MSX2: Техника программирования

Fix Me!

Олег Шамшура 1989-1990

Исходные файлы

Любой микропроцессор (в том числе Z80, на котором строится компьютер MSX/MSX2), имеет достаточно развитую систему команд, предписывающих ему то или иное действие. К числу таких команд обычно относятся: сложение, вычитание, условные и безусловные переходы, работа с памятью, логические операции над битами и пр. Фактически, команда — это определенный числовой код, который активизирует соответствующие счетно-логические схемы микропроцессора.

Последовательность таких кодов, совместно с числовыми и символьными операндами, образует программу. Несмотря на примитивный характер команд, с их помощью можно построить алгоритм любой сложности (например, интерпретатор языка BASIC). Нужно только хорошо помнить несколько сотен числовых кодов и соответствующих им операций.

Разумеется, эта непосильная на первый взгляд задача решена давно и просто: весь набор команд разбит на группы, которым присвоены буквенные обозначения (МНЕМОНИКА), происходящие от сокращенного наименования выполняемых операций.

Программа разрабатывается в мнемоническом коде, который затем превращается в естественный числовой формат при помощи специальной программы, именуемой АССЕМБЛЕРом. Это же название закрепилось за всей методикой программирования с использованием мнемонических обозначений. В простейшем случае роль ассемблера сводится к тому, чтобы распознавать эти обозначения и заменять их соответствующими числовыми комбинациями.

Таким образом, процесс разбивается на несколько последовательных этапов:

- исходный текст набирается в любом экранном РЕДАКТОРЕ;
- затем АССЕМБЛЕР превращает его в готовую к исполнению программу или промежуточный REL–файл (это зависит от версии ассемблера). Во втором случае REL–файлы, имеющие отношение к одному проекту, сцепляются воедино при помощи КОМПОНОВЩИКА. (Такой подход удобен при создании крупных проектов, содержащих десятки тысяч команд: он позволяет разрабатывать программу по частям, не затрагивая уже завершенных фрагментов). В процессе ассемблирования/компоновки выявляются грубые синтаксические ошибки;
- полученная программа тестируется в ОТЛАДЧИКЕ или непосредственно, и при необходимости весь процесс повторяется.

В отличие от MSX BASIC–интерпретатора, который «видит» ошибки только в том операторе, который исполняется в текущий момент, ассемблер обрабатывает весь текст программы целиком, включая и те ветви алгоритма, которые, возможно, никогда не будут пройдены микропроцессором. Поэтому исходный текст должен быть закончен хотя бы формально: всем переходам должны соответствовать метки, незавершенные подпрограммы следует заменить «заглушками», все константы нужно задать и т.д.

Поскольку компьютер представляет собой сложное сочетание разнообразных внутренних и периферийных устройств, одного знания мнемоники оказывается мало. Необходимо иметь представление о многих важных вещах: как переключать банки памяти, как выводить тексты и графику на экран или принтер, как обмениваться данными с диском, генерировать звук, опрашивать клавиатуру, джойстик, мышь и т.д.

Инструментарий

Половина успеха кроется в удачном подборе инструментов. Несколько замечаний по этому поводу.

Во—первых, Вам потребуется Дисковая Операционная Система, которая должна облегчить Вам работу с дисковыми файлами, а также обеспечить переносимость программ с одной модели ЭВМ на другую.

Особого столпотворения дисковых операционных систем для компьютеров MSX, к счастью, не наблюдается. В комплект обычно входят две системы: CP/M-80 и MSX-DOS, но безусловным лидером является, конечно, последняя, поскольку она:

- входит в стандарт MSX/MSX2;
- зашита фирмой непосредственно в машину;
- лежит в основе файловой подсистемы MSX Disk BASIC;
- совместима на уровне вызовов с самой распространенной DOS для 8–битовых машин — CP/M;
- имеет тот же формат файлов и командный интерфейс, что
- и популярнейшая система для компьютеров IBM PC — MS DOS.

Файлы `MSXDOS.SYS` и `COMMAND.COM` находятся на одноименном диске, входящем в комплект Вашего компьютера.

Теперь об ассемблере и других инструментах.

Чтобы постичь основы, достаточно приобрести пакет DUAD, который содержит в себе все необходимое.

Проекты средней сложности наилучшим образом реализуются в среде пакета DevPac80, в состав которого входят:

ED80.COM	текстовый редактор
GEN80.COM	быстрый ассемблер
MON80.COM	экранный отладчик

Достоинства пакета — очень высокая скорость ассемблера GEN80 и вразумительная диагностика ошибок.

Профессиональный набор инструментов, по всеобщему признанию, включает в себя (как минимум):

TOR.COM (или XTOR.COM)	текстовый редактор
M80.COM	макроассемблер
L80.COM (или LINK.COM)	компоновщик
LIB80.COM	библиотекарь
DBG.COM (или XDBG.COM)	экранный отладчик

Высокая мощность такого «сборного» пакета позволяет решать очень серьезные проблемы. Немаловажно и то, что все его составляющие совместимы с компилятором MSX-C.

Заметим, что в среде MSX-DOS все инструменты заменяемы и взаимозаменяемы. В принципе, можно использовать любой другой редактор (SCED, MIM), ассемблер (ASM, Z80), отладчик (SBUG, ZSID, DDT), но с меньшим комфортом для себя.

MSX-DOS: минимальные сведения

Любая работа начинается с запуска DOS: для этого достаточно вставить диск с файлами `MSXDOS.SYS` и `COMMAND.COM` в дисковод «A» и включить питание компьютера.

Первым с диска считывается файл `MSXDOS.SYS`, который реорганизует память компьютера, предоставляя в Ваше распоряжение 64K ОЗУ. Затем загружается файл `COMMAND.COM`, который будет принимать с клавиатуры команды DOS, разбирать их и обращаться к MSX-DOS, зашитой в постоянную память компьютера.

Индикатором успешного запуска системы служит «приглашение» `A>`. Буква перед угловой скобкой обозначает текущий дисковод.

В большинстве современных дисковых операционных систем, как и в MSX-DOS, имя файла имеет следующий вид: код дисковода (латинская буква с двоеточием), собственно имя (до 8 символов максимум), точка и расширение (до 3 символов).

В имя и расширение можно включать символы-джокеры «*» и «?». Звездочка «*» обозначает любую группу символов, а «?» — любой одиночный символ. Их используют, если нужно одним именем охватить несколько файлов одновременно.

Иногда некоторые элементы имени можно пропускать.

Примеры:

.	все файлы текущего дисковода
PROG.1	файл PROG.1 текущего дисковода
B:PROG.1	файл PROG.1 дисковода B:
C:PROG.*	все файлы с именем PROG дисковода C:
*.TXT	все файлы текущего дисковода с расширением TXT
???.*	все файлы, имя которых содержит 3 символа

Файлами могут быть: программы, массивы данных, дисковые подкаталоги или обычные тексты.

Формат программного файла продиктован особенностями той операционной системы, в которой создавалась программа. Обычно такой файл содержит либо последовательность команд микропроцессора, либо условные коды высокоуровневых операторов.

Структура файла данных полностью зависит от построившей его программы: блоки данных могут разделяться условными байтами–ограничителями (например, запятой «,»), или иметь заранее известную длину (тогда необходимость в разграничении отпадает). Наиболее изощренные программы могут работать с файлом как с нерегулярной цепочкой битов.

В файле может также храниться подкаталог диска, который организуется наряду с основным каталогом для удобной классификации файлов. v1, из соображений совместимости, имеет кое-какие рудиментарные возможности для организации подкаталогов, хотя они и остаются невостребованными. Структура подкаталога полностью повторяет структуру основного (корневого) каталога.

Наконец, файл может содержать какой-либо текст. Строки текста хранятся в своем естественном виде; каждая строка заканчивается байтами 13 и 10, которые названы «возврат каретки» и «подача строки» соответственно.

Признаком окончания текстового файла служит код CTRL Z, или, как это принято обозначать, ^Z, десятичное значение которого равно 26.

v1 способна исполнять три вида команд: внутренние, внешние и пакетные.

ВНУТРЕННИЕ команды содержатся непосредственно в файле COMMAND.COM, поэтому исполняются немедленно. Самые основные внутренние команды:

DIR	просмотреть оглавление диска
TYPE	распечатать содержимое файла
COPY	скопировать файл
FORMAT	отформатировать диск
BASIC	передать управление MSX BASIC (для возврата в DOS наберите _SYSTEM)
DEL	уничтожить файл
B:	считать текущим дисковод B:
A:	считать текущим дисковод A:

Параметры команд разделяются пробелом.

Действие любой команды можно приостановить клавишей CTRL s, прервать клавишей CTRL c. Подачей кода ^P можно включить эхо-печать на принтер, ^N — выключить.

Примеры: DIR DIR *.REL TYPE PROG.TXT

COPY B:PROG.1 C:	с B: на C:
COPY B:PROG.*	с B: на текущий (несколько файлов)
COPY *.TXT B:*.DOC	с текущего на B: (смена расширения)

ПРИМЕЧАНИЕ. COPY может сцепить несколько файлов в один, если между их именами указан знак '+'. Для правильной сцепки текстовых файлов необходимо после каждого имени ставить ключ /A. Прочие файлы должны сцепляться по ключу /B. (Ключ управляет интерпретацией кода ^Z).

Некоторые имена файлов зарезервированы и имеют неизменное значение, даже если попытаться снабдить их расширениями или кодами дисководов:

CON	консоль (клавиатура при вводе, экран при выводе)
LST и PRN	принтер (только вывод)
AUX и NUL	несуществующее устройство (всегда возвращает признак конца файла, удобно при отладке)

Это означает, что в среде v1 с дисплеем, принтером или клавиатурой можно работать, как с обычным файлом:
COPY PROG.TXT PRN — распечатать файл PROG.TXT на принтере

Можно, например, набирать тексты безо всяких редакторов (после каждой строки нажимайте клавишу RETURN):

COPY CON A.BAT	копировать с клавиатуры в файл A.BAT
DIR	собственно текст
^Z	признак конца текстового файла CTRL Z

Под ВНЕШНЕЙ командой v1 понимается любой файл, который имеет расширение COM. Вызов внешней команды аналогичен запуску внутренней: нужно набрать имя файла (без «.COM») и через пробел перечислить параметры. Вызванная команда загружается с диска в память и получает управление, черпая параметры из специальной области. По окончании работы управление возвращается MSXDOS.SYS.

Осталось познакомиться с ПАКЕТНЫМИ командами — наиболее приятной сервисной возможностью DOS. Пакетная команда — это обычный текстовый файл, в каждой строке которого указана некоторая внутренняя, внешняя или пакетная команда. Основным отличием такого файла является расширение BAT.

Пакетная команда представляет собой своеобразную «программу», которая экономит Ваши усилия при многократном вызове одной и той же последовательности команд. Пример BAT-файла, содержащего единственную команду DIR, приведен выше.

Возможности пакетных команд обогащаются наличием команд REM (вывод комментария) и PAUSE (приостановка исполнения).

Естественно, в пакетные команды тоже можно передавать параметры. Положение первого параметра отмечается в файле значком %1, второго — %2, ..., девятого — %9. Имя самого BAT—файла соответствует значку %0.

Ассемблеры: минимальные сведения

```
   Все программы пакета DevPac80, а также SBUG, SCED, TOR,
DBG и множество других хорошо вписываются в перечисленные
правила вызова. Например, ассемблирование исходного текста
PROG.ASM с получением готовой программы PROG.COM в среде
пакета DevPac80 выглядит так:

A>GEN80 PROG.ASM

   Вызов текстового редактора с одновременной загрузкой
нескольких файлов можно осуществить, например, так:

A>TOR FILE1.ASM FILE2.ASM FILE3.ASM

   или так, что то же самое:

A>TOR FILE?.ASM

   Однако существуют и исключения. Командные строки для
L80, LINK, LIB80 и некоторых других программ вынужденно изо-
билуют ключами, опциями и разделителями. Компоновка промежу-
точных файлов в готовую программу может выглядеть, например,
так:
```

```
A>L80 /P100,/DC000, FILE1,FILE2,FILE3/S,FILE/N/Y/E:BEGIN

   или так:

A>LINK FILE[GBEGIN,DC000,P100]=FILE1,FILE2,FILE3[S]

   В обоих случаях:
      - FILE1, FILE2 и FILE3 - исходные файлы в промежуточном
формате (у всех по умолчанию расширение REL);
      - FILE - имя программы (по умолчанию расширение COM);
      - BEGIN - глобальная метка внутри программы, с которой
должно начинаться выполнение;
   - 100 - адрес тела программы (в 16-ричном виде);
      - C000 - адрес области данных (в 16-ричном виде);
      - /P или [P...] - установка адреса тела программы (по
умолчанию 100H);
      - /D или [D...] - установка адреса области данных (по
умолчанию соответствует адресу окончания программы);
      - /S или [S] - обозначает библиотечный файл (особая
организация REL-файла, позволяющая выбирать из него только
те модули, которые нужны в данной ситуации. Формируется
программой LIB80 из модулей в REL-формате. Командная строка
со знаком =, завершается ключом /E. Непременное условие: имя
вызывающего модуля должно стоять левее имени вызываемого);
      - /N - обозначает, что данное имя следует присвоить
готовой программе;
      - /Y - разрешает создание на диске файла, содержащего
список глобальных меток с их адресами, очень полезного при
отладке программы в DBG.COM (XDBG.COM);
      - /E - завершает работу программы L80.COM.

   На первый взгляд - сложно. Однако, если Ваша программа
умещается целиком в едином текстовом файле, процесс превра-
щения PROG.ASM в PROG.COM усложняется минимально:

A>M80 =PROG.ASM (результат - файл PROG.REL)
A>LINK PROG     (результат - файл PROG.COM)
A>DEL PROG.REL  (удалить файл PROG.REL за ненадобностью)

   Кроме того, если вовремя вспомнить о пакетных файлах,
то можно вообще избавиться от нужды постоянно набирать эту
последовательность (это касается любых, в том числе и очень
длинных команд). Файл с именем, допустим, M.BAT и следующим
содержанием:

     M80 =%1.ASM
     LINK %1
     DEL %1.REL

позволит Вам обходиться единственной строчкой:

A>M PROG
```

Системы счисления

```
   В программировании абсолютно равноправны три системы
счисления: двоичная, десятичная и шестнадцатиричная. Первая
из них, как известно, органически присуща компьютеру, вторая
привычна нам, а третья представляет собой своеобразный ком-
промисс между программистом и машиной (иногда вместо нее ис-
пользуют восьмиричную систему).
   Различие между ними кроется в так называемом ОСНОВАНИИ
```

системы, а проще говоря, в том, какое число принять за "10".
В двоичной системе для изображения чисел достаточно только двух цифр: 0 и 1 (поскольку 2 - это 10, 3 - это 11 и т.д.).
В шестнадцатиричной системе символом 10 обозначено число 16, поэтому требуется 16 цифр: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Чтобы избежать недоразумений, принято к двоичному числу приписывать символ "b", а к шестнадцатиричному - символ "h".

В связи с наличием нескольких числовых систем возникает вопрос о переводе чисел из одной системы в другую. Это делается очень просто. Чтобы перевести десятичное число в другую систему, нужно последовательно делить его на основание новой системы, записывая остатки от деления справа налево.

Обратный перевод выполняется так: самый старший разряд числа умножается на основание системы и суммируется с разрядом, соседним справа. Эта сумма вновь умножается на основание, вновь суммируется с соседним разрядом и т.д.

```
Задача: 13 = ?b

    13 / 2 = 6                    1b
         6 / 2 = 3                   01b
             3 / 2 = 1              101b
                 1 / 2 = 0         1101b
Ответ: 13 = 1101b

Задача: 1101b = ?

    (1 * 2) + 1 = 3
              (3 * 2) + 0 = 6
                   (6 * 2) + 1 = 13
Ответ: 1101b = 13
```

Взаимный перевод чисел из 16-ричной системы в двоичную и обратно выглядит еще элегантнее: достаточно заметить соответствие между 16-ричными цифрами и четверками битов.

```
0h = 0000b  4h = 0100b    8h = 1000b    Ch = 1100b
1h = 0001b  5h = 0101b    9h = 1001b    Dh = 1101b
2h = 0010b  6h = 0110b    Ah = 1010b    Eh = 1110b
3h = 0011b  7h = 0111b    Bh = 1011b    Fh = 1111b
```

Таким образом, перевод из двоичной системы в 16-ричную сводится к резке числа на четверки битов и "переименовании" каждой четверки в 16-ричную цифру, что без труда выполняется в уме:

```
     101011b = 10 1011 = 2 B = 2Bh
```

Обратный перевод вообще тривиален: каждая цифра заменяется соответствующей четверкой битов.

```
     F63Ah = 1111 0110 0011 1010 = 1111011000111010b
```

Машинная арифметика

Теперь нетрудно установить, что БАЙТ (единица измерения информации, равная 8 битам) способен представлять числа:

- от 00000000b до 11111111b
- от 00h до FFh
- от 0 до 255

Кроме того, несложно определить весь числовой диапазон, доступный микропроцессору, зная, что микропроцессор способен оперировать двухбайтовыми величинами (СЛОВАМИ):

- от 0000000000000000b до 1111111111111111b
- от 0000h до FFFFh
- от 0 до 65535

Поначалу может показаться, что этого явно недостаточно. Однако не стоит забывать о специфике системных программ, которые чаще всего имеют дело с адресами памяти, кодами символов, экранными координатами и прочими объектами, для нумерации которых вполне достаточно байта или слова. При этом за Вами всегда остается право хранить большие числа в нескольких байтах, обрабатывая их по частям.

Другая особенность машинной арифметики Z80 - это отсутствие команд умножения, деления, возведения в степень и т.д. Зато в Вашем распоряжении остаются вычитание, сложение, увеличение и уменьшение на 1, разнообразные сдвиги битов и логические операции, аргументами которых служат разряды двоичных чисел:

```
0101b AND 0011b = 0001b
0101b OR  0011b = 0111b
0101b XOR 0011b = 0110b
      CPL 0011b = 1100b
```

Все недостающие операции можно построить из названных.

Наконец, одна из важнейших черт арифметики микропроцессора, учитывать которую необходимо в обязательном порядке, заключается в следующем: МИКРОПРОЦЕССОР НЕ ОБРАЩАЕТ ВНИМАНИЯ НА ПЕРЕПОЛНЕНИЕ РАЗРЯДНОЙ СЕТКИ. Если результат вычислений не укладывается в числовой диапазон, его старшие разряды попросту отбрасываются.

Это, казалось бы, незначительное явление имеет далеко идущие последствия. Судите сами.

Суммируя два однобайтовых числа, FFh и 01h, мы получаем вместо 100h нуль. К этому результату можно отнестись двояко:

- считать это досадным ограничением;
- допустить нетривиальную мысль, что FFh = -1.

Второй вывод открывает интересные перспективы, особенно если распространить его на все однобайтовые числа:

```
256 = 0, 255 = -1, 254 = -2, ..., 1 = -255, 0 = -256
```

С одной стороны, мы приобретаем простой и естественный аппарат отрицательных чисел, но, с другой стороны, полностью теряем возможность интерпретировать результаты вычислений, поскольку положительные и отрицательные числа в таком виде ничем не отличаются друг от друга.

Чтобы устранить эту неоднозначность, числовой диапазон принято делить пополам, приписывая первой его половине положительное значение, а второй - отрицательное:

```
0, 1, 2, ..., 126, 127, -128, -127, -126, ..., -2, -1
```

Поступив так, мы получаем в руки достаточно осязаемый признак - старший бит, который равен 0 для первой группы чисел и 1 для второй. Этот бит, естественно, называется "знаковым".

Теперь все зависит от программиста. Если он в своих

выкладках учитывает знаковый бит, значит, он выбрал диапазон -128...127. Если же знаковый бит игнорируется, значит выбран "классический" диапазон 0...255. Микропроцессору же, как Вы сами понимаете, это глубочайше безразлично.

Знаковый бит позволяет уверенно различать ситуации типа

126 + 129 = 255 и 126 + (-127) = -1

но, к сожалению, на этом его сходство с математическим знаком числа заканчивается. Простейшая из операций - унарный минус - не может быть выполнена путем инверсии знакового бита. Анализ расположения чисел подсказывает: чтобы сменить знак, нужно инвертировать ВСЕ биты числа и к результату прибавить 1. (Кстати, рекомендуем проделать эту процедуру с 0 и -128. Что получится ?)

С двухбайтовыми величинами дело обстоит аналогично, за малым исключением: роль "минус единицы" играет, естественно, 65535, а "водораздел" проходит между числами 32767 и -32768.

Это скромное различие проявляет себя, когда требуется превратить однобайтовую величину в слово. Для чисел 0...255 проблем нет: их помещают в младший байт слова, а старший байт обнуляют.

Диапазон -128...127 требует более тонкой процедуры, называемой "распространением знака", суть которой сводится к следующему: число также помещается в младший байт слова, но каждый разряд старшего байта приравнивается к знаковому биту исходного числа. Очевидно, только так удается обеспечить соответствие между отрицательными байтами и словами.

К счастью, обратное превращение можно произвести только одним способом - отбросив старший байт слова.

Кстати, за всеми этими рассуждениями не стоит забывать и о более прозаических последствиях переполнения, искажающих ожидаемый результат, и использовать в вычислениях величины с заведомо достаточной разрядностью.

BCD–арифметика

Неистребимая приверженность человечества к десятичной системе счисления породила странный гибрид — BCD–арифметику, которая почти полностью аналогична шестнадцатиричной, но за малым исключением: в ней наложен запрет на цифры A,B,...,F.

Так, например, операция 88h + 04h дает вместо 8Ch число 92h. Сложение же чисел 99h и 01h приводит к результату 100h, от которого остается 00h плюс все прежние выкладки.

Микропроцессор умеет оперировать BCD—числами; для этого в системе команд предусмотрена десятичная коррекция результата, которая должна выполняться вслед за сложением и вычитанием (операции уменьшения и увеличения на 1 коррекции НЕ поддаются!):

88h + 04h → 8Ch	обычное сложение
8Ch → 92h	десятичная коррекция

Следует понимать, что коррекция отнюдь не выполняет перевода из 16—ричной системы в десятичную. Если операнды содержат «запрещенные» цифры (A...F), результат коррекции оказывается непредсказуемым.

Многоразрядная арифметика

Чтобы облегчить обработку многоразрядных чисел, введено понятие БИТА ПЕРЕНОСА. Бит переноса устанавливается в 1, если в процессе вычислений требуется перенести единицу в следующий байт, или «занять» ее там.

Наличие команд сложения и вычитания, которые учитывают текущее состояние бита переноса, позволяет вести вычисление байт за байтом, оперируя величинами любой разрядности.

Микропроцессор Z80

Микропроцессор Z80 пользуется заслуженной популярностью среди изготовителей персональных микроЭВМ. Известнейшие модели SINCLAIR (ZX-81, SPECTRUM), SCHNEIDER (CPC 464, JOYCE), COMMODORE (128PC), а также многочисленные образцы MSX и MSX2 построены именно на этом процессоре.

Относительно невысокое быстродействие (при стандартной для MSX тактовой частоте 3.75 MHz около 800 000 опер/с) окупается богатым набором инструкций, а полная совместимость с не менее популярным микропроцессором 8080 значительно расширяет круг доступных программных средств.

С точки зрения программиста, процессор Z80 представляет собой совокупность нескольких регистров. Каждый регистр имеет свое стандартное название и назначение:

ОДНОБАЙТОВЫЕ регистры:

A - аккумулятор
B,C,D,E,H,L - вспомогательные регистры
F - регистр признаков результата
I - регистр, содержащий информацию о прерываниях
R - регистр, связанный с регенерацией динамического ОЗУ

ДВУХБАЙТОВЫЕ регистры:

PC - счетчик команд
SP - указатель стека
IX, IY - регистры индексирования
AF',BC',DE',HL' - альтернативный комплект регистров

Некоторые из команд трактуют регистры A и F как единое целое - слово AF. Аналогичным образом регистры B и C, D и E, H и L могут объединяться в BC, DE и HL. Это значит, что Z80 может "за один прием" обрабатывать 16-битовые данные.

В счетчике PC находится номер (АДРЕС) ячейки памяти, которая содержит код очередной команды. Z80 распознает этот код и выполняет требуемые действия, после чего значение PC увеличивается на длину команды, и цикл повторяется.

Поскольку PC является 16-битовым регистром, диапазон доступных процессору адресов памяти составляет 0...65535, то есть 64 Килобайта (1 Килобайт = 1024 байтам).

Встречая команду перехода, процессор помещает указанный в ней адрес в регистр PC, и исполнение продолжается с новой точки. (В ряде случаев значение PC изменяется на некоторую величину в пределах -128...127, что сокращает команду на 1 байт).

Регистр SP служит для организации СТЕКА - особой формы управления памятью, позволяющей записывать данные в память в одном порядке, а считывать их в обратном. Запись и чтение

происходят по адресу, который указан в SP, но перед записью SP уменьшается на 2, а после чтения вновь увеличивается.

Стек позволяет организовать работу с подпрограммами: по команде вызова текущее значение PC сохраняется в стеке, а PC получает новое значение; по команде возврата из подпрограммы состояние PC восстанавливается.

Поскольку стек, заполняясь, растет в сторону уменьшения адресов, SP должен быть настроен на верхние адреса памяти.

Наряду с прямым доступом к памяти, когда номер ячейки указывается как фиксированное число, возможны еще два:

"косвенный", при котором доступ производится по адресу, хранящемуся в регистре BC, DE или HL;

"индексирование", когда задается смещение относительно адреса, хранящегося в регистре IX или IY. Этот вид адресации удобен для обработки однотипных структур.

Регистры I и R связаны с довольно тонкими процессами в схеме ЭВМ, которые мы не будем затрагивать. Совет - ничего в эти регистры не записывать. Впрочем, регистру R все же можно отыскать применение - из него получается приличный генератор случайных чисел с равномерным распределением от 0 до 127.

Регистры A,B,C,D,E,F,H,L существуют в двух комплектах, и в оба комплекта входят так называемые "аккумулятор" и "регистр признаков".

С аккумулятором связано наибольшее количество операций. Регистр признаков содержит биты, характеризующие результат вычислений: SZ.H.PNC

- S - получен отрицательный результат
- Z - получен нулевой результат
- H - был перенос в старший полубайт (или заем)
- P - получен нечетный результат
- C - был перенос в следующий байт (или заем)

Бит C (carry flag) играет особую роль, поэтому им можно управлять отдельно. Среди 16-битовых регистровых пар функции аккумулятора выполняет HL.

В момент включения питания или нажатия RESET состояние процессора, памяти и внешних устройств абсолютно непредсказуемо. Гарантируется лишь одно: регистр PC в этот момент обнуляется. Поэтому в памяти компьютера, начиная с адреса 0000h, должна располагаться программа инициализации, которая занимается глобальной настройкой всей системы.

Существует особый способ вызова подпрограмм, называемый прерыванием. Заключается он в следующем: внешние устройства могут посылать запрос микропроцессору, который заставляет его приостановить работу и перейти к подпрограмме обработки ситуации, вызвавшей запрос.

Прерывания избавляют от необходимости постоянно следить за состоянием периферии и дают возможность сосредоточиться на основном алгоритме. Поскольку прерывание может произойти в самый непредсказуемый момент, подпрограмма обработки должна позаботиться о сохранении текущего значения всех регистров в стеке, обязательно восстановив их перед возвратом.

Если алгоритм критичен по времени, или процесс обмена с периферией не допускает вмешательства, прерывания можно отключить.

В стандарте MSX/MSX2 подпрограмма обработки прерываний начинается с адреса 38h. Она сканирует клавиатуру, исполняет музыку, отсчитывает время и т.д.

"Внешний мир" для Z80 выглядит как совокупность ПОРТОВ ВВОДА/ВЫВОДА (пронумерованных устройств, способных принимать или возвращать определенные данные). В зависимости от номера конкретного порта и характера данных, периферия выполняет требуемые действия.

Система команд микропроцессора Z80

Примем обозначения:

n	однобайтовое число без знака (0...255)
nn	двухбайтовое число без знака (0...65535)
e	однобайтовое число со знаком (-128...127)
b	номер бита в байте (нумерация битов: 76543210)
f	любой из байтов: 00h 08h 10h 18h 20h 28h 30h 38h
r	любой из регистров: B C D E H L A
ii	любой из регистров: IX IY
rr	любой из регистров: BC DE HL SP
rx	любой из регистров: BC DE IX SP
ry	любой из регистров: BC DE IY SP
qq	любой из регистров: BC DE HL AF
cc	любой из признаков: NZ Z NC C PO PE P M
NZ	не ноль (Z=0)
PO	нечетный (P=1)
Z	ноль (Z=1)
PE	четный (P=0)
NC	не было переноса (C=0)
P	неотрицательный (S=0)
C	был перенос (C=1)
M	отрицательный (S=1)
cd	любой из признаков: NZ Z NC C

Круглые скобки означают косвенную адресацию. Сравните:

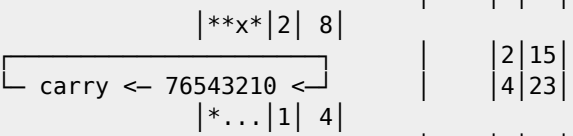
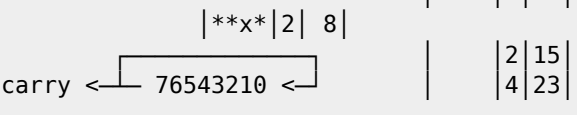
HL	значение регистра HL
(HL)	содержимое ячейки, на которую указывает HL

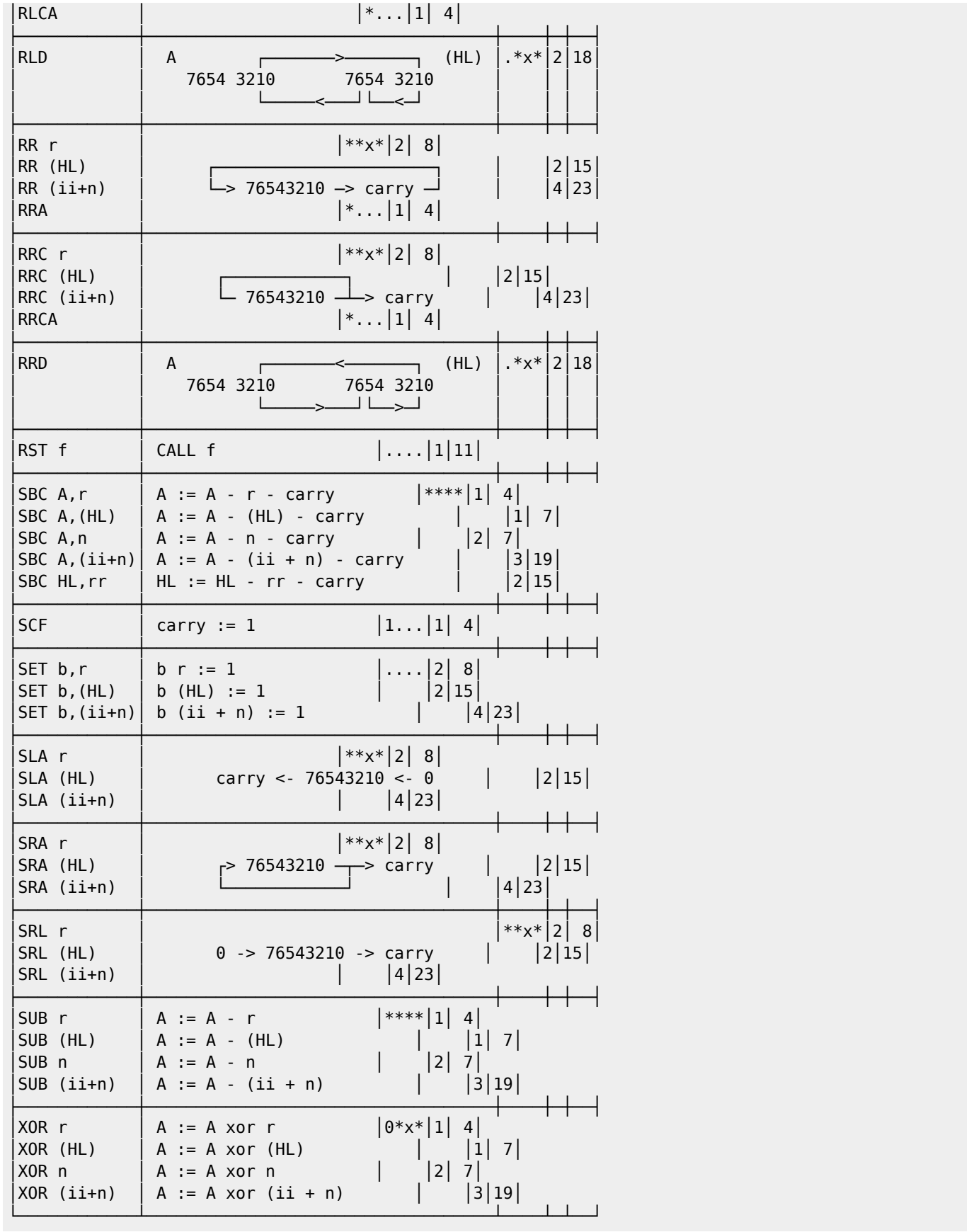
Кроме этого:

*	признак соответствует результату
0	признак сбрасывается в 0
1	признак устанавливается в 1
x	состояние признака непредсказуемо
.	состояние признака не изменяется
l	длина команды в байтах
t	время исполнения команды в тактах

мнемокод	действие	CZPS	l	t
ADC A,r	A := A + r + carry	****	1	4
ADC A,(HL)	A := A + (HL) + carry		1	7
ADC A,n	A := A + n + carry		2	7
ADC A,(ii+n)	A := A + (ii + n) + carry		3	19
ADC HL,rr	HL := HL + rr + carry		2	15
ADD A,r	A := A + r	****	1	4
ADD A,(HL)	A := A + (HL)		1	7
ADD A,n	A := A + n		2	7
ADD A,(ii+n)	A := A + (ii + n)		3	19
ADD HL,rr	HL := HL + rr	*...	1	11
ADD IX,rx	IX := IX + rx		2	15
ADD IY,ry	IY := IY + ry		2	15
AND r	A := A and r	0*x*	1	4
AND (HL)	A := A and (HL)		1	7
AND n	A := A and n		2	7
AND (ii+n)	A := A and (ii + n)		3	19
BIT b,r	Z := not b r	.*xx	2	8
BIT b,(HL)	Z := not b (HL)		2	12
BIT b,(ii+n)	Z := not b (ii + n)		4	20
CALL nn	PUSH PC; PC := nn	....	3	17
CALL cc,nn	если верно cc, то CALL nn		3	12
	иначе NOP			10
CCF	carry := not carry	*...	1	4
CP r	A - r (без присвоения результата)	****	1	4
CP (HL)	A - (HL)		1	7
CP n	A - n		2	7
CP (ii+n)	A - (ii + n)		3	19
CPD	A - (HL); DEC HL; DEC BC	.***	2	16
CPDR	повторять CPD	.***	2	21
	пока BC ≠ 0 и Z = 0			16
CPI	A - (Hl); INC HL; DEC BC	.***	2	16
CPIR	повторять CPI	.***	2	21
	пока BC ≠ 0 и Z = 0			16
CPL	A := not A	....	1	4
DAA	десятичная коррекция результата	**x*	1	4
DEC r	r := r - 1	.***	1	4
DEC (HL)	(HL) := (HL) - 1		1	11
DEC (ii+n)	(ii + n) := (ii + n) - 1		3	23
DEC rr	rr := rr - 1		1	6
DEC ii	ii := ii - 1		2	10
DI	запретить прерывания	....	1	4
DJNZ e	DEC B; если B≠0, то JR e	....	2	13
	иначе NOP			8
EI	разрешить прерывания	....	1	4

EX AF,AF'	AF <-> AF'	....	1	4
EX DE,HL	DE <-> HL		1	4
EX (SP),HL	(SP) <-> HL		1	4
EX (SP),ii	(SP) <-> ii		2	23
EXX	BC <-> BC'; DE <-> DE'; HL <-> HL'	....	1	4
HALT	стоп (до следующего прерывания)	....	1	4
IM 1	установка режима прерывания 1	....	2	8
IM 2	установка режима прерывания 2			
IM 3	установка режима прерывания 3			
IN A,(n)	A := данные из порта (n)	....	2	11
IN r,(C)	r := данные из порта (C)	.*x*	2	12
INC r	r := r + 1	.***	1	4
INC (HL)	(HL) := (HL) + 1		1	11
INC (ii+n)	(ii + n) := (ii + n) + 1		3	23
INC rr	rr := rr + 1		1	6
INC ii	ii := ii + 1		2	10
IND	(HL) := порт (C); DEC B; DEC HL	x*xx	2	16
INDR	повторять IND пока B ≠ 0	x1xx	2	21/16
INI	(HL) := порт (C); DEC B; INC HL	x*xx	2	16
INIR	повторять INI пока B ≠ 0	x1xx	2	21/16
JP nn	PC := nn	....	3	10
JP cc,nn	если верно cc, то JP nn иначе NOP		3	10
JP (HL)	PC := HL		1	4
JP (ii)	PC := ii		2	8
JR e	PC := PC + e	....	2	12
JR cd,e	если верно cd, то JR e иначе NOP		2	12/7
LD r,r	r := r	....	1	4
LD r,(HL)	r := (HL)		1	7
LD r,n	r := n		1	7
LD r,(ii+n)	r := (ii + n)		3	19
LD (HL),r	(HL) := r		1	7
LD (ii+n),r	(ii + n) := r		3	19
LD (HL),n	(HL) := n		2	10
LD (ii+n),n	(ii + n) := n		4	19
LD A,(BC)	A := (BC)		1	7
LD A,(DE)	A := (DE)		1	7
LD A,(nn)	A := (nn)		3	13
LD (BC),A	(BC) := A		1	7
LD (DE),A	(DE) := A		1	7
LD (nn),A	(nn) := A		3	13
LD A,I	P := 1, если прерывания разрешены	.***	2	9
LD A,R	A := R		2	9
LD I,A	I := A	....	2	9
LD R,A	R := A		2	9
LD rr,nn	rr := nn		3	10
LD ii,nn	ii := nn		4	14
LD HL,(nn)	HL := (nn)		3	16
LD rr,(nn)	rr := (nn)		4	20
LD ii,(nn)	ii := (nn)		4	20

LD (nn),HL	(nn) := HL		3	16
LD (nn),rr	(nn) := rr		4	20
LD (nn),ii	(nn) := ii		4	20
LD SP,HL	SP := HL		1	6
LD SP,ii	SP := ii		1	10
LDD	(DE):=(HL); DEC HL; DEC DE; DEC BC	..*.	2	16
LDDR	повторять LDD	..0.	2	21
	пока BC ≠ 0		16	
LDI	(DE):=(HL); INC HL; INC DE; DEC BC	..*.	2	16
LDIR	повторять INI	..0.	2	21
	пока BC ≠ 0		16	
NEG	A := -A	****	2	8
NOP	нет операции	....	1	4
OR r	A := A or r	0*x*	1	4
OR (HL)	A := A or (HL)		1	7
OR n	A := A or n		2	7
OR (ii+n)	A := A or (ii + n)		3	19
OUTD	порт (C) := (HL); DEC B; DEC HL	x*xx	2	16
OTDR	повторять OUTD	x1xx	2	21
	пока B ≠ 0		16	
OUTI	порт (C) := (HL); DEC B; INC HL	x*xx	2	16
OTIR	повторять OUTI	x1xx	2	21
	пока B ≠ 0		16	
OUT (n),A	порт (n) := A	....	2	11
OUT (C),r	порт (C) := r		2	12
POP qq	qq := (SP); SP := SP + 2	....	1	10
POP ii	ii := (SP); SP := SP + 2		2	14
PUSH qq	SP := SP - 2; (SP) := qq	....	1	11
PUSH ii	SP := SP - 2; (SP) := ii		2	15
RES b,r	b r := 0	....	2	8
RES b,(HL)	b (HL) := 0		2	15
RES b,(ii+n)	b (ii + n) := 0		4	23
RET	POP PC	....	1	10
RET cc	если верно cc, то RET		1	11
	иначе NOP		1	5
RETI	завершение обработки прерывания	....	2	14
RETN	завершение немаскированного прерыв	....	2	14
RL r		**x*	2	8
RL (HL)			2	15
RL (ii+n)			4	23
RLA		*...	1	4
RLC r		**x*	2	8
RLC (HL)			2	15
RLC (ii+n)			4	23



Коментарий
В таблице приведены: мнемоническое обозначение команды, выполняемое действие, его влияние на регистр F, длина командного кода и количество затрачиваемых тактов.

В принципе, мнемоника Z80 стандартизована, хотя некоторые из ассемблеров позволяют себе незначительные отклонения. Может оказаться, например, что ADD A,r и ADC A,r не требуют указания регистра A (ADD r, ADC r), а в IM 1…3 нумерация режимов начинается с нуля.

В системе команд нужно ориентироваться свободно. Особое внимание следует уделить длине команд и времени исполнения, ибо учет этих характеристик напрямую связан с компактностью и быстродействием программ.

Некоторых пояснений заслуживают команды группы CP, а также LDIR и LDDR.

Инструкция CP предназначена для сравнения аккумулятора с другим значением и установки признаков в регистре F: *

- бит Z = 1 означает, что числа равны
- бит Z = 0 означает, что числа не равны
- бит S = 1 означает, что аккумулятор меньше
- бит S = 0 означает, что аккумулятор больше или равен
- бит C = 1 означает, что аккумулятор меньше
- бит C = 0 означает, что аккумулятор больше или равен

Если Вы работаете в числовом диапазоне 0...255, то результат сравнения следует определять по биту C. Если Вы выбрали диапазон -128...127, то результат определяется битом S. Бит Z, естественно, подходит для любого случая.

Инструкции CPDR и CPIR удобны для поиска элемента в таблице. Если элемент найден, Z = 1, а его адрес равен HL-1.

Команды LDIR и LDDR занимаются перемещением фрагментов памяти. LDIR начинает с младшего адреса фрагмента, поэтому ее следует применять для сдвига в сторону меньших адресов, а LDDR — наоборот.

Ассемблер

```
   Текст программы пишется в три  колонки.  Левая  колонка
предназначена для МЕТОК, средняя - для мнемоники, а правая -
для операндов (данных, над которыми  выполняется  действие).
Колонки удобно отделять друг от друга нажатием клавиши  TAB.
   МЕТКА в ассемблере заменяет собой конкретный адрес  или
числовое значение и может состоять почти из любых  символов,
что позволяет придавать меткам  осмысленный  вид.  Благодаря
такой возможности, программа на ассемблере выглядит не менее
(а иногда и более) понятно, чем, например, на  BASICe.
   Чтобы текст программы не зависел от версии  ассемблера,
старайтесь придерживаться набора символов  A...Z 0...9 _ @   и
обеспечьте различие меток по первым 6  знакам.  Кроме  того,
обычно требуется, чтобы метка заканчивалась  двоеточием.  Не
стоит, полагаем, особо объяснять, что в качестве меток нель-
зя использовать ни мнемокоды, ни наименования регистров.
   Чтобы избежать путаницы между метками и 16-ричными чис-
лами, последние принято начинать с нуля: 0ABh, 0FFE7h  и т.д.
Если число начинается с цифры 1...9, незначащий  нуль  можно
не указывать.
   Кроме обычных чисел можно использовать  ASCII-коды алфа-
витно-цифровых символов. Для этого символ помещают  в  апос-
трофы или кавычки: 45h + 'Z' - 1110b.
   Запомните как следует: В ОЗУ МЛАДШИЙ БАЙТ СЛОВА  ВСЕГДА
ПРЕДШЕСТВУЕТ СТАРШЕМУ. В регистровых парах порядок обычный.

   В текст программы можно вводить произвольное количество
комментариев, которые обозначаются слева точкой с запятой ;.
Все, что находится правее точки с запятой, ассемблером игно-
рируется.
   Обычно программа должна заканчиваться словом  END.  Кроме
того, макроассемблер  M80,  умеющий работать с мнемоникой раз-
ных процессоров, требует указания в самом начале:

      .z80

   Существуют инструкции, которые входят в текст программы
наравне с мнемоникой, но относятся к процессу компиляции:
```

инструкция DEFB n резервирует в памяти один байт, которому присваивает начальное значение n;
инструкция DEFW nn резервирует одно слово, которому присваивает начальное значение nn;
инструкция DEFS nn резервирует в памяти место размером nn байт (без инициализации);
инструкция DEFM "text" задает в памяти строку символов;
инструкция EQU присваивает метке числовое значение;
инструкция DEFL (или SET) изменяет ранее заданное значение метки;
инструкция ORG nn объявляет, что программа начинается с адреса nn. В MSX-DOS этот адрес всегда равен 100h.
Инструкции DEFB и DEFW допускают перечисление через запятую нескольких данных: DEFB 0,7Bh,22,45,11b
Можно также получать адрес текущей команды с помощью псевдопеременной $.

В процессе компиляции ассемблер способен производить различные операции над метками (иногда достаточно сложные, с участием умножения, деления, экспоненты и т.д.). Их не следует путать с командами микропроцессора! Эти вычисления производятся с помощью сложных процедур, "вшитых" в ассемблер, причем только один раз, в период компиляции, после чего результат используется для формирования определенной команды.

Кроме перечисленных общеупотребительных моментов каждая версия ассемблера имеет свои возможности (например, условную компиляцию и макрообработку), но мы ограничимся рациональным минимумом сведений.

Примеры коротких программ

Рассмотрим простейший пример:

```
; ------------------------------------------------------
; программа вычисления выражения U = V + W - 1, версия 1
; ------------------------------------------------------
START:          ; начальная метка программы
    ld    a,(V)     ; читаем V в аккумулятор
    ld    b,a     ; передадим значение в B
    ld    a,(W)     ; читаем W в аккумулятор
    add    a,b     ; сложим A и B (W + V)
    dec    a     ; вычтем 1 (W + V - 1)
    ld    (U),a     ; запишем аккумулятор в U (U = ...)
    ret         ; работа завершена

V:  defb    2     ; V первоначально равен 2
W:  defb    3     ; W первоначально равен 3
U:  defs    1     ; U неопределен (один байт)
; ------------------------------------------------------
```

Ничто не мешает построить алгоритм по-другому:

```
; ------------------------------------------------------
; программа вычисления выражения U = V + W - 1, версия 2
; ------------------------------------------------------
START:  ld    hl,V     ; положить в HL адрес ячейки V
    ld    a,(hl)     ; взять V (по адресу HL)
    inc    hl     ; сдвинуть HL на ячейку W
    add    a,(hl)     ; сложить аккумулятор с W
    inc    hl     ; сдвинуть HL на ячейку U
    ld    (hl),a     ; записать сумму в ячейку U
    dec    (hl)     ; уменьшить U на 1
    ret
```

```
V:  defb    2
W:  defb    3
U:  defs    1
; ------------------------------------------------------
```

Наконец, для тренировки рассмотрим третий вариант:

```
; ------------------------------------------------------
; программа вычисления выражения  U = V + W - 1, версия 3
; ------------------------------------------------------
START:  ld    ix,V        ; загрузить в IX адрес V
    ld    a,(ix+0)    ; возьмем V
    add    a,(ix+1)    ; сложим с W
    dec    a        ; вычтем 1
    ld    (ix+2),a    ; запишем в U
    ret

V:  defb    2
W:  defb    3
U:  defs    1
; ------------------------------------------------------
```

Пользуясь таблицей команд, посчитайте длину и время выполнения каждой из версий. Какая лучше?

Уберите во второй версии команду `LD HL,V`. Что выполняет следующая программа ?

```
; ------------------------------------------------------
; программа вычисления выражений вида  U=V+W-1, версия 4
; ------------------------------------------------------
MAIN:   ld    hl,V1
    call    START
    ld    hl,V2
    call    START
    ld    hl,V3
    call    START
    ret

V1: defb    2
W1: defb    'J'
U1: defs    1

V2: defb    7
W2: defb    11101b
U2: defs    1

V3: defb    45h
W3: defb    62h
U3: defs    1
; ------------------------------------------------------
```

Можно ли того же эффекта добиться от версий 1 и 3 ?

Теперь попробуем обработать сразу сто троек (V,W,U):

```
; ------------------------------------------------------
; программа вычисления выражений вида  U=V+W-1, версия 5
; ------------------------------------------------------
MAIN:
    ld    hl,V1    ; загрузить в HL адрес начала
    ld    b,100    ; задать счетчик цикла
LOOP:           ; метка начала цикла
    push    bc    ; сохранить счетчик в стеке
    push    hl    ; сохранить адрес
    call    START    ; вызвать с первым значением HL
    pop    hl    ; восстановить адрес
```

```
    ld    bc,3    ; загрузить в BC длину тройки
    add    hl,bc    ; увеличить указатель на 3
    pop    bc    ; восстановить счетчик цикла
    djnz    LOOP    ; проверить цикл на окончание
    ret

V1: defb    2    ; область данных (тройки V,W,U)
W1: defb    'J'
U1: defs    1
  ...    ; и так далее
; ------------------------------------------------------
```

Эти примеры выполнены недостаточно рационально. Сможете ли Вы улучшить их, избавившись от лишних операций? Обратите особое внимание на то, чтобы последовательность команд не выполнялась из области данных, иначе данные будут восприняты как коды команд, и программа выйдет из повиновения.

А сейчас попробуем подобрать эквиваленты к основным понятиям BASICа. Надеемся, что это поможет Вам быстрее сориентироваться:

```
; объявление переменных X1 = 5 и X2 = FFEEh

X1: defb    5        ; X1 - однобайтовая
X2: defw    0FFEEH        ; X2 - двухбайтовая

; объявление массивов DIM X1(100), X2(100)

X1: defs    100
X2: defs    100 * 2

; объявление строковой переменной XS$ = "HELLO"

XS: defm    "HELLO"
    defb    0    ; 0 - признак конца строки

; операторы присваивания LET X1 = 5 и LET X2 = FFEEh

    ld    a,5
    ld    (X1),a    ; если регистр HL занят
; или
    ld    hl,X1
    ld    (hl),5    ; если регистр A занят
;
    ld    hl,0FFEEh
    ld    (X2),hl

; операторы присваивания LET X1(15) = 0 и LET X2(15) = FFEEh

    xor    a        ; XOR A обнуляет аккумулятор
    ld    (X1 + 15),a    ; вычисляется ассемблером
;
    ld    hl,0FFEEh
    ld    (X1 + 15 * 2),hl    ; то же

; операторы присваивания LET X1(X2) = Y1 и LET X2(X1) = Y2

    ld    a,(Y1)        ; значение Y1
    ld    hl,(X2)        ; значение X2
    ld    de,X1        ; заголовок массива X1()
    add    hl,de        ; смещение на нужный элемент
    ld    (hl),a        ; присвоение
;
```

```
    ld    bc,(Y2)         ; значение Y2
    ld    hl,(X1)         ; значение X1 поступит в L
    ld    h,0         ; превратить байт в слово
    add    hl,hl          ; умножить смещение на 2
    ld    de,X2         ; заголовок массива X2()
    add    hl,de          ; адрес нужного элемента
    ld    (hl),c          ; сначала младший байт Y2
    inc    hl         ; к следующему байту слова
    ld    (hl),b          ; затем старший байт Y2

; умножение на константу X1 * 2 и X2 * 8

    ld    a,(X1)
    add    a,a         ; X1 * 2 = X1 + X1
;
    ld    hl,(X2)
    add    hl,hl
    add    hl,hl
    add    hl,hl         ; X2 * 8 = ((X2 * 2) * 2) * 2

; умножение на константу X1 * 3 и X2 * 15

    ld    a,(X1)
    ld    c,a
    add    a,a
    add    a,c         ; X1 * 3 = (X1 * 2) + X1
;
    ld    hl,(X2)
    ld    e,l
    ld    d,h         ; скопировать в DE
    add    hl,hl
    add    hl,hl
    add    hl,hl
    add    hl,hl          ; HL * 16
    or    a         ; сбросить бит carry
    sbc    hl,de          ; X2 * 15 = X2 * 16 - X2

; деление на константу X1 / 2 и X2 / 8

    ld    a,(X1)
    srl    a     ; сдвиг вправо равносилен делению
;
    ld    hl,(X2)
    srl    h
    rr    l     ; сдвиг слова через бит carry
    srl    h
    rr    l
    srl    h
    rr    l     ; X2 / 8 = ((X2 / 2) / 2) / 2

; умножение двух переменных X1 * Y1 и X2 * Y2

    ld    a,(X1)
    ld    c,a     ; X1 поступает в C
    ld    a,(Y1)     ; Y1 поступает в A
MULT:   ld    h,a     ; переложить Y1 в H
    xor    a     ; обнулить будущее произведение
    ld    b,8     ; загрузить счетчик битов
MULT1:  add    a,a     ; сдвиг произв-я на разряд влево
    add    hl,hl     ; сдвиг битов Y1 через carry
    jr    nc,MULT2     ; если очередной бит = 0
    add    a,c     ; иначе произведение + X1
MULT2:  djnz    MULT1     ; повторить цикл
    ret         ; произведение в A
```

```
;
    ld    hl,(X2)     ; X2 поступает в HL
    ld    de,(Y2)     ; Y2 поступает в DE
MULT:   ld    b,h
    ld    c,l     ; переложить X2 в BC
    ld    hl,0    ; обнулить произведение
    ld    a,16    ; счетчик 16 битов
MULT1:  add    hl,hl    ; сдвиг произведения влево
    ex    de,hl    ; обмен DE<->HL
    add    hl,hl    ; сдвиг Y2 через carry
    ex    de,hl    ; восстановить
    jr    nc,MULT2    ; если очередной бит = 0
    add    hl,bc    ; иначе произведение + X2
    dec    a    ; уменьшить счетчик цикла
    jr    nz,MULT1    ; повторить, если не 0
    ret        ; произведение в HL

; деление двух переменных X1 / Y1 и X2 / Y2

    ld    a,(Y1)
    ld    c,a    ; делитель в C
    ld    a,(X1)    ; делимое в A
DIV:    ld    l,a    ; частное = X1
    ld    h,0    ; остаток = 0
    ld    b,8    ; счетчик 8 битов
DIV1:   add    hl,hl    ; сдвиг влево на 1 бит
    ld    a,h    ; остаток в A
    jr    c,DIV2    ; если был carry
    cp    c    ; сравнить с Y1
    jr    c,DIV3    ; если меньше
DIV2:   sub    c    ; иначе вычесть Y1
    inc    l    ; увеличить частное
    ld    h,a    ; сохранить остаток
DIV3:   djnz    DIV1    ; повторять цикл
    ret        ; частное в L, остаток в H
;
    ld    hl,(X2)     ; делимое в HL
    ld    de,(Y2)     ; делитель в DE
DIV:    ld    b,d
    ld    c,e    ; передать Y2 в BC
    ld    de,0    ; остаток = 0
    ld    a,16    ; счетчик 16 битов
DIV1:   push    af    ; сохранить счетчик
    add    hl,hl    ; сдвиг X2 влево на 1 разряд
    rl    e
    rl    d    ; сдвиг Y2 через carry
    jr    c,DIV2    ; если carry
    ld    a,e
    sub    c
    ld    a,d
    sbc    b    ; сравнить BC и DE
    jr    c,DIV3    ; если меньше
DIV2:   ld    a,e
    sub    c
    ld    e,a
    ld    a,d
    sbc    b
    ld    d,a    ; вычесть BC из DE
    inc    l    ; увеличить частное
DIV3:   pop    af    ; восстановить счетчик
    dec    a
    jr    nz,DIV1    ; повторить цикл
    ret        ; частное в HL, остаток в DE
```

```
; превращение числа из байта в слово: X1 -> X2

    ld    hl,(X1)     ; для чисел 0...255
    ld    h,0
    ld    (X2),hl
;
    ld    a,(X1)      ; для чисел -128...127
    ld    l,a
    rlca
    sbc    a     ; распространение знака
    ld    h,a
    ld    (X2),hl

; унарный минус -X1 и -X2

    ld    a,(X1)
    neg         ; готовая операция
;
    ld    hl,(X2)
    ld    a,h
    cpl
    ld    h,a
    ld    a,l
    cpl
    ld    l,a     ; инвертировать все биты
    inc    hl     ; прибавить 1

; операторы GOSUB xxxx и RETURN

    call    LABEL     ; вызов подпрограммы
    ...
LABEL:  ...         ; подпрограмма
    ret         ; выход из подпрограммы

; оператор RETURN xxxx (выход на нужный номер строки)

    pop    hl     ; изъять из стека адрес возврата
    jp    LABEL     ; перейти на нужный адрес

; операторы IF X1 = 0 THEN xxxx и IF X2 = 0 THEN xxxx

    ld    a,(X1)
    or    a     ; так короче, чем CP 0
    jr    z,LABEL     ; переход на метку
;
    ld    hl,(X2)
    ld    a,h
    or    l     ; HL ≠ 0 если H или L ≠ 0
    jr    z,LABEL     ; переход на метку

; операторы IF X1 < Y1 THEN xxxx ELSE xxxx
;       IF X2 < Y2 THEN xxxx ELSE xxxx

    ld    a,(Y1)
    ld    b,a
    ld    a,(X1)
    cp    b     ; сравнить
    jr    c,LAB1     ; если меньше - выполнить THEN
  ...    ; часть ELSE
    jr    LAB2     ; перепрыгнуть часть THEN
LAB1:   ...         ; часть THEN
LAB2:            ; выход из оператора IF
;
    ld    bc,(Y2)
```

```
    ld    hl,(X2)
    ld    a,l
    sub    c    ; сравнить младший байт
    ld    a,h
    sbc    b    ; сравнить старший байт
    jr    c,LAB1    ; если меньше - выполнить THEN
  ...    ; часть ELSE
    jr    LAB2    ; перепрыгнуть часть THEN
LAB1:   ...        ; часть THEN
LAB2:           ; выход из оператора IF

; оператор FOR I=X1 TO Y1 STEP 3

    ld    a,(Y1)
    ld    b,a    ; предел в B
    ld    a,(X1)    ; A - цикловая переменная
LOOP:   cp    b    ; проверка на окончание цикла
    jp    z,EXIT    ; выход, если равно
    push    bc
    push    af    ; сохранить в стеке
  ...    ; тело цикла
    pop    af    ; восстановить
    pop    bc
    inc    a
    inc    a
    inc    a    ; увеличить счетчик на 3
    jp    LOOP    ; повторить
EXIT:           ; метка выхода из цикла

; вложенные циклы

    ld    b,5    ; описать внешний цикл
LOOP1:  push    bc
    ld    b,3    ; описать внутренний цикл
LOOP2:  push    bc
  ...    ; тело внутреннего цикла
    pop    bc
    djnz    LOOP2    ; проверка внутреннего цикла
    pop    bc
    djnz    LOOP1    ; проверка внешнего цикла

; оператор ON X2 GOTO

    jp    (hl)    ; адрес перехода - в HL

; оператор ON X2 GOSUB

    ld    de,RETADR
    push    de    ; поместить в стек адрес возврата
    jp    (hl)    ; в HL - адрес подпрограммы
RETADR:         ; точка возврата из подпрограмм

; сравнение строк XS$ и YS$ (с устновкой бита Z)

    ld    hl,XS    ; HL указывает на XS$
    ld    de,YS    ; DE указывает на YS$
COMPAR: ld    a,(de)    ; взять очередной байт строки YS
    cp    (hl)    ; сравнить с очередным байтом XS
    ret    nz    ; выход с Z=0, если не равны
    inc    hl    ; к следующему байту XS
    inc    de    ; к следующему байту YS
    or    a    ; конец строки ?
    jr    nz,COMPAR    ; нет-повторять
    ret        ; выход с Z=1
```

Способы передачи параметров в подпрограмму

Одна из наиболее привлекательных черт машинного языка - это возможность построить великое множество самых различных, иногда довольно экзотических решений одной и той же задачи.

Примером может служить проблема передачи параметров в подпрограммы, которая решается несколькими способами.

1. Передача параметров через регистры.

Если число параметров невелико, их можно передавать через регистры, как это сделано, например, в наших версиях 4 и 5. Этот способ наиболее предпочтителен, как самый быстрый и компактный.

2. Передача параметров через ячейки ОЗУ.

Для передачи параметров можно использовать ячейки ОЗУ. Этот способ практикуется, когда данные имеют сложный формат, или их количество превосходит количество регистров. Широко применяется, например, передача данных через буфер.

3. Передача параметров через стек.

Можно передавать параметры, засылая их в стек перед вызовом подпрограммы. Этот способ привлекателен для передачи параметров, количество которых может меняться (естественно, оно каждый раз должно сообщаться подпрограмме), но он несколько тоньше, чем предыдущие. Дело в том, что команда CALL помещает в верхушку стека адрес возврата, который нужно сохранить для корректного срабатывания команды RET. Кроме того, во избежание переполнения стека, переданные параметры следует из него изымать.

4. Передача параметров через индексные регистры.

Этот путь представляет собой нечто среднее между 1 и 2. В памяти располагают нужное количество однотипных структур, а перед вызовом помещают адрес требуемой структуры в регистр IX или IY. Подпрограмма вместо конкретных адресов использует смещения данных относительно значения индексного регистра.

5. Передача параметров через биты признаков.

Широко распространена передача данных типа "да-нет" при помощи битов C и Z, ибо их легче всего протестировать. Бит C можно установить командой SCF, а сбросить командой AND A или OR A. Бит Z используется реже (в основном, для возврата ответа из подпрограммы); его можно установить, выполнив CP A.

6. Передача параметров после команды вызова.

Если параметрами подпрограммы являются КОНСТАНТЫ, можно применить один очень остроумный прием. Представим себе, что нам нужно передать в подпрограмму байт и два слова. Оформим вызов так:

```
    call    SUBR        ; вызов подпрограммы SUBR
    defb    BYTE        ; байт (аргумент #1)
    defw    WORD1        ; слово (аргумент #2)
    defw    WORD2        ; слово (аргумент #3)
```

Хитрость заключается в том, что адрес возврата, который положит в стек команда CALL, будет указывать на первый параметр, поэтому заголовок процедуры SUBR нужно оформить так:

```
SUBR:   ex    (sp),hl        ; передать в HL адрес возврата
```

```
    ld    a,(hl)        ; поместить в A аргумент #1
    inc    hl
    ld    e,(hl)
    inc    hl
    ld    d,(hl)        ; поместить в DE аргумент #2
    inc    hl
    ld    c,(hl)
    inc    hl
    ld    b,(hl)        ; поместить в BC аргумент #3
    inc    hl         ; получен новый адрес возврата
    ex    (sp),hl         ; поместить его обратно в стек
    ...
```

Таким образом, под воздействием подпрограммы инструкция CALL с параметрами воспримется микропроцессором как "единая" команда вызова.

Среда MSX-DOS

Как Вы сами понимаете, нормальная программа не способна обойтись без обмена информацией с внешним миром. Осуществить такой обмен помогают разнообразные периферийные устройства, в минимальный перечень которых входят: клавиатура, дисковод, символьный дисплей и принтер.

Чтобы избавить Вас от излишних подробностей, все заботы по поддержке этих устройств берет на себя MSX-DOS, предлагая Вам набор из 42 процедур, называемых СИСТЕМНЫМИ ВЫЗОВАМИ (мы рассмотрим наиболее употребительные). Благодаря вызовам, не только облегчается сам процесс программирования, но и дости-гается совместимость различных моделей компьютеров.

Ранее упоминалось, что MSX-DOS рассматривает программы как свои внешние команды, которые загружаются в память (и начинают исполняться) с адреса 100h. Выход обратно в MSX-DOS осуществляется любой из команд: RST 0, CALL 0, JP 0 или RET.

Верхний адрес доступной Вам памяти может изменяться в зависимости от количества дисководов, подключенных к машине и версии MSXDOS.SYS, поэтому в каждом конкретном случае его следует получать из ячеек 0006h и 0007h.

Адрес 0005h в среде MSX-DOS используется для системных вызовов. Наиболее полезны из них следующие:

; Проверка состояния клавиатуры

```
    ld    c,0Bh     ; номер процедуры
    call    0005h     ; A = FFh, если было нажатие, иначе 0
```

; Ввод символа с клавиатуры с ожиданием и выводом на дисплей

```
    ld    c,01h     ; номер процедуры
    call    0005h     ; код введенного символа в A
```

; То же без вывода на дисплей

```
    ld    c,07h     ; номер процедуры
    call    0005h     ; код введенного символа в A
```

; Ввод строки с клавиатуры (заканчивается клавишей RETURN).
; Строка сохраняется в буфере, который должен иметь
; достаточные размеры. Длина буфера должна быть записана в
; его первом байте. По окончании ввода в его втором байте
; будет сохранена длина введенной строки, а далее - собственно
; строка.

```
    ld    de,BUF    ; адрес буфера в DE
    ld    c,0Ah     ; номер процедуры
    call    0005h

; Вывод символа на дисплей

    ld    a,'X'     ; код символа в A
    ld    c,02h     ; номер процедуры
    call    0005h

; Вывод строки на дисплей (конец строки - символ $)

    ld    de,XS     ; адрес строки в DE
    ld    c,09h     ; намер процедуры
    call    0005h
```

С помощью этих же процедур управляется курсор. Например, для перехода к новой строке достаточно вывести на печать код 13. Сдвиг курсора на позицию вниз осуществляется кодом 10.

Перемещения курсора организуются также цепочками кодов:

- вверх: `27,'A'`
- вниз: `27,'B'`
- вправо: `27,'C'`
- влево: `27,'D'`
- в левый верхний угол экрана: `27,'H'`
- в произвольную позицию (X,Y): `27,'Y',Y+32,X+32`
- очистить экран: `27,'E'` и т.д.

```
; Вывод символа на принтер

    ld    a,'X'     ; код символа в A
    ld    c,05h     ; номер процедуры
    call    0005h
```

Несколько замечаний.

Во-первых, адрес `0005h` целесообразно заменить на `F37Dh`, поскольку последний корректен не только в режиме `MSX-DOS`, но и в любом другом.

Во-вторых, ни один вызов не гарантирует сохранности регистров, поэтому их нужно сохранять в стеке.

В-третьих, недостающие процедуры легко скомбинировать из представленных, например:

```
; Опрос клавиатуры без ожидания

INKEY:  ld    c,0Bh
    call    5     ; опрос состояния клавиатуры
    or    a
    ret     z     ; выход с Z=1, если не было нажатия
    ld    c,7
    call    5     ; иначе получение кода с клавиатуры
    or    a
    ret         ; выход с Z=0 и кодом символа в A

; Вывод строки на принтер

    ld    de,XS
PRINT:  ld    a,(de)     ; взять очередной символ строки
    cp    '$'     ; это признак конца строки ?
    ret     z     ; возврат, если да
    push    de     ; сохранить текущий адрес
    ld    c,5
    call    5     ; вывести байт на принтер
    pop     de     ; восстановить адрес
```

```
    inc     de     ; перейти к следующему байту
    jr     PRINT     ; повторить
```

Определенные трудности представляет ввод / вывод чисел, однако они вполне преодолимы.

Работа с файлами

Магнитный диск представляет собой сложную и гибкую информационную структуру, для всесторонней обработки которой предусмотрено множество разнообразных процедур.

Мы познакомимся с наиболее употребительным - файловым - методом, которого будет достаточно для реализации 90% Ваших замыслов.

Чтобы программа могла обрабатывать файлы, в ней следует предусмотреть так называемый FCB (File Control Block, блок управления файлами) - область из 37 байт, которая содержит оперативную информацию:

- номер дисковода: 0 - текущий, 1 - A:, 2 - B: (1 байт)
- имя файла (8 байт)
- расширение файла (3 байта)
- текущий блок файла (2 байта)
- размер записи (2 байта)
- размер файла (4 байта)
- дата создания (2 байта)
- время создания (2 байта)
- устройство ввода/вывода (1 байт)
- положение файла в каталоге (1 байт)
- первый кластер (2 байта)
- текущий кластер (2 байта)
- длина обработанной части файла в кластерах (2 байта)
- текущая запись (1 байт)
- номер произвольной записи (4 байта)

Блоков FCB должно быть столько, сколько одновременно обрабатываемых файлов. Понятия "блок", "сектор" и "кластер" означают дисковые информационные единицы (обычно 128, 512 и 1024 байта).

Благодаря FCB, в распоряжении программиста оказываются простые и хорошо интерпретируемые понятия: номер дисковода, имя и расширение файла, а также размер и номер записи.

С точки зрения программиста, файл можно "расчленить" на множество записей произвольной длины и обращаться к нужной записи по ее номеру. При этом свобода выбора абсолютна: файл можно резать на байты, слова и т.д., или трактовать его целиком как одну запись. Номер записи автоматически увеличивается на 1, тем самым предельно упрощая последовательный доступ (не забудьте об обратном порядке следования байтов в числе).

Имя и расширение могут содержать символ ? для поиска по "маске".

Формат времени и даты следующий:

ВРЕМЯ:

- старший байт: h4 h3 h2 h1 h0 m5 m4 m3
- младший байт: m2 m1 m0 s5 s4 s3 s2 s1

Здесь:

- hx - биты часов 0...23
- mx - биты минут 0...59
- sx - биты секунд/2 (0...29)

ДАТА:
- старший байт: y6 y5 y4 y3 y2 y1 y0 m3
- младший байт: m2 m1 m0 d4 d3 d2 d1 d0
Здесь:
yx - биты года (0...99 - соответственно 1980...2079)
mx - биты месяца (1...12)
dx - биты дня (1...31)

Располагая FCB, следите, чтобы он не попал в диапазон адресов 4000h...7FFFh. Причина такого ограничения прояснится позднее.

ЧТЕНИЕ ФАЙЛА

Зарезервируйте в ОЗУ FCB. Поместите в поля "имя файла", "расширение" и "номер дисковода" требуемые данные. Остальную часть FCB необходимо обнулить.
Для нормальной работы MSX-DOS нужно загрузить с диска в FCB служебную информацию об указанном файле. Эта процедура называется "открытием" файла и выполняется автоматически:

```
ld    de,FCB    ; адрес FCB поместить в DE
ld    c,0Fh
call    0005h    ; A=0, если успешно, иначе A=FFh
```

После этого следует установить желаемый размер записи и обнулить номер записи:

```
ld    hl,1        ; читать по 1 байту
ld    (FCB + 14),hl
ld    hl,0
ld    (FCB + 33),hl
ld    (FCB + 35),hl    ; установить на запись #0
```

Чтобы прочитать одну или несколько записей (допустим, три), достаточно выполнить следующее:

```
ld    de,RAM         ; адрес ОЗУ, куда поступит
ld    c,1Ah         ;      считанная информация
call    0005h
ld    hl,3         ; считать 3 записи подряд
ld    de,FCB         ; DE ссылается на FCB файла
ld    c,27h         ; чтение
call    0005h         ; A=0, если успешно, иначе A=1
```

Кроме этого, HL содержит количество реально прочитанных записей. Меняя номер записи в ячейках FCB+33,+34,+35,+36, Вы можете получить доступ к любой точке файла, а переключаясь с одного FCB на другой,- обрабатывать несколько файлов сразу.

СОЗДАНИЕ НОВОГО ФАЙЛА

Если программа должна создать файл заново (уничтожив при этом старый файл с тем же именем), нужно подготовить FCB и выполнить процедуру "создания" файла:

```
ld    de,FCB    ; адрес FCB
ld    c,16h
call    0005h    ; A=0, если успешно, иначе A=FFh
```

после чего также выставить размер и номер записи.

Вывод одной или нескольких записей в файл аналогичен их чтению:

```
    ld    de,RAM         ; адрес ОЗУ, откуда поступит
    ld    c,1Ah          ;      информация для записи
    call    0005h
    ld    hl,3          ; вывести 3 записи подряд
    ld    de,FCB          ; DE ссылается на FCB файла
    ld    c,26h          ; запись
    call    0005h          ; A=0, если успешно, иначе A=1
```

Поскольку файл создан заново, его характеристики нужно поместить на диск с помощью процедуры "закрытия" файла:

```
    ld    de,FCB
    ld    c,10h
    call    0005h    ; A=0, если успешно, иначе A=FFh
```

ВНЕСЕНИЕ ИСПРАВЛЕНИЙ В ИМЕЮЩИЙСЯ ФАЙЛ

Внесение исправлений в файл начинается, как и чтение, с его открытия. После коррекции файл, естественно, закрывают.

УДАЛЕНИЕ ФАЙЛА С ДИСКА

```
    ld    de,FCB    ; FCB уничтожаемого файла
    ld    c,13h
    call    0005h    ; A=0, если успешно, иначе A=FFh
```

Первая буква имени файла будет изменена в каталоге на русское "Е" (ASCII-код E5h).

СМЕНА ДИСКОВОГО ИМЕНИ ФАЙЛА

Новое имя и новое расширение следует поместить в FCB, начиная с 18-го байта, после чего выполнить:

```
    ld    de,FCB
    ld    c,17h
    call    0005h    ; A=0, если успешно, иначе A=FFh
```

ПОИСК ФАЙЛА В КАТАЛОГЕ

Каждый файл автоматически регистрируется в каталоге диска в следующем виде (всего 32 байта):

- имя файла (8 байт)
- расширение (3 байта)
- атрибут (1 байт)
- пустая область для совместимости с MS-DOS (10 байт)
- время создания (2 байта)
- дата создания (2 байта)
- первый кластер файла (2 байта)
- размер файла (4 байта)

Если атрибут файла равен 2, доступ к файлу невозможен. Время, дата и размер имеют тот же формат, что и в FCB.

Поместите в FCB номер дисковода, имя и расширение файла (последние могут содержать произвольное число знаков '?'), затем выполните следующую процедуру:

```
    ld    de,80h
    ld    c,1Ah
    call    5
```

которая разместит область для обмена с диском по адресу 80h.

Первое вхождение имени файла разыскивается процедурой:

```
    ld    de,FCB
    ld    c,11h
    call    5    ; A=0, если найдено, иначе A=FFh
```

Если имя найдено в каталоге, оно копируется по адресу обмена 80h в следующем виде (всего 33 байта):
- имя дисковода (1 байт)
- информация о файле (как и в каталоге)

Все дальнейшие вхождения разыскиваются процедурой:

```
    ld    c,12h
    call    5    ; A=0, если найдено, иначе A=FFh
```

которая не требует больше указания адреса FCB. Эта процедура возвращает сведения в том же виде.

ПЕРЕДАЧА ПАРАМЕТРОВ КОМАНДНОЙ СТРОКИ

В свое время упоминалось, что внешняя команда способна получать аргументы из командной строки. Как это происходит?
MSX-DOS всегда трактует первый и второй аргументы как имена файлов, поэтому соответственно их оформляет (номер дисковода, имя файла и расширение) и помещает по адресам 5Ch и 6Ch. Отсюда их копируют в FCB.
Командную строку можно расшифровать и "руками", так как она полностью сохраняется, начиная с адреса 80h. Первый байт строки играет роль счетчика символов.

ПЕРЕХВАТ ОШИБОК ДИСКОВОДА

Из предыдущих примеров видно, что многие системные вызовы возвращают в регистре A ненулевое значение, если цель не была достигнута (например, файл отсутствовал в каталоге). Более серьезные ошибки (отсутствие диска в дисководе, сбой по контрольной сумме, искажение структуры диска и т.д.) перехватываются иначе.
По адресу F323h запишите адрес ячейки памяти, которая содержит адрес Вашей процедуры обработки ошибок. Чтобы подавить запрос "Abort, Retry, Ignore ?", нужно перед системным вызовом запомнить в отдельной ячейке состояние SP, которое Ваша процедура обработки ошибок должна восстановить перед возвратом.
В момент перехвата регистр C содержит информацию о зарегистрированной ошибке:
- выставлен старший бит - ошибка FAT;
  - C = 10 - неизвестный формат диска;
  - (C and FEh) = 0 - установлен протектор;
  - (C = 2) - устройство не готово;
- остальные состояния - ошибка дисковода.

Максимальный доступный номер дисковода можно определить по адресу F347h.

Слотовая конфигурация MSX

Конфигурация компьютеров MSX/MSX2 может варьироваться в широчайших пределах, благодаря исключительно гибкой структуре памяти этих машин.

Вся оперативная (RAM) и постоянная (ROM) память машины распределяется между четырьмя СЛОТАМИ - отдельными адресными пространствами, из которых компонуются различные варианты рабочего пространства микропроцессора.

Каждый слот может:

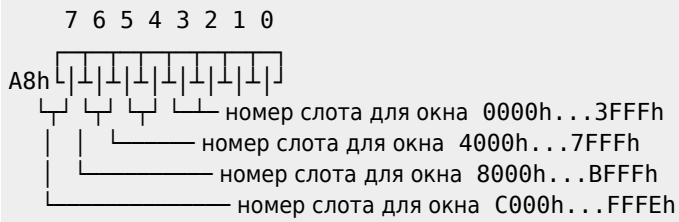
- содержать 0...4 страницы ROM или RAM (по 16 K);
- в свою очередь, делиться на 4 вторичных слота, каждый из которых может объединять по 0...4 страницы памяти.

Такая разветвленная структура поддерживает до 1 Мб.

Слоты могут быть дополнены МЭППЕРОМ, который позволяет увеличить число страниц в слоте до 8 и более, т.е. расширить доступную память еще в несколько раз.

Адресное пространство микропроцессора, равное, как известно, 65536 байтам, разделено на четыре одинаковых "окна". Если для какого-либо из этих окон указать номер первичного слота (и номер вторичного, если есть), то в нем "высветится" страница RAM или ROM, которая находится в заданном диапазоне адресов выбранного слота.

Номер слота для каждого из окон указывают в специальном регистре, роль которого чаще всего играет порт A8h. Его формат приведен ниже:



Выбор вторичных слотов производится при помощи дополнительного регистра FFFFh, формат которого аналогичен. Обращение к нему производится так же, как к рядовой ячейке памяти, но с некоторым отличием: данный регистр возвращает свое значение в инверсном виде, хотя принимает в нормальном.

Номера слотов принято представлять в формате F000SSPPb: здесь биты PP представляют номер первичного слота, биты SS - вторичного. Если слот расширен, флаг F равен 1, иначе 0.

ПРИМЕР. Предположим, что ROM BIOS и ROM BASIC-интерпретатора находятся в слоте 0, по адресу 0000h и 4000h соответственно, а вся RAM компьютера подключена к слоту 3-2. Допустим также, что слоты 1 и 2 оформлены как внешние разъемы для катриджей. Как может выглядеть состояние регистров при некоторых наиболее употребительных конфигурациях памяти?

A. Режим MSX-DOS:

A8h = 11111111b, FFFFh = 10101010b

- 0000h...3FFFh: RAM слот 10001011b
- 4000h...7FFFh: RAM слот 10001011b
- 8000h...BFFFh: RAM слот 10001011b
- C000h...FFFEh: RAM (системная) слот 10001011b

B. Режим MSX-BASIC:

```
A8h = 11110000b, FFFFh = 10100000b

- 0000h...3FFFh: ROM BIOS      слот 00000000b
- 4000h...7FFFh: ROM BASIC     слот 00000000b
- 8000h...BFFFh: RAM       слот 10001011b
- C000h...FFFEh: RAM (системная)  слот 10001011b
```

C. К слоту 2 подключен игровой катридж:

```
A8h = 11111000b, FFFFh = 10100000b

- 0000h...3FFFh: ROM BIOS      слот 00000000b
- 4000h...7FFFh: ROM (катридж) слот 00000010b
- 8000h...BFFFh: RAM       слот 10001011b
- C000h...FFFEh: RAM (системная)  слот 10001011b
```

Слот, оборудованный мэппером, имеет еще четыре регистра для отдельного выбора страниц памяти:

- порт FCh: номер страницы для окна 0000h...3FFFh
- порт FDh: номер страницы для окна 4000h...7FFFh
- порт FEh: номер страницы для окна 8000h...BFFFh
- порт FFh: номер страницы для окна C000h...FFFEh

Действительное значение имеют только несколько младших битов регистра (в зависимости от количества страниц), поэтому старшие биты следует отбрасывать. Порты FCh...FFh, в отличие от регистров A8h и FFFFh, позволяют произвольно переставлять страницы,- например, менять их местами или выбирать одну страницу для нескольких окон одновременно (при помощи мэппера можно оперировать с RAM любой величины,"высвечивая" ту или иную ее страницу в "окне" 8000h...BFFFh).

Начальная инициализация регистров мэппера обычно следующая (хотя теоретически возможны отклонения):

```
FCh = 3; FDh = 2; FEh = 1; FFh = 0 (системная)
```

В интересах дела Вы должны запретить себе всякие эксперименты с регистром FFh, а также с двумя старшими битами регистров A8h и FFFFh! Причина проста: в последнем окне адресного пространства располагается страница, содержащая системные переменные, копии дисковых FAT, FCB, стек, и т.п. Нечаянная или преднамеренная замена этой страницы на другую может обернуться, в наилучшем случае, зависанием компьютера.

Примерно по той же причине нужно остерегаться опрометчивых манипуляций с окном 0000h...3FFFh. В данном случае переключения возможны, но только между ROM BIOS и страницей #3 RAM, так как содержащаяся в них системная информация частично взаимозаменяема.

Стандарт не закрепляет за слотами каких-либо неизменных функций: фирма-изготовитель вправе размещать страницы памяти или разъемы для катриджей там, где сочтет нужным. Однако все сведения о занятости слотов содержатся в системной странице любого MSX-компьютера. Правильно написанная программа должна уметь пользоваться этой информацией, чтобы проявлять должную гибкость в любой незнакомой ситуации.

По адресу FCC1h располагаются 4 байта. Бит 7 каждого из них показывает, расширен ли соответствующий слот (= 1 - да).

По адресу FCC5h размещена информация о вторичных слотах для каждого окна (в формате регистра FFFFh).

По адресу FCC9h расположена таблица из 64 байтов, по числу возможных страниц памяти, в которую компьютер помещает

результаты самотестирования: старший бит каждого байта отражает наличие катриджа с BASIC-программой, бит 6 - катриджа с драйвером периферийного устройства, и бит 5 - ROM расширения оператора CALL.

Что же касается RAM, ее размещение можно определить по-разному. Проще всего вычислить слот самой системной страницы по данным регистров A8h и FFFFh. Можно также воспользоваться таблицей F341h, которая формируется при старте MSXDOS: в ней содержатся 4 номера слотов в формате F000SSPPb (по одному на каждое из "окон").
Оба метода очень просты, но, к сожалению, имеют крупный недостаток: не позволяют судить о наличии памяти в остальных слотах. Решить проблему может лишь прямая проверка слотов на запись/чтение.

Исследуя ресурсы RAM, полезно проверить наличие мэппера и выяснить число страниц в слоте. Исходя из того, что мэппер чаще всего не имеет ни резидентной программной поддержки, ни каких-либо системных таблиц, все делается посредством прямого доступа. Прежде всего, сравнивая значение регистра FFh со значением регистра FCh, можно сделать вывод о физическом существовании мэппера: очевидно, значения должны различаться.
Оценить количество страниц можно, посылая в регистр FEh произвольные байты и проверяя, какая группа битов считывается без искажений.

Слотовые процедуры

Из рассмотренных примеров можно понять, что обращение к слотам, которые в данный момент отключены, сопряжено с большими неудобствами: нужно очень корректно переконфигурировать память, сохранив в регистрах Z80 передаваемые аргументы, обратиться к желаемой процедуре, после чего безошибочно вернуться к исходной конфигурации, сохранив в регистрах результат. Поэтому, как обычно, проще и корректнее (по крайней мере, на первых порах), пользоваться готовыми слотовыми процедурами, к которым можно получить доступ из любого режима:

```
000Ch - ЧТЕНИЕ БАЙТА из слота
вход: A - номер слота в формате F000SSPPb
      HL - адрес байта
выход: A - прочитанное значение

0014h - ЗАПИСЬ БАЙТА в слот
вход: A - номер слота в формате F000SSPPb
      HL- адрес байта
      E - значение для записи

001Ch - ОБРАЩЕНИЕ К ПОДПРОГРАММЕ в другом слоте
вход: старший байт IY - номер слота в формате F000SSPPb
      IX - адрес подпрограммы
   требуемые параметры в регистрах Z80
выход: зависит от вызываемой подпрограммы

0024h - ВЫБОР слота
вход: A - номер слота в формате F000SSPPb
   2 старших бита HL - номер "окна" (остальные биты = 0)

0030h - ОБРАЩЕНИЕ К ПОДПРОГРАММЕ в другом слоте
вход: требуемые параметры в регистрах Z80
   Вызов оформляется следующим образом:
```

```
    rst    30h        ; рестарт = CALL 30h
    defb    F000SSPPb    ; номер слота
    defw    SUBR        ; адрес подпрограммы

выход: зависит от вызываемой подпрограммы


   Программа должна уметь определять собственное положение
и представлять его в формате F000SSPPb. Эта задача возникает
достаточно часто, а потому решается стандартно:

; ВНИМАНИЕ !
; Процедура верна для диапазона адресов 0000h...3FFFh
; Если интересующий фрагмент программы находится в другом
; диапазоне, внесите в текст следующие изменения:
; 4000h...7FFFh: уберите все RLCA,
;        перед AND 11b поставьте две RRCA;
; 8000h...BFFFh: замените все RLCA на две RRCA,
;        перед AND 11b поставьте четыре RRCA;
; C000h и далее: замените две RLCA на четыре RRCA,
;        перед AND 11b поставьте две RLCA.

MYSLOT: in    a,(0A8h)
    and    11b        ; определить первичный слот
    ld    c,a
    ld    b,0
    ld    hl,FCC1h
    add    hl,bc        ; найти слот в таблице
    or    (hl)        ; установить бит F формата
    ld    c,a
    inc    hl
    inc    hl
    inc    hl
    inc    hl
    ld    a,(hl)        ; найти слот в таблице
    rlca
    rlca
    and    1100b        ; определить вторичный слот
    or    c        ; сформировать F000SSPPb в A
    ret
```

BIOS

В каждый без исключения компьютер MSX2 встроено большое количество подпрограмм со стандартными правилами вызова, заготовленных практически на любой случай жизни:

- опрос мануальных устройств (клавиатуры, мыши, трекбола, джойстика, светового пера, тач-панели и пр.);
- работа с графикой (смена режимов экрана, точечно-векторная и растровая графика, регулировка палитры, прямой доступ к VDP, спрайты, скоростные пересылки фрагментов, гашение
- дисплея и пр.);
- ввод и вывод строк, управление координатами и стилем курсора;
- музыкально-шумовые эффекты, битовый ввод–вывод звука, прямой доступ к AUDIO;
- математический пакет (разбор выражений, арифметика с плавающей точкой, тригонометрия);
- обслуживание слотов, принтера, видеотайзера, таймера;

Этот набор носит сокращенное название BIOS и размещается в двух ROM, именуемых MAIN-ROM (есть и у MSX, и у MSX2) и SUB-ROM (только у MSX2). Каждая ROM имеет стандартный список входных точек, к которым и производится обращение.

BIOS составляет основу всей «философии MSX»: если Ваша программа все свои операции проводит строго через BIOS, она полностью застрахована от коллизий, связанных с переходом на другую модель компьютера или непредвиденным изменением его конфигурации.

Доступ к входным точкам MAIN-ROM и SUB-ROM открывается,благодаря межслотовым процедурам. Требуемый номер слота для MAIN-ROM содержится по адресу FCC1h, для SUB-ROM — по адресу FAF8h, поэтому корректный вызов точки BIOS выглядит так:

```
; вызов подпрограммы из MAIN-ROM

    ld    ix,SUBR        ; адрес входной точки SUBR
    ld    iy,(0FCC0h)    ; номер слота поступает...
    call    001Ch        ; ...в старший байт IY

; вызов подпрограммы из SUB-ROM

    ld    ix,SUBR        ; аналогично
    ld    iy,(0FAF7h)
    call    001Ch
```

Кстати, машины MSX не имеют SUB-ROM, поэтому в их ячейке FAF8h содержится 0, что позволяет отличать их от MSX2.

Важно отчетливо осознавать, что стандартом закреплены только адреса входных точек, правила их вызова и выполняемые ими действия, в то время как расположение и содержание самих подпрограмм имеет право изменяться от машины к машине: в том и заключается миссия BIOS, чтобы компенсировать аппаратные различия компьютеров программным путем.

Попутно заметим, что интерпретатор языка MSX BASIC не является частью BIOS (как и любая другая программа, содержащаяся во внешнем или резидентном катридже), поэтому любые адреса и процедуры в теле BASIC-ROM имеют право радикально изменяться вместе с моделью компьютера.

Видеопроцессор (VDP) MSX-VIDEO

Рассуждая о графических возможностях машин MSX2, трудно воздержаться от восхищенных эпитетов в адрес видеопроцессора (VDP) V9938 MSX-VIDEO, благодаря которому компьютеры MSX2 не уступили в мультипликационной графике мощным 16/32-разрядным персональным машинам.

MSX-VIDEO — это не просто графический адаптер, задача которого сводится только лишь к отображению памяти на экран. Это независимый процессор, самостоятельно выполняющий массу специфических операций, для обстоятельного изучения которых нужен отдельный разговор. Попытаемся хотя бы перечислить их.

MSX-VIDEO синтезирует экранное изображение на основе данных, находящихся в отдельной видеопамяти (VRAM), выдавая на разъемы сигнал RGB и композитный телесигнал в стандарте PAL или NTSC. Он имеет 2 текстовых и 8 графических режимов, близких по характеристикам к адаптерам CGA, EGA и VGA. Его цветовая палитра содержит 512 градаций, причем на экране мо- гут сосуществовать 256 цветов. Благодаря режиму interlace, максимальная дискретность достигает 512×424.

MSX-VIDEO умеет самостоятельно рисовать точки, отрезки, прямоугольники и переносить фрагменты изображений, выполняя разнообразные операции над цветами. Исключительно компактная знакоместная графика позволяет упаковать в RAM изображение, превосходящее площадь экрана в сотни раз, а в сочетании со спрайтами обеспечивает высокоскоростную мультипликацию.

VDP может изменять цвет рамки экрана, гасить и включать экран, плавно скроллировать его в вертикальном направлении, генерировать прерывания в заданной строке развертки, мерцать символами, автоматически переключать страницы VRAM, смещать изображение относительно экранных границ, регистрировать наложение спрайтов, переходить в монохромный режим, изменять частоту развертки, оцифровывать внешний сигнал, смешивать с ним свой сигнал и многое другое.

Поскольку VDP имеет 39 внутренних управляющих регистров (со сложным битовым форматом), целесообразно оставить заботу обо всех тонкостях управления VDP процедурам BIOS и заняться более общими принципами графики MSX2.

Структура видеопамяти (VRAM)

VRAM содержит ряд таблиц, определенным образом задающих экранное изображение. Их количество и назначение зависят от выбранного режима VDP.

В "знакоместных" режимах изображение формируется в два этапа: сначала в таблице ЗНАКОГЕНЕРАТОРА описывается внешний вид символов, из которых (подобно мозаике) будет построена картинка; затем в таблице ИМЕН задается построчный порядок размещения этих символов на экране.

В режимах "бит-мэп" изображение определяется только таблицей ИМЕН. Каждый ее байт содержит цвета одной, двух или четырех соседних по горизонтали точек (по 8, 4 или 2 бита на точку).

Помимо этого, во VRAM могут размещаться таблицы:

- ЦВЕТОВ СИМВОЛОВ;
- МЕРЦАНИЯ СИМВОЛОВ;
- ШАБЛОНОВ СПРАЙТОВ;
- АТРИБУТОВ СПРАЙТОВ;
- ЦВЕТОВ СПРАЙТОВ;
- ПАЛИТРЫ (которая не влияет непосредственно на VDP, но требуется для правильной работы подпрограмм BIOS).

Адреса всех таблиц относительно постоянны, хотя, как мы уже знаем, лучше уточнить это, обратившись к системным переменным:

- F922h - содержит адрес таблицы ИМЕН;
- F924h - содержит адрес таблицы ЗНАКОГЕНЕРАТОРА;
- F3BFh - содержит адрес таблицы ЦВЕТОВ для SCREEN 1;
- F3C9h - содержит адрес таблицы ЦВЕТОВ для SCREEN 2/4;
- F926h - содержит адрес таблицы ШАБЛОНОВ СПРАЙТОВ;
- F928h - содержит адрес таблицы АТРИБУТОВ СПРАЙТОВ;

- адрес таблицы ЦВЕТОВ СПРАЙТОВ всегда меньше адреса таблицы АТРИБУТОВ СПРАЙТОВ на 512.

К сожалению, адрес таблицы МЕРЦАНИЯ СИМВОЛОВ для режима SCREEN 0:WIDTH 80 нигде не содержится (хотя обычно он равен 00800h). В общем случае, недоступен и адрес таблицы ПАЛИТРЫ.

Поскольку VRAM MSX2 занимает обычно 128K, весь диапазон ее адресов недоступен микропроцессору Z80. По этой причине видеопамять разделена на несколько страниц, размеры которых (16K/32K/64K) определяются режимом VDP.

Номер ОТОБРАЖАЕМОЙ страницы VRAM (которая должна быть видна на экране) выставляется так:

```
    ld    (0FAF5h),a    ; номер страницы в A
    ld    ix,013Dh    ; процедура SETPAG
    ld    iy,(0FAF7h)
    call    001Ch        ; вызвать из SUB-ROM
```

Номер АКТИВНОЙ страницы VRAM (с которой должны работать графические процедуры BIOS) можно указать так:

```
    ld    (0FAF6h),a    ; номер страницы в A
```

Кстати, адреса всех таблиц отсчитываются от начала этой страницы.

Режимы видеопроцессора

Выбор желаемого режима осуществляется тривиально:

```
  ld    a,5        ; например, SCREEN 5
  ld    (0FCAFh),a    ; сохранить режим в SCRMOD
  ld    ix,005Fh    ; процедура CHGMOD
  ld    iy,(0FCC0h)    ; вызвать из MAIN-ROM
  call    001Ch
```

Поскольку SCREEN 0 существует в модификациях WIDTH 40 и WIDTH 80 (для VDP они принципиально различаются), текстовый режим можно включать так:

```
  ld    a,40        ; или 80
  ld    (0F3AEh),a    ; сохранить в LINL40
  ld    ix,006Ch    ; процедура INITXT
  ld    iy,(0FCC0h)    ; вызвать из MAIN-ROM
  call    001Ch
```

Не имея возможности уделить внимание всем режимам VDP, мы рассмотрим три наиболее популярных, сделав относительно остальных беглые замечания (в скобках приведены наименования режимов по стандарту MSX-VIDEO):

SCREEN 0:WIDTH 80 (TEXT 2)

Основной текстовый режим MSX2.
Формат экрана 80 x 26.5 символов, символы 6 x 8 точек.
Спрайты запрещены.
Начертание символов задается в таблице знакогенератора, размер которой равен 2048 байт, по 8 на символ. Каждый байт в двоичном представлении описывает одну горизонтальную линию символа: "единицы" задают очертания шрифта, "нули" - всегда прозрачные, причем в данном режиме два младших бита на экран не выводятся. Все символы получают номера от 0 до 255.

```
байт 00 ■■■■···· = 11110000b = F0h
байт 01 ■···■··· = 10001000b = 88h
байт 02 ■···■··· = 10001000b = 88h
байт 03 ■■■■···· = 11110000b = F0h
байт 04 ■·■····· = 10100000b = C0h
байт 05 ■··■···· = 10010000b = 90h
байт 06 ■···■··· = 10001000b = 88h
байт 07 ·····0·· = 00000000b = 00h
и т.д.
```

Внешний вид экрана определяется таблицей имен, занимающей в нашем случае 2160 байт (80 x 27). Каждый ее байт содержит номер (ASCII-код) символа, который должен появиться в соответствующем знакоместе экрана.
Регистр 7 VDP задает цвет знакогенератора: его старший полубайт определяет цвет "единиц", младший - "нулей".

```
  ld    b,0F4h        ; цвет "1" = 15, "0" = 4
  ld    c,7        ; номер регистра VDP
  ld    ix,0047h    ; процедура записи в VDP
  ld    iy,(0FCC0h)    ; вызвать из MAIN-ROM
  call    001Ch
```

Первые две команды обычно объединяют: LD BC,0F407h

Символы можно выделять мерцанием; для этого существует специальная таблица, каждый байт которой управляет мерцанием восьми соседних знакомест (по биту на штуку): отмеченные "1" знакоместа периодически получают цвета то из регистра 7, то из регистра 12 (формат которого такой же).
Период указывается в регистре 13: его младший полубайт определяет время нормального свечения (в 1/6 сек), старший - время альтернативного. Если нормальное время задать равным нулю, отмеченные знакоместа будут ПОСТОЯННО светиться цветом из регистра 12.
Соответствие между таблицами и экранными знакоместами самое естественное - построчное.

Режим SCREEN 0:WIDTH 40 (основной текстовый режим MSX), отличается меньшими размерами таблицы имен и отсутствием таблицы мерцания.

SCREEN 2 (GRAPHIC 2)

Основной графический режим MSX.
Формат экрана 32 x 24 знакоместа. Знакоместо 8 x 8.
Режим спрайтов 1 (о режимах см. ниже).
Экран делится на три равные горизонтальные области, и в пределах каждой из них действуют: собственный знакогенератор и собственная таблица имен с форматом 32 x 8.
Для раскраски символов предусмотрены три таблицы цветов (по одной на знакогенератор). Каждый байт этих таблиц задает цвет соответственной ЛИНИИ символа в том же формате, что и регистр 7. Таким образом, любая горизонтальная линия символа может содержать не более двух цветов! Конечно, столь сильное ограничение не проходит бесследно: графика MSX явно тяготеет к кубизму.
Все три знакогенератора представляют собой части единой таблицы, имеющей длину 6144 байта; то же касается и таблиц цветов.

Режим SCREEN 1 аналогичен SCREEN 2, но имеет один общий знакогенератор и одну таблицу цветов, каждый байт которой задает цвет (в формате регистра 7 VDP) сразу восьми подряд идущих символов.

Режим SCREEN 4 - это SCREEN 2 с режимом спрайтов 2.

SCREEN 5 (GRAPHIC 4)

Основной графический режим MSX2 (бит-мэп).
Формат экрана 256 x 212 точек.
На экране 16 цветов из 512 возможных.
Режим спрайтов 2 (о режимах см. ниже).
Длина страницы VRAM 32768 байт, число страниц 4.
Изображение построчно определено в таблице имен, каждый байт которой содержит данные о цвете двух соседних точек (по 4 бита на каждую). SCREEN 5 является наиболее оптимальным из режимов, т.к. в нем достаточно богатая палитра и приемлемая дискретность сбалансированы с объемом требуемой памяти.

SCREEN 6 занимает такой же объем памяти, но каждый байт таблицы имен содержит сведения о 4 точках, т.е. дискретность

составляет 512 x 212 при четырехцветной палитре. Этот режим наилучшим образом подходит для "делового софта": настольных издательств, электронных таблиц и пр.

SCREEN 7 (512 x 212, 16 цветов, 2 стр. по 65536 байт) - сочетает в себе качества SCREEN 5 и SCREEN 6, но он слишком расточителен в смысле требуемой памяти, поэтому используется довольно редко.

SCREEN 8 (256 x 212, 256 цветов, 2 стр. по 65536 байт) - режим с предельными возможностями и простейшей структурой: каждой точке экрана соответствует отдельный байт, который задает цвет "напрямую" (в формате RGB):

- биты 7,6,5 - яркость ЗЕЛЕНОЙ составляющей;
- биты 4,3,2 - яркость КРАСНОЙ составляющей;
- биты 1 и 0 - яркость СИНЕЙ составляющей.

В данном режиме палитра полностью игнорируется (в том числе и спрайтами).

Спрайты

Давайте вспомним, как снимаются рисованные мультфильмы: вначале перед объективом кинокамеры помещается фон, затем на него накладываются прозрачные листы с рисунками персонажей. Такой способ позволяет совершенно произвольно перемещать или заменять персонажи, не затрагивая фона.

MSX-VIDEO моделирует этот процесс во всех подробностях:

- Фон готовится в таблице знакогенератора и/или таблице имен;
- подвижные объекты - СПРАЙТЫ - располагаются по одному в 32-х "прозрачных плоскостях" (0 - ближайшая к наблюдателю, 31 - к фону) и могут независимо перемещаться по экрану.

Спрайты могут быть четырех видов:

- размер 0: 8x8 точек
- размер 1: 8x8 точек увеличенный в 2 раза
- размер 2: 16x16 точек
- размер 3: 16x16 точек увеличенный в 2 раза

Размер относится одновременно ко всем спрайтам; его рекомендуется устанавливать непосредственно после режима VDP:

```
    ld    b,a         ; A = размер (0...3) спрайта
    ld    a,(0F3E0h)
    and    11111100b     ; выставить размер в двух...
    or    b         ; ...младших битах
    ld    (0F3E0h),a
    ld    b,a
    ld    c,1         ; записать в регистр #1
    ld    ix,0047h     ; процедура записи WRTVDP
    ld    iy,(0FCC0h)     ; вызвать из MAIN-ROM
    call    001Ch
```

VDP поддерживает два режима спрайтов. Режим 1 совместим с MSX; его изобразительные возможности недостаточно богаты, поскольку на всю площадь спрайта распространяется один цвет, а на одной горизонтали экрана может находиться не более 4-х спрайтов.

Режим 2 гораздо интереснее: каждая линия спрайта может иметь собственный цвет, который указывается в таблице ЦВЕТОВ СПРАЙТОВ (по 16 байт на каждую плоскость). Кроме этого, на одной горизонтали экрана допускается до 8 спрайтов.

Внешний вид персонажей описывается в таблице шаблонов, аналогичной таблице знакогенератора. Поскольку спрайт 16x16 эквивалентен четырем 8x8, его описание в четыре раза длиннее (сначала задается левая половина изображения, затем правая). Таким образом, таблица может определять 256 спрайтов 8x8 или 64 спрайта 16x16.

Распределение изображений по плоскостям задает таблица атрибутов, которая содержит 32 четверки байтов, относящихся к плоскостям 0...31. Формат каждой четверки следующий:

- байт 0: координата Y левого верхнего угла спрайта;
- байт 1: координата X левого верхнего угла спрайта;
- байт 2: номер шаблона из таблицы;
- байт 3: 4 младших бита задают цвет "единиц" шаблона.

Чтобы вывести спрайт на экран, следует заполнить данными требуемую четверку. Для перемещения спрайта в другую точку экрана или смены его внешнего вида достаточно вписать в соответствующий байт новое значение. Временно удалить спрайт можно, поместив его "под нижний край" экрана (Y > 211).

Важно знать: координатное пространство спрайтов смещено относительно фоновых координат вниз на 1 (фоновой координате Y = 0 соответствует спрайтовая Y = -1 и пр.), а дискретность по горизонтали в любом режиме VDP составляет 256.

И последнее: номера шаблонов для спрайтов размера 16x16 задаются с точностью до 4 (например: 0, 4, 8, 12, ...).

Полную инициализацию спрайтов осуществляет процедура:

```
  ld    ix,0069h    ; INISPR
  ld    iy,(0FCC0h)
  call    001Ch
```

Доступ к видеопамяти

Итак, экранное изображение почти полностью определяется состоянием видеопамяти. Это означает, что практически любую операцию над фоном и спрайтами можно осуществить, располагая всего двумя элементарными процедурами: записью байта во VRAM и чтением байта из VRAM.

```
0177h/MAIN - запись одного байта во VRAM
вход: (FAF6h) - номер страницы VRAM
      HL - адрес внутри страницы
      A - значение для записи

0174h/MAIN - чтение одного байта из VRAM
вход: (FAF6h) - номер страницы VRAM
      HL - адрес внутри страницы
выход: A - прочитанное значение
```

К сожалению, каждое обращение к описанным подпрограммам сопряжено с межслотовым вызовом и настройкой регистров VDP, так что пересылка серии байтов путем цикличного повтора этих процедур оказывается недостаточно эффективной.

Перечисленные ниже входные точки ускоряют доступ к VRAM во много раз (за счет однократного обращения к BIOS):

```
016Bh/MAIN - заполнение вектора VRAM константой
вход: (FAF6h) - номер страницы VRAM
      HL - адрес внутри страницы
      BC - длина вектора
```

```
      A  - константа

016E/MAIN - настройка VDP на чтение серии байтов из VRAM
вход: (FAF6h) - номер страницы VRAM
     HL - адрес внутри страницы, с которого начнется чтение.
   Чтение данных ведется через порт, номер которого указан
в MAIN-ROM по адресу 0006h (обычно это порт 98h).

0171h/MAIN - настройка VDP на запись серии байтов во VRAM
вход: (FAF6h) - номер страницы VRAM
     HL - адрес внутри страницы, с которого начнется запись.
   Запись данных ведется через порт, номер которого указан
в MAIN-ROM по адресу 0007h (обычно это тот же порт 98h).

   В качестве примера рассмотрим пересылку вектора из  RAM
во VRAM, которую логично назвать LDIRVM:

; (FAF6h) <- номер страницы VRAM
; HL <- адрес внутри страницы
; DE <- адрес RAM
; BC <- длина пересылаемого массива (0 < BC <= 65535)

LDIRVM: push    de         ; сохранить адрес RAM
    push    bc         ; сохранить счетчик длины

    ld    ix,0171h    ; настроить VDP на прием
    ld    iy,(0FCC0h)
    call    001Ch

    ld    hl,0007h
    ld    a,(0FCC1h)    ; номер порта...
    call    000Ch        ; ...прочитать из MAIN-ROM
    ld    c,a         ; передать его в C

    pop    de         ; длину поместить в DE
    pop    hl         ; адрес RAM поместить в HL

ldvm00: ld    a,(hl)         ; взять очередной байт
    out    (c),a        ; передать его во VRAM
    inc    hl         ; перейти к следующему байту

    dec    de         ; уменьшить счетчик длины
    ld    a,e
    or    d        ; проверить его на 0
    jr    nz,ldvm00    ; повторить цикл
    ret            ; выйти, если выполнено

   Обратную процедуру разработайте самостоятельно. Кстати,
если Вы догадаетесь приспособить инструкции OTIR и INIR, Вас
ждет еще больший выигрыш по скорости!
```

Знакоместная графика

```
   Рассмотрим пример простой программы, которая заполняет
верхнюю треть экрана изображением домика:

; Присвоение числовых значений символическим именам:

MODE2   equ    2    ; SCREEN 2
EXBRSA  equ    0FAF8h    ; номер слота SUB-ROM
EXPTBL  equ    0FCC1h    ; номер слота MAIN-ROM
CALSLT  equ    001Ch    ; межслотовый вызов подпрограммы
```

```
SCRMOD  equ    0FCAFh    ; текущий режим VDP
CHGMOD  equ    005Fh    ; процедура установки режима VDP
ACPAGE  equ    0FAF6h    ; номер активной страницы
CGPBAS  equ    0F924h    ; адрес таблицы знакогенератора
GRPCOL  equ    0F3C9h    ; адрес таблицы цветов
NAMBAS  equ    0F922h    ; адрес таблицы имен
LENGTH  equ    8    ; длина графданных в байтах
PART    equ    256    ; размер экранной трети

    ld    a,MODE2
    call    SCREEN        ; установить SCREEN 2

    xor    a
    ld    (ACPAGE),a    ; работаем со страницей 0

    ld    hl,(CGPBAS)    ; адрес знакогенератора
    ld    de,DATA1    ; адрес изображения домика
    ld    bc,LENGTH    ; длина 8 байт
    call    LDIRVM        ; переслать во VRAM

    ld    hl,(GRPCOL)    ; адрес таблицы цветов
    ld    de,DATA2    ; адрес расцветки домика
    ld    bc,LENGTH    ; длина 8 байт
    call    LDIRVM        ; переслать во VRAM

    ld    hl,(NAMBAS)    ; адрес таблицы имен
    ld    bc,PART        ; одна треть
    xor    a        ; символ номер 00h
    call    BIGFIL        ; заполнить экран

LOCK:   jr    LOCK    ; зациклить программу

; Подпрограмма установки любого режима VDP
; вход: A <- номер режима

SCREEN: ld    (SCRMOD),a    ; поместить в SCRMOD
    ld    ix,CHGMOD
    jp    MAINRM        ; вызвать из MAIN-ROM

; Подпрограмма пересылки вектора из RAM во VRAM
; (ACPAGE) <- номер страницы VRAM
; HL <- адрес внутри страницы
; DE <- адрес RAM
; BC <- длина пересылаемого массива (0 < BC <= 65535)

LDIRVM: ...            ; см. предыдущий раздел

; Подпрограмма заполнения VRAM константой
; (ACPAGE) <- номер страницы VRAM
; HL <- адрес внутри страницы
; BC <- длина (0 < BC <= 65535)
; A <- константа

BIGFIL: ld    ix,016Bh
    jp    MAINRM

; Вызов подпрограммы из MAIN-ROM
; вход: IX <- адрес подпрограммы

MAINRM: ld    iy,(EXPTBL-1)    ; номер слота MAIN-ROM
    jp    CALSLT

; Вызов подпрограммы из SUB-ROM
; вход: IX <- адрес подпрограммы
```

```
SUBROM: ld    iy,(EXBRSA-1)    ; номер слота SUB-ROM
    jp    CALSLT

; Изображение домика:

DATA1:  defb    00111100b
    defb    01111110b    ; крыша
    defb    11111111b
    defb    11111110b    ; стена
    defb    11000110b
    defb    11000110b    ; окно
    defb    11111110b
    defb    11010110b    ; трава

; Раскраска домика:

DATA2:  defb    064h    ; красная крыша на синем фоне
    defb    064h
    defb    064h
    defb    0FEh    ; белая стена с серым оттенком
    defb    0F7h
    defb    0F7h    ; голубое окно
    defb    0FEh
    defb    02Ah    ; зеленая трава с желтыми цветами

    END
```

Программа зациклена, поэтому для возврата в DOS нужно нажать кнопку RESET. Чтобы охватить одним общим изображением все три части экрана, следует загрузить одинаковые наборы символов в каждый знакогенератор. Естественно, подпрограмма BIGFIL взята лишь для примера: обычно вместо нее используют LDIRVM, которая мгновенно "вбрасывает" в таблицу имен вектор из 768 байтов (32 x 24), кодирующий экранное изображение.

Очевидно, знакоместная графика плохо приспособлена для рисования точками, отрезками или окружностями. Полиэкранные изображения разрабатываются целиком в специальных редакторах (наиболее популярны многочисленные версии редактора GRED), а затем "подклеиваются" к программе при помощи команды COPY с ключом /b. Важно помнить, что "склеивание" осуществляется по границе дисковых блоков, поэтому адрес графданных необходимо скорректировать:

```
TRAIL:  equ    $        ; последняя метка программы
DATA:   ((TRAIL+127)/128)*128    ; адрес подключенных данных
```

Если Вы пользуетесь ассемблером M80, добавьте к формуле слагаемое +256.

Графика высокого разрешения

В отличие от знакоместной графики, режимы бит-мэп менее экзотичны, но обладают несравненно большими изобразительными возможностями. Наряду с уже известным Вам прямым доступом к VRAM, здесь можно использовать встроенные команды MSX-VIDEO, обеспечивающие практически весь "нижний уровень" графики.

Эти команды рассматривают видеопамять как одну большую координатную плоскость (разбиение на страницы игнорируется):

- в режиме SCREEN 5: 256 x 1024
- в режиме SCREEN 6: 512 x 1024
- в режиме SCREEN 7: 512 x 512

- в режиме SCREEN 8: 256 x 512

В процедурах, с которыми мы познакомимся ниже, нередко требуется так называемая "логическая операция", выполняемая над ИСХОДНЫМ цветом точки и НОВЫМ цветом, который несет этой точке команда VDP (обозначены соответственно DC и SC):

код 00000000b: DC = SC
код 00000001b: DC = DC and SC
код 00000010b: DC = DC or SC
код 00000011b: DC = DC xor SC
код 00000100b: DC = not SC

(если в этих кодах выставить бит 3, нулевой цвет будет рассматриваться как прозрачный, что исключительно удобно при наложении фрагментов со сложными очертаниями).

Для запуска команд рекомендуется обращаться к MSX-VIDEO напрямую, чтобы с наибольшей отдачей использовать скоростные возможности видеопроцессора. К сожалению, эта задача слишком сложна для ознакомительного повествования, поэтому мы с Вами ограничимся только теми процедурами, которые предусмотрены в BIOS, смирившись с отсутствием некоторых возможностей. Итак:

008Dh/MAIN - вывод символа на графический экран
вход: A - ASCII-код символа
(FB02h) - код логической операции
(F3E9h) - цвет символа
После вывода позиция сдвигается вправо на 8 точек.
Начальные координаты можно указать в ячейках:
(FCB3h) и (FCB7h) - координата X
(FCB5h) и (FCB9h) - координата Y

ЗДЕСЬ И ДАЛЕЕ ВСЕ КООРДИНАТЫ - ДВУХБАЙТОВЫЕ!

00CDh/SUB - нарисовать закрашенный прямоугольник
вход: BC - X1
DE - Y1, координаты одной из вершин
(FCB3h) - X2
(FCB5h) - Y2, координаты противоположной вершины
(F3F2h) - цвет
(FB02h) - логическая операция
(очевидно, эта же процедура позволит Вам рисовать ТОЧКИ и ОТРЕЗКИ, параллельные осям координат!)

00C9h/SUB - нарисовать прямоугольную рамку
вход: BC - X1
DE - Y1, координаты одной из вершин
(FCB3h) - X2
(FCB5h) - Y2, координаты противоположной вершины
(F3F2h) - цвет
(FB02h) - логическая операция

0191h/SUB - перенос прямоугольного фрагмента изображения
вход: (F562h) - SX
(F564h) - SY, левый верхний угол фрагмента во VRAM
(F566h) - DX
(F568h) - DY, координаты переноса во VRAM
(F56Ah) - NX
(F56Ch) - NY, размеры фрагмента
(F56Fh) - обнулить (1 байт)
(F570h) - логическая операция (1 байт)
HL - F562h

```
0195h/SUB - пересылка фрагмента из RAM во VRAM
вход: (F562h) - адрес массива графданных в RAM
      (F566h) - DX
      (F568h) - DY, координаты VRAM
      (F56Fh) - атрибут (1 байт). См. ниже
      (F570h) - логическая операция (1 байт)
      HL - F562h
   Массив должен иметь следующий вид:
   - первые 2 байта - горизонтальный размер фрагмента
   - следующие 2 байта - вертикальный размер фрагмента
   - далее следуют графические данные
выход: бит CARRY = 1, если при пересылке обнаружена ошибка

0196h/SUB - пересылка фрагмента из VRAM в RAM
вход: (F562h) - SX
      (F564h) - SY, координаты левого верхнего угла
      (F566h) - адрес массива в RAM
      (F56Ah) - NX
      (F56Ch) - NY, размеры фрагмента
      (F56Fh) - атрибут (1 байт). См. ниже
      HL - F562h
   Массив получает заголовок, описанный выше. Необходимый
размер массива можно вычислить по формуле: NX * NY / CC + 4,
где CC - количество точек в 1 байте таблицы имен.

   Две последние подпрограммы могут симметрично отображать
передаваемый фрагмент:
   - атрибут = 00h: фрагмент передается без изменений
   - атрибут = 04h: симметрия относительно оси Y
   - атрибут = 08h: симметрия относительно оси X
   - атрибут = 0Ch: симметрия относительно обеих осей

   Из всех перечисленных процедур наибольший практический
интерес представляет перенос фрагмента (0191h/SUB) из одной
точки VRAM в другую.
```

Мультипликационная графика

```
   Заставим двигаться по экрану небольшое изображение:

PATBAS  equ    0F926h    ; адрес таблицы шаблонов спрайтов
ATRBAS  equ    0F928h    ; адрес таблицы атрибутов спрайтов

    ld    a,5
    call    SCREEN        ; установить SCREEN 5

    xor    a
    ld    (0FAF6h),a    ; работаем со страницей 0
    call    SPRSIZ        ; размер спрайтов тоже 0
    call    INISPR        ; очистить VRAM от спрайтов

    ld    hl,(PATBAS)    ; адрес таблицы шаблонов
    ld    de,FACE1    ; адрес графданных
    ld    bc,16        ; длина графданных
    call    LDIRVM        ; упаковать изображение

    ld    hl,(ATRBAS)    ; адрес таблицы атрибутов
    dec    h
    dec    h        ; вычесть из него 512
    ld    de,FACE2    ; адрес цветовых данных
    ld    bc,32        ; длина
    call    LDIRVM        ; упаковать цвета
```

```
LOOP:   ei
    halt        ; задержка 1/50 сек
    ei
    halt        ; задержка 1/25 сек

    ld    hl,X0
    inc    (hl)     ; увеличить координату X
    ld    a,(hl)
    ld    (X1),a     ; скопировать ее во второй вектор

    xor    a     ; плоскость 0
    ld    de,Y0     ; первый вектор данных
    call    PUTSPR     ; поставить спрайт

    ld    a,1     ; плоскость 1
    ld    de,Y1     ; второй вектор данных
    call    PUTSPR     ; поставить спрайт

    jr    LOOP     ; повторить

; Вектора данных, расположенные в том же порядке, что
; и в таблице АТРИБУТОВ спрайтов:

Y0: defb    100     ; данные о первом спрайте
X0: defb    0
S0: defb    0     ; спрайт #0
C0: defb    0     ; (роли не играет)

Y1: defb    100     ; данные о втором спрайте
X1: defb    0
S1: defb    1     ; спрайт #1
C1: defb    0     ; (роли не играет)

; Подпрограмма установки любого режима VDP
; вход: A <- номер режима

SCREEN: ld    (0FCAFh),a     ; поместить в SCRMOD
    ld    ix,005Fh
    jp    MAINRM        ; вызвать из MAIN-ROM

; Подпрограмма выбора размера спрайтов
; вход: A <- размер

SPRSIZ: ld    b,a
    ld    a,(0F3E0h)
    and    11111100b
    or    b
    ld    (0F3E0h),a
    ld    b,a
    ld    c,1        ; "проваливается" в WRTVDP

; Подпрограмма записи в регистр VDP
; вход: C <- номер регистра, B <- данные

WRTVDP: ld    ix,0047h
    jp    MAINRM

; Подпрограмма инициализации спрайтов

INISPR: ld    ix,0069h
    jp    MAINRM

; Подпрограмма установки спрайта в заданную плоскость
; вход: A <- номер плоскости (0...31)
```

```
;       DE <- адрес RAM, по которому размещен вектор Y,X,S,C

PUTSPR: add    a,a
    add    a,a    ; номер плоскости умножить на 4
    ld    c,a
    ld    b,0        ; превратить в слово
    ld    hl,(ATRBAS)
    add    hl,bc         ; сложить с базовым адресом
    ld    bc,4        ; "проваливается" в LDIRVM

; Подпрограмма пересылки вектора из RAM во VRAM
; (ACPAGE) <- номер страницы VRAM
; HL <- адрес внутри страницы
; DE <- адрес RAM
; BC <- длина пересылаемого массива (0 < BC <= 65535)

LDIRVM: ...            ; см. выше

; Вызов подпрограммы из MAIN-ROM
; вход: IX <- адрес подпрограммы

MAINRM: ld    iy,(0FCC0h)    ; номер слота MAIN-ROM
    jp    001Ch

; Изображение спрайтов:

FACE1:  defb    00000000b
    defb    00000000b
    defb    00100100b    ; глаза
    defb    00000000b
    defb    00100100b    ; рот
    defb    00011000b
    defb    00000000b
    defb    00000000b

    defb    00111100b    ; прическа
    defb    01111110b
    defb    11111111b    ; лицо
    defb    11111111b
    defb    11111111b
    defb    11111111b
    defb    01111110b
    defb    00111100b

; Расцветка каждой линии спрайтов:

FACE2:  defb    0,0,12,0,1,1,0,0,0,0,0,0,0,0,0,0
    defb    6,6,11,11,11,11,10,10,0,0,0,0,0,0,0,0

    END
```

Управление палитрой

```
   Тот набор из 16 цветов, который предоставляется в Ваше
распоряжение после включения компьютера, Вы можете изменить
по собственному усмотрению. SUB-ROM содержит все необходимое
для управления палитрой:

0141h/SUB - восстановить стандартный набор цветов

0145h/SUB - упаковать таблицу палитры из VRAM в регистры VDP
```

```
0149h/SUB - получить описание цвета из таблицы палитры VRAM
вход: A - номер цвета 0...15
выход: BC в формате 0RRR 0BBB 0000 0GGGb, где R,G,B - биты
    яркости красной, зеленой и синей составляющей

014Dh/SUB - задать новое описание цвета в таблице VRAM
вход: D - номер цвета 0...15
       A в формате 0RRR 0BBBb
       E в формате 0000 0GGGb

   Динамичное управление палитрой позволяет добиться очень
выразительных видеоэффектов (плавное затемнение изображения,
дрожание контуров, мерцание, "тление" и пр.).
```

Дополнительные возможности

```
   Графический пакет BIOS этим далеко не исчерпывается.
MAIN-ROM содержит ряд входных точек более узкого назначения:

008Ah/MAIN - вычислить длину шаблона спрайта
выход: A - длина (8 для 8x8, 32 для 16x16)
    бит CARRY = 1, если спрайт 16x16

00A2h/MAIN - вывести символ на текстовый экран
вход: A - ASCII-код символа

00C6h/MAIN - задать позицию для курсора
вход: H - столбец, L - строка

00C3h/MAIN - очистить экран

0041h/MAIN - погасить дисплей
0044h/MAIN - включить дисплей
   Эта пара подпрограмм полезна в тех случаях, когда нужно
быстро подготовить экран, не посвящая зрителей в подробности
этого процесса.

00D2h/MAIN - возврат из графики в исходный текстовый режим

   Корректная установка цветов текста, фона и рамки экрана
также производится через BIOS (например, COLOR 15,4,7):

    ld    hl,0F3E9h
    ld    (hl),15         ; цвет текста 15 (FORCLR)
    inc    hl
    ld    (hl),4         ; цвет фона 4 (BAKCLR)
    inc    hl
    ld    (hl),7         ; цвет рамки 7 (BDRCLR)
    ld    a,(0FCAF)     ; уточнить режим VDP
    ld    ix,0062h     ; процедура CHGCLR
    ld    iy,(0FCC0h)     ; вызвать из MAIN-ROM
    call 001Ch


   Особо впечатляющих эффектов можно добиться, обращаясь к
MSX-VIDEO напрямую. Для этого необходимы две вспомогательные
процедуры BIOS:

0047h/MAIN - запись байта в регистр VDP
вход: B - данные, C - номер регистра

013Eh/MAIN - чтение из регистра статуса #0
```

выход: A - значение регистра

Кроме того, следует знать, что в системной области есть копии регистров VDP:
- регистры 0...7: ячейки F3DFh...F3E7h
- регистры 8...23: ячейки FFE7h...FFF6h

Поэтому выражение "выставить бит в таком-то регистре" означает, что нужно взять значение соответствующей системной ячейки, произвести с ним требуемое действие и записать его в VDP с помощью процедуры 0047h (которая автоматически зашлет новое значение и в регистр VDP, и обратно в ячейку).

Итак:

СМЕЩЕНИЕ ЭКРАНА. Запишите в регистр 18 VDP значение в следующем формате:
- старший полубайт: вертикальное смещение (-8...+7)
- младший полубайт: горизонтальное смещение (-8...+7)

ВЕРТИКАЛЬНЫЙ СКРОЛЛИНГ ИЗОБРАЖЕНИЯ. Последовательно запишите в регистр 23 числа от 225 до 0 (для вращения вверх) или от 0 до 255 (для вращения вниз). Изображение можно "подкачивать" на невидимой части страницы VRAM.

ПЕРЕКЛЮЧЕНИЕ ДИСКРЕТНОСТИ ПО ВЕРТИКАЛИ. Бит 7 регистра 9 позволяет выбрать вертикальный размер картинки:
- сбросить бит - 192 точки
- установить бит - 212 точек

УДВОЕНИЕ ВЕРТИКАЛЬНОЙ ДИСКРЕТНОСТИ (РЕЖИМ "INTERLACE"). Режим interlace (чересстрочная развертка) позволяет повысить вертикальную дискретность до 424, засчет того, что нечетные строки экрана будут выводиться с нечетной страницы, четные - с четной (ее номер на 1 меньше). Выберите нечетный номер для отображаемой страницы и установите биты 3 и 2 регистра 9.

АВТОМАТИЧЕСКАЯ СМЕНА СТРАНИЦ VRAM. Установите нечетный номер отображаемой страницы, затем в регистр 13 поместите время демонстрации страниц (в 1/6 сек):
- старший полубайт: время четной (номер на 1 меньше)
- младший полубайт: время нечетной

Выключение производится записью нуля в регистр 13.

ВЫКЛЮЧЕНИЕ ПРОЗРАЧНОГО ЦВЕТА. Установите бит 5 регистра 8. Нулевой цвет потеряет функции "прозрачного", и его можно будет использовать наравне с другими цветами.

ГЕНЕРАЦИЯ ПРЕРЫВАНИЙ В УКАЗАННОЙ СТРОКЕ ЭКРАНА. Эта функция позволяет разделить экран на две части: неподвижную и скроллирующую, текстовую и графическую и т.п. Номер строки раздела укажите в регистре 19, после чего установите бит 4 регистра 0. Теперь VDP, завершая вывод этой строки, будет генерировать запросы прерывания, подтверждая их старшим битом регистра статуса 0, который читается процедурой 013Eh. (Обработку прерываний см. ниже).

ОБНАРУЖЕНИЕ ЛИШНЕГО СПРАЙТА НА ОДНОЙ ГОРИЗОНТАЛИ. Если на одной горизонтали экрана расположено более 4-х (или 8-ми) спрайтов, бит 6 регистра статуса 0 принимает значение "1". Здесь же можно определить номер плоскости "лишнего" спрайта: он записан в пяти младших битах того же регистра.

ОБНАРУЖЕНИЕ СТОЛКНОВЕНИЯ СПРАЙТОВ. Когда два или более спрайтов соприкасаются своими значащими частями, это событие

регистрируется в пятом бите регистра статуса 0.

ВЫКЛЮЧЕНИЕ СПРАЙТОВ. Спрайты можно полностью запретить (это ускоряет работу VDP, так же как и гашение экрана). Для этого нужно установить бит 1 регистра 8.

Наконец, небольшое уточнение, касающееся цвета спрайта: обычно в цветовом байте используются только 4 младших бита, которые определяют цвет всего спрайта (или одной его линии). На самом же деле, биты старшего полубайта цвета имеют вполне определенное назначение:
- если установлен бит 7, спрайт выводится на 32 точки левее своих координат;
- если установлен бит 6, данный цвет смешивается по OR с цветом спрайта на "ближайшей сверху" плоскости;
- если установлен бит 5, спрайт не конфликтует.

Программируемый звуковой генератор (PSG)

Генератор звука AY-3-8910 состоит из следующих блоков:
- 3 генератора тона для каналов A, B и C
- 3 усилителя для каналов A, B и C
- общий генератор шума для всех каналов
- общий генератор формы огибающей для всех каналов
- общий микшер для всех каналов

Такая, казалось бы, незамысловатая система отличается удивительными способностями: подражает звучанию музыкальных инструментов (струнных, духовых, ударных и т.д.) и свободно имитирует разнообразнейшие природные и технические шумы.

Три независимых звуковых канала позволяют воспроизвести многие музыкальные эффекты: аккорды, реверберацию, звучание оркестра, иллюзию "пространства", глиссандо, вибрато и т.д.

Как и MSX-VIDEO, звуковой генератор работает независимо от центрального процессора и управляется примерно так же - с помощью 14 регистров, обращаться к которым нужно через BIOS:

```
00C0h/MAIN - сигнал BEEP

0090h/MAIN - полный сброс PSG

0093h/MAIN - запись в регистр PSG
вход: A - номер регистра, E - данные для записи

0096h/MAIN - чтение из регистра PSG
вход: A - номер регистра
выход: A - прочитанное значение
```

Синтез звука

ГЕНЕРАТОР ТОНА. Настройка частоты генератора начинается с подсчета коэффициента деления (он должен лежать в пределах от 0 до 4095):

K = 111860.78125 / f, где f - желаемая частота в Гц.

Теперь, в зависимости от того, какой из каналов выбран, поместите полученный коэффициент в регистры:
- A: регистр 0 <- младший байт, регистр 1 <- старший
- B: регистр 2 <- младший байт, регистр 3 <- старший
- C: регистр 4 <- младший байт, регистр 5 <- старший

ГЕНЕРАТОР ШУМА. Коэффициент деления для средней частоты шума вычисляется по той же формуле (он не должен выходить за границы диапазона 0...31). Полученное значение записывается в регистр 6 (и распространяется на все три канала).

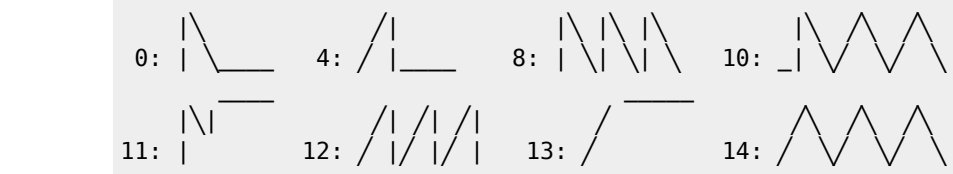
МИКШЕР управляет подключением генераторов к усилителям. Доступ к микшеру производится через регистр 7, который имеет формат `10cbaCBA`, где биты `cba` управляют шумом, а биты `CBA` - тональными генераторами (0 означает "включено").

УСИЛИТЕЛЬ. Каналам `A`, `B` и `C` соответствуют регистры 8, 9 и 10, куда записывается громкость звучания. Если она лежит в пределах 0...15, канал генерирует звук постоянной громкости. Если же значение равно 16 (точнее, установлен бит 4), данный усилитель управляется генератором огибающей.

ГЕНЕРАТОР ОГИБАЮЩЕЙ занимается тем, что управляет всеми каналами одновременно, изменяя их громкость по определенному графику. `PGS` содержит восемь таких графиков для разных форм огибающих. Номер графика выбирается с помощью регистра 13, который одновременно выполняет запуск генератора.

Одну и ту же огибающую можно "сжать" или "растянуть" во времени. Вычислив коэффициент деления по приведенной ниже формуле, запишите его в регистры 11 (младший байт) и 12:

`K = 143.03493 / t, где t -` период в микросекундах



Другие способы вывода звука

Стандартом `MSX2` предусмотрены еще два способа генерации звука, которые принципиально отличаются от описанного выше.

Первый, наиболее примитивный, является единственным для большинства персональных компьютеров; он основан на быстрой инверсии бита в одном из портов ввода/вывода. Поскольку этим вынужден заниматься `Z80`, все остальные процессы, в том числе и прерывания, приходится временно останавливать. Этот способ генерации звука обычно используют для вывода речи:

`0135h/MAIN -` управление звуковым битом
`вход: A = 0 -` сбросить бит, `A ≠ 0 -` установить бит

Второй путь заключается в использовании многоканального синтезатора `FM MSX-AUDIO`, основанного на принципах частотной модуляции. Это устройство поставляется дополнительно и дает оркестровое звучание почти профессионального качества.

Клавиатура

С точки зрения микропроцессора, клавиатура представляет собой таблицу (МАТРИЦУ) из 9 строк и 8 столбцов, элементами которой являются алфавитно-цифровые и функциональные клавиши (дополнительная цифровая клавиатура добавляет еще 2 строки).

0141h/MAIN - проверка строки клавиатурной матрицы
вход: A - номер строки
выход: A - состояние строки
(каждый бит, равный 0 означает нажатую клавишу)

Такой метод рекомендуется применять только для проверки функциональных клавиш (поскольку положение остальных зависит от экспортной версии компьютера):

```
строка 6:    F3     F2    F1  CODE  CAPS  GRAPH  CTRL  SHIFT
строка 7:  RETURN SELECT  BS  STOP  TAB    ESC    F5    F4
строка 8:    ►      ▼     ▲    ◄    DEL    INS   HOME  SPACE
строка 9:   4   3   2   1   0
строка 10:  .   ,   -   9   8   7   6   5
```

Алфавитно-цифровые клавиши целесообразнее определять по возвращаемому ASCII-коду:

009Fh/MAIN - возвращает код нажатой клавиши (с ожиданием)
выход: A - ASCII-код клавиши

Коды клавиш черпаются из специального буфера, куда они поступают по мере нажатия: это "смягчает" работу клавиатуры. Иногда возникает необходимость избавиться от накопившихся в буфере кодов; для этого предусмотрена точка BIOS 0156h/MAIN. Нередко бывает полезно знать, пуст ли буфер или уже содержит код какой-нибудь клавиши:

009Ch/MAIN - проверка состояния буфера
выход: бит Z = 1, если буфер пуст
Совместно с 009Fh эта процедура позволяет смоделировать INKEY$, подобно тому, как это делалось с функциями MSX-DOS.

Некоторые сложности возникают с графическими символами, поскольку многие из них кодируются словом: вначале идет байт 01h, за которым следует код графсимвола, увеличенный на 40h.

Для проверки нажатия CTRL STOP предусмотрена отдельная входная точка 00B7h/MAIN, которая возвращает бит carry, если было зарегистрировано нажатие.

Вы можете также поменять значения функциональных клавиш F1...F10, зная положение соответствующих системных ячеек:
- начало области F87Fh
- под каждую клавишу выделено 16 байт
- восстановление в исходное состояние 003Eh/MAIN

Альтернативные мануальные устройства

Бывают (и нередко) случаи, когда клавиатура оказывается не самым удобным средством для ввода информации: например, в играх или графических редакторах; в таких случаях клавиатуру заменяют другие устройства, среди которых наиболее популярны джойстик, мышь и световое перо:

ДЖОЙСТИК возвращает число 1...8, означающее НАПРАВЛЕНИЕ наклона рукоятки: 1 = "вперед", и далее - по часовой стрелке через каждые 45° (нейтральному положению соответствует 0).

00D5h/MAIN - возвращает направление рукоятки джойстика
вход: A - номер разъема джойстика (0, 1, 2)
выход: A - направление

Процедура способна опрашивать клавиши курсора ("джойстик" 0)

00D8h/MAIN - возвращает состояние кнопок джойстика
вход: A - номер кнопки (0, 1, 2, 3, 4)
выход: A = 255, если нажата, иначе 0
Кнопки (ТРИГГЕРЫ) нумеруются следующим образом:
0 - клавиша пробела ("джойстик" 0)
1 - кнопка на рукоятке (разъем 1)
2 - кнопка на рукоятке (разъем 2)
3 - кнопка на подставке (разъем 1)
4 - кнопка на подставке (разъем 2)

МЫШЬ сообщает программе СМЕЩЕНИЕ по X и Y относительно своей предыдущей позиции (в виде чисел -128...127).
Чтобы опросить мышь, подключенную к разъему 1, вызовите трижды подряд процедуру 0DBh/MAIN, помещая в регистр A числа 12, 13 и 14. Первый вызов обеспечит измерение смещения мыши, а второй и третий возвратят приращение координат по X и Y (в регистре A).
Разъем 2 опрашивается с аргументами 16, 17 и 18. Доступ к клавишам мыши аналогичен опросу кнопок джойстика (клавиша DO эквивалентна кнопке на рукоятке).

СВЕТОВОЕ ПЕРО возвращает в программу свои КООРДИНАТЫ.
Доступ к световому перу аналогичен контролю мыши: точка 00DBh должна вызываться 4 раза подряд со значениями регистра A = 8, 9, 10 и 11. Первый вызов определяет положение пера на экране и, в случае успеха, возвращает 255, а два последующих сообщают координаты X и Y. Последний вызов предназначен для определения состояния кнопки пера (255 - нажата, 0 - нет).
Аналогично опрашивается и графический планшет (значения 0,1,2,3 для разъема 1 и 4,5,6,7 для разъема 2).

Существуют и более экзотические приспособления: игровые манипуляторы PDL, музыкальные клавиатуры, системы типа CARD READER, тач-панели и пр. Принципы их обработки, как правило, даются в руководствах к этим изделиям.

MSX-принтер

Вывод на принтер помогают осуществить две процедуры:

00A8h/MAIN - возвращает Z = 1, если принтер отключен

00A5h/MAIN - выводит символ на принтер
вход: A - ASCII-код символа
выход: CARRY = 1, если были нажаты клавиши CTRL STOP

Кроме этого, в системной области предусмотрены ячейки:

F417h - переключает вариант принтера (0 = MSX, ≠0 - другой)
F418h - должна содержать 0 при печати текстов и 1 при выводе графики (управляет выводом кода 09h).

MSX-принтер в обязательном порядке "понимает" следующие управляющие коды (минимальный набор):
09h - табуляция
0Ah - подача листа на 1 строку
0Ch - выброс листа
0Dh - возврат каретки к левой границе
1Bh, 'A' - нормальный интервал между строками
1Bh, 'B' - нулевой интервал (для вывода графики)

1Bh, 'Snnnn' - печать графической строки (nnnn - длина)
Естественно, реальный перечень кодов гораздо богаче, он детально описывается в руководстве к принтеру.

Биткордер

Сейчас магнитофонная кассета, как средство для хранения программ, кажется архаизмом, но всего несколько лет назад магнитофон (биткордер) был единственным устройством внешней памяти, доступным широкому кругу пользователей.
Поэтому неудивительно, что стандарт MSX весьма детально проработан в этом смысле:

00E1h/MAIN - приготовить компьютер к чтению файла
выход: бит carry = 1, если ошибка

00E4h/MAIN - считать один байт
выход: A - прочитанный байт
бит carry = 1, если ошибка

00E7h/MAIN - завершить чтение файла

00EAh/MAIN - приготовить компьютер к записи файла
вход: A - тип хэдера (0 - короткий, 1 - длинный) см. ниже
выход: бит carry = 1, если ошибка

00EDh/MAIN - записать один байт
вход: A - байт для записи
выход: бит carry = 1, если ошибка

00F0h/MAIN - завершить запись файла

00F3h/MAIN - управление мотором биткордера
вход: A - код: 0 - стоп, 1 - старт, 255 - инверсия состояния

Важно понимать, что процедуры чтения/записи критичны по времени (магнитная лента не ждет, она движется), поэтому какая-либо обработка данных между вызовами процедур 00E4h или 00EDh, кроме обмена с RAM, недопустима.

Кассетные файлы отличаются от дисковых своеобразным форматом:
- длинный "хэдер" (фрагмент, заполненный тоном заданной частоты, предназначенный для выравнивания скорости мотора и настройки процедуры чтения);
- 10 байтов, определяющих тип файла (D3h для BASIC, D0h для BLOAD и EAh для ASCII);
- имя файла (6 символов);
- пауза;
- тело файла (формат зависит от типа файла);

Разные типы файлов представляются на ленте в различном формате. Наиболее прост формат BASIC-программ:
- короткий хэдер
- тело программы любой длины
- 7 байтов 00h

BLOAD-программы имеют другой вид:
- короткий хэдер
- начальный адрес загрузки (2 байта)
- конечный адрес загрузки (2 байта)
- стартовый адрес (2 байта)

- тело программы

Наиболее сложен формат ASCII-файлов (текстов и данных): файл разбивается на блоки по 256 байт, каждый из которых предваряется коротким хэдером. Конец файла отмечается байтом ^Z (1Ah = 26) в последнем блоке.

Обработка прерываний

Таймер компьютера обеспечивает прерывания с частотой 50 (иногда 60) герц, которые удобны для тактирования различных процессов в программе. Существуют три способа подключения к таймеру.

Первый из них Вам уже знаком (мы пользовались им, когда двигали спрайты по экрану):

```
ei    ; разрешить прерывания
halt    ; приостановиться
```

Поскольку Z80 останавливается в полном смысле слова, все процессы замораживаются до очередного сигнала таймера, что не всегда позволительно.

Второй путь помогает отделить от остальных тот процесс, который должен исполняться по таймеру. В системной области расположено слово, которое увеличивается на 1 каждые 0.02 с: FC9Eh. Вашей программе остается как можно чаще проверять, не изменилось ли значение данной ячейки и, если да, то вызывать тактируемую процедуру. Время между проверками можно посвятить текущим задачам.

Наиболее сложный, но корректный во всех отношениях путь основан на подключении к резидентной процедуре BIOS, которая 50 раз в секунду вызывает, как подпрограмму, адрес FD9Fh.

В обычном состоянии по этому адресу записан код команды RET (C9h), поэтому процессор немедленно возвращается, но Вы имеете право поместить по этому адресу вызов Вашей процедуры (под этот вызов отводится 5 байт).

Корректное построение вызова достаточно нетривиально: Ваша программа должна определить слот, в котором находится тактируемая процедура, и поместить по адресу FD9Fh следующую конструкцию из 5 байт:

```
rst    30h    ; межслотовый вызов
defb    slot    ; номер слота в формате F000SSPPb
defw    address    ; адрес процедуры
ret
```

Необходимость в межслотовом вызове обусловлена тем, что конфигурация памяти компьютера в этот момент изменяется.

Точно так же можно подключиться к обработке прерываний, поступающих от внешних устройств (например, от VDP). В этом случае адрес равен FD9Ah.

Вообще, подобные адреса, именуемые "хуками" (hook) есть у многих подпрограмм BIOSа и BASICа (всевозможные RAM-диски и вирусы с их помощью перехватывают обращения к периферии, подменяя их другими операциями), но они используются реже.

К большому сожалению, среди программистов MSX широчайше распространено заблуждение, будто приведенная выше конструкция предназначена для обращения к входным точкам BIOS. Ошибка налицо - данный способ вызова содержит только константы,

следовательно, он не обладает должной гибкостью и может привести к краху на машинах с необычной конфигурацией.

И последнее, что касается использования хуков. Вполне возможно, что какая-либо из резидентных программ уже заняла хук, к которому Вы собираетесь подключиться. Как поступить в такой ситуации? Уничтожать уже размещенный вызов нельзя, ибо неизвестны последствия столь грубой операции.
Остается одно - сделать копию хука и только после этого помещать в него свой вызов. Что касается подпрограммы обработки, то она должна обеспечить передачу управления полученной копии.
Таким образом, обе программы будут удовлетворены.

Часы и календарь

Доступ к часам и календарю проще всего осуществить при помощи функций BDOS:

```
2Ah - получить значение текущей даты
выход: HL - год
       D  - месяц
       E  - день месяца
       A  - день недели

2Bh - установить новую дату
вход:  HL - год
       D  - месяц
       E  - день месяца
выход: A = 0, если успешно

2Ch - получить текущее время
выход: H - часы
       L - минуты
       D - секунды
       E - сотые доли секунд

2Dh - установить новое время
вход:  H - часы
       L - минуты
       D - секунды
       E - сотые доли секунд
выход: A = 0, если успешно
```

Системная область MSX2

Системная область располагается в третьей странице RAM, начиная с адреса F380h и содержит обширный набор переменных, которые используются BIOS и MSX BASIC–интерпретатором. Многие из этих переменных Вам уже хорошо известны.

Перед системной областью, в этой же странице, размещены рабочее пространство DOS и аппаратный стек, размеры которых могут изменяться, поэтому предельный адрес пользовательского пространства нужно уточнять с помощью ячейки 0006h.

Программист должен следить, чтобы системная страница не отключилась в результате некорректного обращения со слотами или мэппером, иначе это приведет к зависанию компьютера.

Заключение для пользователей MSX

Весь изложенный в нашей беседе материал можно с успехом применить к программированию компьютеров серии MSX, если из предыдущего повествования исключить следующие моменты:

- процедуры, находящиеся в SUB-ROM;
- рассуждения, касающиеся мэппера и его регистров;
- режимы битмэп и текстовый SCREEN 0:WIDTH 80;
- режим спрайтов 2;
- все, что связано со страницами видеопамяти;
- операции, использующие регистры VDP, начиная с 9-го;

Кроме того, для обращения к VRAM следует использовать другие подпрограммы BIOS (с теми же аргументами):

004Ah	чтение байта из VRAM
004Dh	запись байта во VRAM
0050h	подготовка к чтению серии байтов
0053h	подготовка к записи серии байтов
0056h	заполнение VRAM константой

https://sysadminmosaic.ru/msx/msx2_programming_technique/msx2_programming_technique

2023-02-04 23:28