше, может быть с учётом 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 приведём листинг программы Host.mac. Эта программа состоит из двух частей. Первая часть, вызываемая оператором USR из MSX BASIC 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 передаётся 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
```

2.18. Работа с портами ввода/вывода

Перейдём к командам ввода/вывода. В процессоре Z80 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.

2.19. 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 её вызовом отключать прерывания командой DI, а в конце работы с VDP — снова их активировать командой EI.

2.19.1. 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
```

Ещё 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 запрещён. 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. Её 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 приведём 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
```

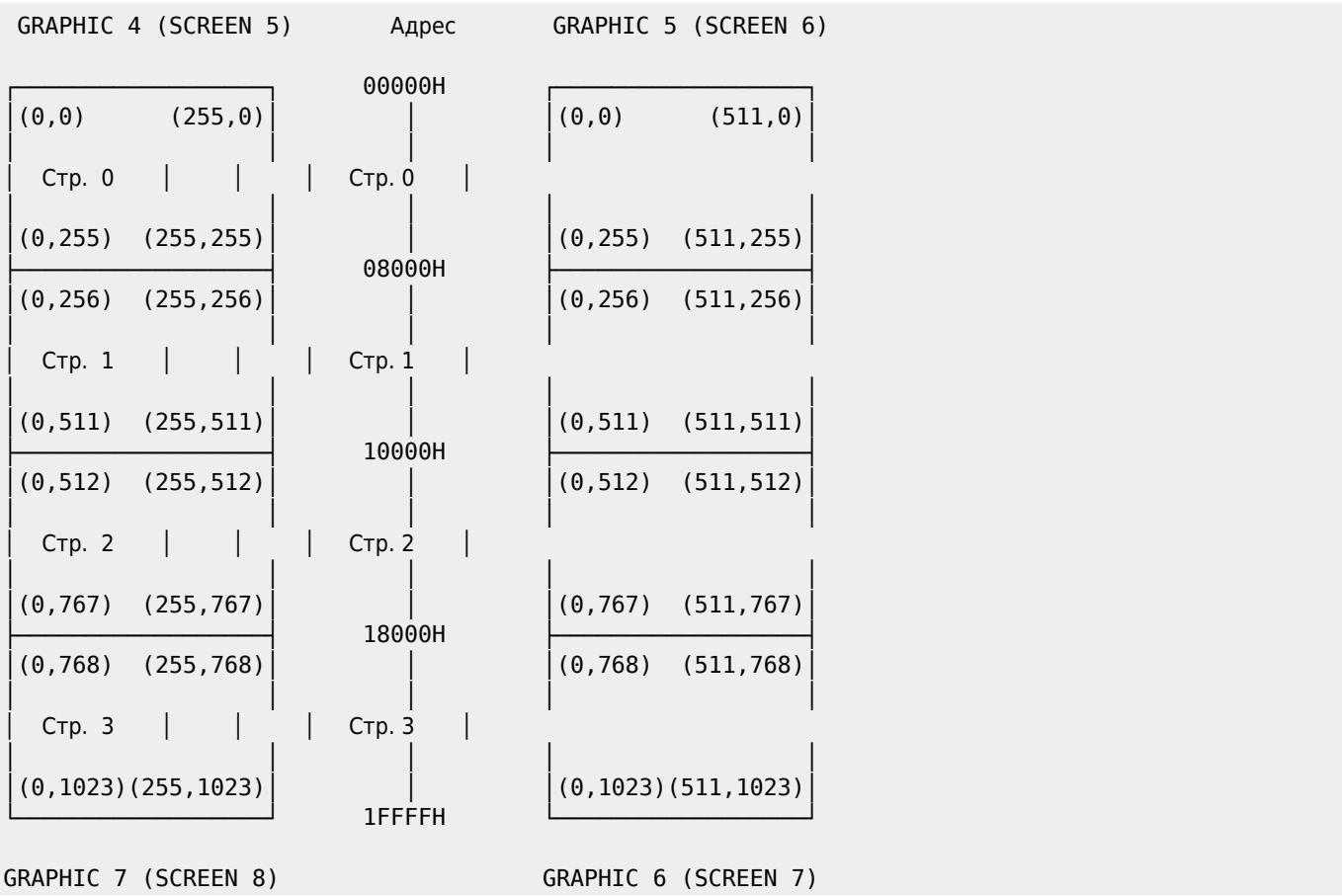
```
90D1 D399      OUT     (99h),A ; байт адреса  VRAM
90D3 7A        LD      A,D     ; выбрасываем старший
90D4 F640      OR      40h     ; байт адреса  VRAM
90D6 D399      OUT     (99h),A ; установив 6 бит в 1
90D8 EDB3      OTIR            ; переписываем блок
  ; === Ширина экрана  - 80 символов
90DA 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
```

2.19.2. Использование команд видеопроцессора

Fix Me!

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 Кб VRAM. Она приведена на рис.19.1.



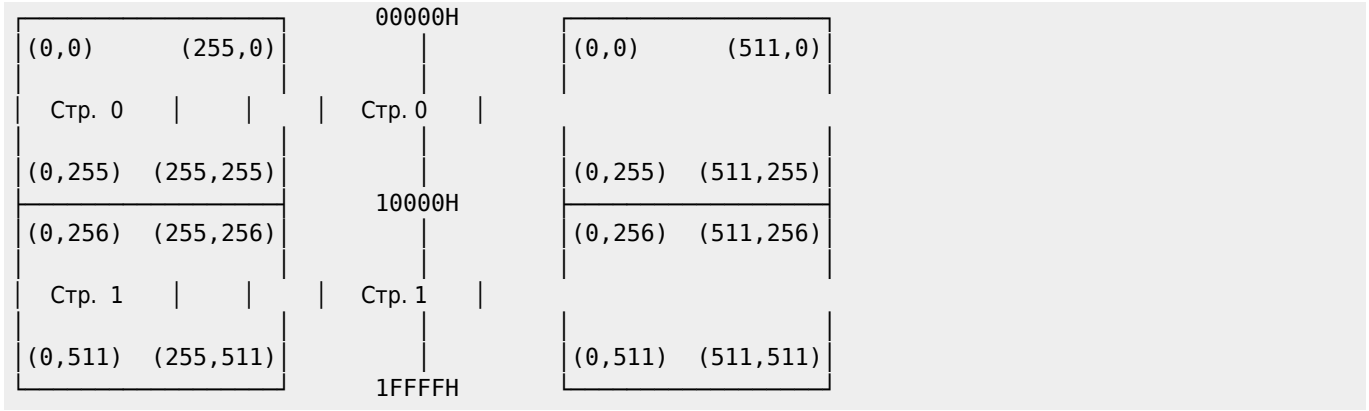


Рис.19.1. Координатная система VRAM

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 путём установки первого бита регистра #8. Можно также отключать при начальной загрузке изображение на экране.

Приведём 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
```

2.20. 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.

Приведём 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
```

Ещё 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 приведём 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
```

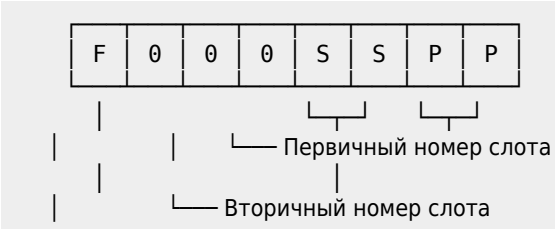
```
004A' D3 A1       OUT   (0A1h),a
004C' 3E 01       LD    a,1         ; запись 1  => 1
004E' D3 A0       OUT   (0A0h),a
0050' D3 A1       OUT   (0A1h),a
0052' F7          RST   30h         ; BEEP, чистка регистров
0053' 00          DB    0
0054' 00C0        DW    0C0h
0056' C9          RET
  ; === 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
```

2.21. 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: ```



┌──────┴────── │ 0 - вторичный слот не используется │ 1 - используется вторичный слот ```

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.

2.21.1. Работа с картриджами

Компьютер MSX обычно имеет по крайней мере один внешний слот. Та аппаратура (hardware), которая к нему подключается, называется картриджем (cartridge). Существуют картриджи 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, размещённые в соответствующей странице памяти. Информация кодируется побайтно следующим образом:

```
Бит: 7  6  5  4  3  2  1  0
     ┌──┬──┬──┬──┬──┬──┬──┬──┐
     │ x│ x│ x│ 0│ 0│ 0│ 0│ 0│
     └──┴──┴──┴──┴──┴──┴──┴──┘
       └┬┘ └┬┘ └┬┘
   │   │   └─── Разрешение обработки  CALL
       │    │
   │   └──── Разрешение работы с нестандартным  I/O
       │
   └─────── Текст на языке  BASIC
```

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 BASIC 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, приведён ниже.

```
4000h или 8000h+0000h ┌──────────┐
                      │    ID    │
                +0002h ├──────────┤
                      │   INIT   │
                +0004h ├──────────┤
                      │STATEMENT │
                +0006h ├──────────┤
                      │  DEVICE  │
                +0008h ├──────────┤
                      │   TEXT   │
                +000Ah ├──────────┤
            │ резерв │            │
                +0010h └──────────┘
```

- `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 Z80 `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 MSX 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 <инструкция> [(<параметр>[,<параметр>])]
  ```

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

  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 MSX BASIC. Содержимое HL должно быть возвращено неизменённым.

  В этом случае интерпретатор языка MSX BASIC пытается вызвать другой слот расширения. Если ни один слот «не отзовётся», 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 четырёх логических имён устройств.

  Когда MSX BASIC встречает имя устройства (например, OPEN«OPT:»...), то он записывает его в PROCNM (FD89h), код FFh - в аккумулятор и передаёт 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 MSX BASIC, если эта программа MSX BASIC в картридже должна автоматически запускаться при перезагрузке. В противном случае там хранится ноль. Размер программы должен быть не более 16 Кбайт, по адресам с 8000h по BFFFh.

Интерпретатор языка MSX 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 программа на MSX BASIC. Она должна храниться в промежуточном коде и её начало обозначается кодом ноль.

2.21.2. Создание CALL-подпрограмм пользователем

Приведём 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.

После чтения значений из ячейки управления вторичными слотами (FFFFh) они должны инвертироваться. Это нужно помнить и при запоминания 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 псевдо-ROM
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
```

2.22. 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.

2.22.1. Абсолютное чтение/запись

Под абсолютным чтением/записью здесь понимается чтение/запись логических секторов диска, в том числе загрузочного 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 приведём 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 ждёт ввода программистом первой литеры стёртого файла. В конце работы восстановленный директорий записывается назад на диск. Обратите внимание, что программа не восстанавливает 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
```

2.22.2. 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, MSX 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:

```
┌──────────────────────────────────────────────┐
│ F0 D2 C9 D7 C5 D4 20 21 0A 0D FC D4 CF 20 D1 2E 0A 0D 1A │
└──────────────────────────────────────────────┘
 П р и в е т │ ! └┘ Э т о │ я . └┘ │
                  │      │          │         │   └──┘
```

```
        пробел  новая    пробел   новая  конец
             строка          строка  файла
```

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 формате Intel.

MSX 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 её 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 размещён 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 (MSX Disk 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 MSX 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 её запуске дважды произойдёт 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
```

2.23. Ошибки программирования 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 найдёте причину плохой работы программы.

Другая часто встречающаяся ошибка — загрузка адреса вместо загрузки значения (опускание скобок). Вы должны чётко 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, её 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).

2.23.1. 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 директории 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.

2.23.2. 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 определённое время после заражения.

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

Три основных типа вирусов, разработанных в настоящее время для системы MSX — это:

1. Загрузочные вирусы;
2. Вирусы MSX-DOS;
3. 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 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 определённой 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 определённой программы при выполнении процедуры INIT), подготавливается конфигурация DISK-BASIC и управление передаётся на H.STKE.
3. Если же содержимое H.STKE равно C9h, во всех слотах ищется кассета со входом TEXT. В случае её нахождения подготавливается конфигурация MSX Disk BASIC и выполняется программа MSX BASIC из этой кассеты.
4. Затем содержимое загрузочного сектора диска (логический сектор #0) передаётся 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 MSX 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 ёмкость ОЗУ (его содержимое при этом не разрушается). Если она менее 64 Кбайт, вызывается MSX Disk BASIC.
7. 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 передаётся 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 MSX Disk BASIC.
8. 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. 9. При первой загрузке MSX Disk BASIC, если существует файл с именем `AUTOEXEC.BAT`, он выполняется как обычный пакетный файл. Когда MSX-DOS не запущен и работает MSX Disk BASIC, и если на диске имеется файл `AUTOEXEC.BAS`, то он будет автоматически запущен.

Загрузочные вирусы

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

2.23.3. 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.

2.23.4. 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 (приобретённое) 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 защищённая 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 своём 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 ведётся 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 включённом кем-то компьютере MSX, обязательно выключите его ненадолго и затем произведите загрузку со своей дискеты. Компьютер мог быть умышленно или случайно заражён вирусом, а его выключение, к счастью, уничтожает все вирусы.
- Если Вы поняли, что диск заражён вирусом, выключите компьютер, загрузите «чистую» систему с эталонного диска MSX-DOS или ND, перепишите файлы с заражённой дискеты, за исключением файлов типа COM, SYS, OBJ, GM (а лучше всего переписывать только текстовые файлы), на чистую дискету, отформатируйте и отверифицируйте заражённую дискету и восстановите на ней файлы.

---

https://sysadminmosaic.ru/msx/assembler_programming_guide-fakhrutdinov_bocharov/02

2022-09-09 22:17