# ВВЕДЕНИЕ

Компьютеры серии "Ямаха MSX-2" обладают весьма разнообразными возможностями в сравнении с другими компьютерами аналогичной стоимости. Гибкость архитектуры MSX-2 позволяет решать такие задачи, как

- Обработка текстов и передача данных

  В распоряжении пользователя имеются разнообразные программы текстовых редакторов для различных языков (английского, французского, немецкого, арабского, русского, японского и ряда других). Применение интерфейса RS232C позволяет связать компьютеры через модемы и канал телефонной линии.

- Создание и исполнение музыкальных произведений

  Применение модуля звуковых эффектов и необходимых программных средств превращает компьютер в электромузыкальный синтезатор с дополнительными возможностями ввода, редактирования, хранения на магнитных носителях и вывода на принтер созданных Вами мелодий.

- Машинная графика и обработка изображений

  Программные средства компьютера - такие, как программа "ПЕЙНТЕР" позволяют создавать весьма сложные рисунки, которые можно записывать на дискеты и выводить на принтер. Кроме того, для компьютеров MSX-2 разработана аппаратура ввода изображений, поступающих от внешнего источника видеосигналов; эти изображения в цифровой форме могут быть обработаны и записаны на дискеты.

- Игры

  Для компьютеров MSX разработаны и тиражируются сотни игровых программ, записанных на магнитных носителях или в специальных кассетах с постоянной полупроводниковой памятью.

- Изучение основ программирования

  Язык программирования Бейсик был разработан специально для изучения основ программирования. Впоследствии он занял ведущее положение среди языков, применяемых на персональных компьютерах. Язык MSX-Бейсик, оставаясь доступным для начинающих, предоставляет в то же время все средства доступа к аппаратуре компьютера, необходимые специалисту.

- Выполнение научно-технических расчетов

  В интерпретаторе языка MSX-Бейсик имеется библиотека стандартных программ математических функций, благодаря которой многие расчеты могут быть выполнены без помощи высокопроизводительных ЭВМ. Программы, не нуждающиеся для своей работы в большом объеме оперативной памяти и высоком быстродействии процессора, могут выполняться "в домашних условиях". Эти возможности дополнены описанными ранее возможностями обработки текстов и передачи данных.

Все это разнообразие применений обеспечивается особенностями архитектуры и конструкции компьютеров, отвечающих требованиям стандарта MSX. Разъемы на корпусе компьютера позволяют подключать модули дополнительной оперативной памяти (RAM), кассеты с программами в постоянной памяти (ROM), модули электромузыкальных синтезаторов, адаптеры различных периферийных устройств...



В любом случае трудно представить себе ситуацию, когда пользователь персонального компьютера обходится без знания основ программирования. Программирование является важной составной частью компьютерной грамотности; только овладение программированием позволит Вам использовать все возможности компьютера.

В этом руководстве содержатся основные сведения о компьютере MSX. Изучение этих сведений позволит начинающим оценить весь диапазон возможностей персонального компьютера с тем, чтобы выбрать некоторые из них для более детального изучения.

Материал руководства построен по схеме последовательного усложнения; многочисленные примеры позволят Вам проверить на практике полученные знания.

Авторы стремились избежать свойственного многим пособиям по программированию "перечислительного" стиля изложения. Разумеется, построенные таким образом справочники удобны для профессионалов; однако начинающим в таком материале трудно ориентироваться. Поэтому весь справочный аппарат книги вынесен в приложения.

Диапазон рассматриваемых тем оказывается весьма широким от изучения правил работы с клавиатурой до приемов программирования на уровне команд микропроцессора. Для закрепления материала в руководстве подробно разобраны две программы: программа расчета биоритмов, написанная на Бейсике и иллюстрирующая технику "нисходящего" построения програм, и программа "монитор звуковых эффектов", в которой применяются короткие подпрограммы в машинных кодах.

# СОДЕРЖАНИЕ


**Глава 1. Начинаем осваивать MSX Бейсик ..... 1**

**1-1. Клавиатура и экран ..... 1**
1-1-1. Некоторые определения ..... 2
1-1-2. Изучаем клавиатуру ..... 3
1-1-3. Логическая и физическая строки ..... 7
**1-2. Язык программирования MSX Бейсик ..... 8**
1-2-1. Арифметические операции ..... 9
1-2-2. Числовые константы и переменные ..... 11
1-2-3. Символьные строки ..... 13
**1-3. Что такое программа? ..... 14**
1-3-1. Составление и использование программ ..... 15
1-3-2. Редактирование текста программы ..... 17
1-3-3. Операторы программы ..... 19
**1-4. Порядок выполнения операторов в программе .. 20**
1-4-1. Программные циклы ..... 20
1-4-2. Подпрограммы ..... 22
**1-5. Принятие решений в программе ..... 23**
1-5-1. Оператор IF... THEN.. ..... 23
1-5-2. Оператор IF... THEN... ELSE ..... 26
**1-6. Взаимодействие с работающей программой ..... 27**
1-6-1. Оператор INPUT ..... 27
1-6-2. Функция INKEY$ ..... 28
**1-7. Запись данных в программе ..... 31**
1-7-1. Операторы READ и DATA ..... 31
1-7-2. Оператор RESTORE ..... 33

**Глава 2. Константы, переменные и выражения ..... 35**

**2-1. Константы ..... 35**
2-1-1. Символьные константы ..... 35
2-1-2. Числовые константы с двойной точностью ..... 36
2-1-3. Числовые константы с одинарной точностью ..... 37
2-1-4. Целые числовые константы ..... 39
**2-2. Переменные ..... 40**
2-2-1. Типы переменных ..... 41
2-2-2. Массивы переменных ..... 43
2-2-3. Пространство хранения переменных в памяти ... 45
**2-3. Выражения ..... 47**
2-3-1. Операции ..... 48
2-3-2. Функции ..... 51

2-3-3. Функции, определяемые пользователем ..... 61
2-3-4. Точность числовых выражений ..... 62

**Глава 3. Редактирование программ** ..... 65

3-1. Просмотр текста программы ..... 65
3-2. Отладка программ ..... 67
3-3. Процедуры ввода и отладки ..... 70
3-4. Функциональные клавиши ..... 72

**Глава 4. Условные и безусловные переходы** ..... 75

4-1. Безусловные переходы ..... 75
4-1-1. Оператор GOTO ..... 75
4-1-2. Оператор GOSUB ..... 77
4-1-3. Еще о функциях, определяемых пользователем ..... 79
4-2. Условные переходы ..... 80
4-2-1. Оператор IF THEN ELSE ..... 80
4-2-2. Операторы ON GOTO и ON GOSUB ..... 83
4-2-3. Прерывания ..... 84
4-2-4. Циклы FOR NEXT ..... 85

**Глава 5. Работа с внешней памятью** ..... 89

5-1. Сохранение и загрузка программ, написанных на MSX Бейсике ..... 89
5-1-1. Сохранение программ ..... 89
5-1-2. Загрузка программы ..... 93
5-2. Файлы с последовательным доступом ..... 96
5-2-1. Открытие файла с последовательным доступом ..... 97
5-2-2. Запись в файл ..... 99
5-2-3. Чтение из файла ..... 99
5-2-4. Закрытие файла ..... 100
5-2-5. Определение конца файла ..... 101
5-3. Файлы с произвольным доступом ..... 102
5-4. Работа с файлами на дискетах ..... 105

**Глава 6. Работа с экраном** ..... 107

6-1. Введение ..... 107
6-1-1. Выбор режима экрана ..... 107
6-1-2. Компоненты изображения ..... 108
6-2. Режим SCREEN 0 ..... 109
6-2-1. Выбор цвета ..... 110
6-2-2. Отображение текста ..... 112
6-3. Режим SCREEN 1 ..... 121
6-3-1. Выбор цвета ..... 121
6-3-2. Отображение текста ..... 122
6-3-3. Спрайты ..... 123



6-4. Режим SCREEN 2 ..... 133
6-4-1. Выбор цвета ..... 133
6-4-2. Графические операторы ..... 134
6-4-3. Спрайты ..... 152
6-4-4. Отображение текста ..... 153
6-5. Режим SCREEN 3 ..... 155
6-5-1. Выбор цвета ..... 155
6-5-2. Спрайты ..... 156
6-5-3. Графические операторы ..... 156
6-5-4. Текстовое изображение ..... 158
6-6. Режим SCREEN 4 ..... 158
6-7. Режим SCREEN 5 ..... 161
6-7-1. Разрешение и выбор цвета ..... 162
6-7-2. Экранные страницы ..... 162
6-8. Режим SCREEN 6 ..... 168
6-8-1. Разрешение и выбор цвета ..... 168
6-8-2. Графические операторы ..... 169
6-8-3. Вывод текста ..... 170
6-8-4. Применение экранных страниц и видео ..... 170
6-9. Режим SCREEN 7 ..... 170
6-9-1. Разрешение и выбор цвета ..... 170
6-9-2. Вывод текста ..... 171
6-9-3. Применение экранных страниц ..... 171
6-10. Режим SCREEN 8 ..... 171
6-10-1. Разрешение и выбор цвета ..... 171
6-10-2. Применение экранных страниц ..... 172

**Глава 7. Обработка прерываний в программе пользователя** ..... 173

7-1. Обработка прерывания по событию ..... 173
7-1-1. Нажатие [CTRL] + [STOP] ..... 175
7-1-2. Прерывания в фиксированные интервалы времени ..... 177
7-1-3. Столкновения спрайтов ..... 178
7-1-4. Функциональные клавиши ..... 180
7-1-5. Триггеры ..... 180
7-1-6. Особенности одновременной обработки событий различных типов ..... 181
7-2. Обработка ошибок ..... 182

**Глава 8 Ввод с клавиатуры и с помощью манипуляторов** ..... 187

8-1. Операции ввода с клавиатуры ..... 187
8-1-1. INKEY$ ..... 187
8-1-2. INPUT$ ..... 189

8-1-3. INPUT ..... 190
8-1-4. LINE-INPUT ..... 191
8-1-5. STRIG(0) ..... 192
8-1-6. STICK(0) ..... 193
**8-2. Манипуляторы** ..... 193
8-2-1. STRIG ..... 194
8-2-2. STICK ..... 194
8-2-3. PAD и PDL ..... 195

**Глава 9. Музыкальные возможности MSX Бейсика** 197

**9-1. Оператор PLAY** ..... 197
9-1-1. Высота звука ..... 198
9-1-2. Длительность нот ..... 201
9-1-3. Паузы (R) ..... 202
9-1-4. Динамические отметки ..... 202
9-1-5. Специальные эффекты ..... 203
9-1-6. Использование переменных ..... 203
9-1-7. Подпрограммы ..... 204
9-1-8. Работа с продолжительными звуковыми партиями ..... 205
9-1-9. Прекращение воспроизведения звука ..... 206
**9-2. Оператор SOUND** ..... 206
9-2-1. Регистры, устанавливающие частоту звука ..... 206
9-2-2. Генератор шумов ..... 208
9-2-3. Микширование каналов ..... 209
9-2-4. Установка громкости ..... 209
9-2-5. Генератор огибающей ..... 210
9-2-6. Использование оператора SOUND ..... 211

**Глава 10 Инициализация в MSX Бейсике** ..... 215

**Глава 11 Краткие сведения о численных методах** ... 221

**11-1. Численное интегрирование** ..... 221
**11-2. Вычисление функций Бесселя** ..... 224
**11-3. Програма ускорения сходимости** ..... 226

**Глава 12 Разработка программ** ..... 229

**12-1. Структурирование программы** ..... 229
12-1-1. Деление программы на подпрограммы ..... 229
12-1-2. Две иерархические системы ..... 230
**12-2. Проектирование программы** ..... 231
12-2-1. Разделение на подпрограммы ..... 231
12-2-2. Прерывания ..... 233
12-2-3. Написание программы ..... 233
**12-3. Пример программирования** ..... 235
12-3-1. Входы, выходы и обработка ..... 235
12-3-2. Разделение на подпрограммы второго уровня .. 236
12-3-3. Подпрограммы третьего уровня ..... 237
12-3-4. Подпрограмма последнего уровня ..... 240
12-3-5. Номера строк и инициализация ..... 240

**Глава 13 Структура памяти** ..... 249

**13-1. Постоянное запоминающее устройство (ПЗУ) и оперативное запоминающее устройство (ОЗУ)** ..... 249
13-1-1. ПЗУ ..... 250
13-1-2. ОЗУ ..... 252
**13-2. Управление ОЗУ** ..... 255
13-2-1. Оператор CLEAR ..... 255
13-2-2. Функция FRE(0) ..... 256
13-2-3. Функция FRE("") ..... 257
13-2-4. Оператор MAXFILES = ..... 258
13-2-5. Функция VARPTR ..... 258
13-2-6. Операторы BSAVE и BLOAD ..... 259
**13-3. PIT (Program Instruction Table - Таблица программных команд)** ..... 260
13-3-1. Адрес PIT ..... 261
13-3-2. Структура PIT ..... 261
**13-4. VT (Variables Table - Таблица переменных)** ... 264
**13-5. Стек** ..... 265
**13-6. Строковое пространство** ..... 266
**13-7. Рабочая область** ..... 268

**Глава 14 Структура видеопамяти** ..... 271

**14-1. Режим SCREEN 0** ..... 272
14-1-1. Структура видеопамяти в SCREEN 0 ..... 272
14-1-2. Соотношение отображения и таблиц в SCREEN 0. ..... 273
**14-2. Режим SCREEN 1** ..... 275
14-2-1. Структура видеопамяти в SCREEN 1 ..... 275
14-2-2. Соотношение отображения и таблиц в SCREEN 1 ..... 276
**14-3. Режим SCREEN 2** ..... 277
14-3-1. Структура видеопамяти в SCREEN 2 ..... 277
14-3-2. Соотношение отображения и таблиц в SCREEN 2 ..... 279
**14-4. Режим SCREEN 3** ..... 283
14-4-1. Структура видеопамяти в SCREEN 3 ..... 283
14-4-2. Соотношение отображения и таблиц в SCREEN 3 ..... 285
**14-5. Режим SCREEN 4 (для компьютеров MSX-2)** .. 286

14-6. Режим SCREEN 5 (для компьютеров MSX-2) .. 287
14-6-1. Структура видеопамяти в SCREEN 5 ..... 287
14-6-2. Связь между PNT и отображением в режиме SCREEN5 ..... 288
14-6-2. Связь между PNT и отображением в режиме SCREEN5 ..... 288
14-7. Режим SCREEN 6 (для компьютеров MSX-2) .. 289
14-7-1. Структура видеопамяти в SCREEN 6 ..... 289
14-7-2. Связь между PNT и отображением в режиме SCREEN 6 ..... 290
14-8. Режим SCREEN 7 (для компьютеров MSX-2 с объемом видеопамяти 128 Кбайт) ..... 290
14-8-1. Структура видеопамяти в SCREEN 7 ..... 290
14-8-2. Связь между PNT и отображением в режиме SCREEN 7 ..... 290
14-9. Режим SCREEN 8 (для компьютеров MSX-2 с объемом видеопамяти 128 Кбайт) ..... 291
14-9-1. Структура видеопамяти в SCREEN 8 ..... 291
14-9-2. Связь между PNT и отображением в SCREEN 8 ..... 292
14-10.Таблица палитр (для компьютеров MSX-2) .... 293
14-11. Спрайты ..... 295
14-11-1. SGT ..... 295
14-11-2. SAT ..... 296
14-11-3. SCT (для компьютеров MSX-2) ..... 298

Глава 15 Управление видеопамятью ..... 299

15-1. Псевдопеременная BASE ..... 299
15-1-1. Чтение адресов таблиц ..... 299
15-1-2. Установление расположения таблиц ..... 300
15-2. Псевдопеременная VDP ..... 302

Глава 16 Введение в программирование на машинном языке ..... 303

16-1. Процессор Z-80 ..... 303
16-2. Оператор DEFUSR и функция USR ..... 304
16-3. Подпрограммы BIOS ..... 305
16-3-1. Подпрограммы BIOS без параметров ..... 305
16-3-2. Подпрограммы BIOS, требующие параметров .. 307
16-3-3. Подпрограммы BIOS, возвращающие параметры 310
16-4. Использование ловушек. ..... 312
16-5. Написание программы на машинном языке (без программы-монитора) ..... 314
16-6. Короткая программа "Монитор звуковых эффектов" ..... 319



Глава 17 Программирование портов ..... 329
17-1. Программируемый параллельный интерфейс (PPI) ..... 330
17-2. Другие порты ..... 332

ПРИЛОЖЕНИЕ А УПРАВЛЯЮЩИЕ КЛАВИШИ И КОДЫ ..... 333
ПРИЛОЖЕНИЕ Б СООБЩЕНИЯ ОБ ОШИБКАХ ... 337
ПРИЛОЖЕНИЕ В ВЫРАЖЕНИЯ ДЛЯ НЕКОТОРЫХ МАТЕМАТИЧЕСКИХ ФУНКЦИЙ ..... 345
ПРИЛОЖЕНИЕ Г СПИСОК КЛЮЧЕВЫХ СЛОВ С ВНУТРЕННИМИ КОДАМИ ... 347
ПРИЛОЖЕНИЕ Д СИСТЕМЫ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ ..... 351
Д-1. Десятичная система ..... 351
Д-2. Двоичная, восьмеричная и шестнадцатеричная системы ..... 352
Д-3. Преобразования ..... 353
Д-4. Представление в памяти целых чисел ..... 355
Д-5. Хранение целых чисел ..... 359
Д-6. Двоично-десятичная система счисления ..... 362
Д-7. Хранение чисел одинарной и двойной точности ..... 363
ПРИЛОЖЕНИЕ Е ПОДПРОГРАММЫ BIOS ..... 367
Е-1. Интерпретатор, управление слотами, аппаратные прерывания ..... 368
Е-2. Инициализация ввода-вывода ..... 371
Е-3. Доступ к видеопроцессору (VDP) (режим TI9918) ..... 372
Е-4. Доступ к PSG (звукогенератору) ..... 378
Е-5. Доступ к консоли (клавиатуре и монитору) ... 379
Е-6. Доступ к кассетной ленте ..... 384
Е-7. Обработка очередей (для оператора PLAY) .. 385
Е-8. Подпрограммы, используемые модулями GENGRP и ADVGRP ..... 386
Е-9. Дополнительные подпрограммы ..... 389
Е-10. РАСШИРЕННОЕ ПЗУ ..... 394
ПРИЛОЖЕНИЕ Ж РАБОЧАЯ ОБЛАСТЬ ..... 415
Ж-1. Область переменных MSX Бейсика ..... 415

Ж-2. Ловушки ..... 419
ПРИЛОЖЕНИЕ З ОБРАЗЫ И КОДЫ СИМВОЛОВ ... 421
ПРИЛОЖЕНИЕ И КАРТА ПАМЯТИ ..... 423
ПРИЛОЖЕНИЕ К ИСПОЛЬЗОВАНИЕ ДИСКА "ОЗУ"(ВИРТУАЛЬНОГО ДИСКА) (ТОЛЬКО ДЛЯ MSX-2) ..... 425
К-1. Оператор CALL MEMINI(размер) ..... 426
К-2. Спецификации файлов ..... 426
К-3. Операторы расширения, используемые с Диском ОЗУ ..... 427
К-4. Другие операторы и функции ..... 427
ПРИЛОЖЕНИЕ Л ПРИМЕЧАНИЯ ДЛЯ ПОЛЬЗОВАТЕЛЯ СХЕМЫ УПРАВЛЕНИЯ ПАМЯТЬЮ (М А П П Е Р А) ..... 429
Л-1. Описание подпрограмм поддержки маппера .. 430
Л-2. Спецификации подпрограмм поддержки маппера ..... 435
Л-3. Примеры использования маппера ..... 440
Л-4. Спецификация маппера MSX-2 ..... 443
ПРИЛОЖЕНИЕ М СРЕДСТВА КОММУНИКАЦИИ .. 447
М-1. Введение в методы передачи данных ..... 447
М-2. Интерфейс RS-232C ..... 452
М-3. Терминальный режим ..... 458
М-4. Заключение ..... 460
М-5. Режим передачи данных ..... 462
М-6. Расширение Бейсика для связи по каналу RS-232C ..... 466

# Начинаем осваивать MSX Бейсик

Глава 1

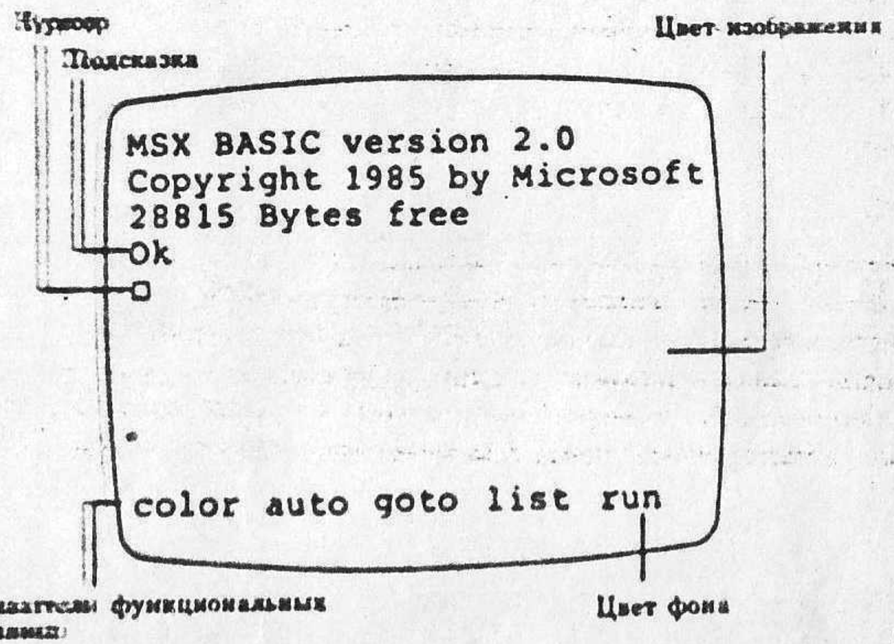
Эта глава рассчитана на пользователей, которые не имеют опыта программирования. Поэтому рекомендуется разобрать и попробовать на компьютере все включенные в нее примеры. Здесь используются лишь самые элементарные концепции программирования; большую роль играет интуиция читателя. Более детальное описание конструкций языка содержится в последующих главах.

## 1.1. Клавиатура и экран

Перед тем, как включить компьютер, ознакомьтесь с руководством для пользователя компьютера серии "Ямаха MSX-2".

Компьютер готов к работе, когда экран выглядит так, как это показано на приведенном ниже рисунке.

Экран после включения питания



### 1.1.1. Некоторые определения

**Сообщение об авторских правах** — В первой строке на экране указан номер версии MSX Бейсика, которая установлена на Вашем компьютере: версия 1.0 означает, что компьютер относится к серии MSX-1 (или просто MSX); версия 2.0 означает, что компьютер относится к серии MSX-2. В третьей строке сообщается, что в распоряжении Вашей программы имеется 28815 байтов памяти (байт - это единица памяти, хранящая один символ). Общее распределение памяти в компьютере будет рассмотрено далее.

**Подсказка Ok** — Эта подсказка означает, что компьютер находится в состоянии ожидания ввода команд или текста программы.

**Курсор** — Маленький белый прямоугольник, расположенный под подсказкой Ok, называется курсором. Он показывает место, где появится следующий вводимый с клавиатуры символ.

**Указатели функциональных клавиш** — Пять слов, появляющихся в нижней части экрана, указывают назначение пяти клавиш - F1-F5. Эти клавиши называются функциональными. Далее мы увидим, что назначение функциональных клавиш можно изменять с помощью команд или операторов программы; поэтому их называют еще программируемыми клавишами.

**Цвет изображения** — Цвет текста, появляющегося на экране, называется цветом изображения.

**Цвет фона** — Цвет пустой части экрана называется цветом фона.





### 1.1.2. Изучаем клавиатуру

Клавиатура компьютера используется так же, как клавиатура пишущей машинки. Тем не менее, существует несколько важных отличий.

- Большинство клавиш являются клавишами с автоповтором: достаточно нажать клавишу A и удержать ее некоторое время, чтобы убедиться в этом.

- Курсор автоматически возвращается к началу следующей строки, когда он достигает правой границы экрана; в этом легко убедиться, если удерживать клавишу A нажатой достаточно долго.

- Нажатие клавиши RETURN (большая клавиша с изображением изогнутой стрелки) не только переводит курсор на следующую строку, но и служит для компьютера признаком того, что введенная строка должна быть проанализирована. Если компьютер "не понимает" введенный текст, на экране появляется какое-либо сообщение об ошибке (например, Syntax error - ошибка в синтаксисе), а под ним загорается курсор.

- В отличие от пишущей машинки компьютер различает символы 0 (ноль) и заглавную букву O, а также 1 (один) и прописную букву l. Обращайте на это внимание с тем, чтобы не ввести l и O как числа.

- Клавиатура компьютера имеет много специальных клавиш, которых не имеет пишущая машинка. Они позволяют корректировать введенный текст и т.п. Последующие упражнения позволят Вам овладеть применением этих клавиш-команд.

Существует некоторое различие в расположении клавиш на клавиатуре, принятом в разных странах (французская клавиатура и т.п.). Международный вариант клавиатуры показан ниже. Если Вы работаете с другим типом клавиатуры, изучите описание Вашего компьютера.

Международный вариант клавиатуры


В дальнейшем мы придерживаемся следующих договоренностей:

- Клавиши, имеющие только одно значение, будут указываться этим значением, взятым в маленький квадрат (например, Q = [Q]).
- Клавиши, имеющие два значения, могут быть указаны любым из них (например, [F1] будет использована для указания на первую функциональную клавишу; [F6] будет указывать на эту же клавишу, когда она нажимается одновременно с клавишей [SHIFT]).
- Клавиши, не имеющие указанных значений, будут указываться так, как они показаны на рисунке (см.выше).
- Знак +, расположенный между двумя обозначениями клавиш, указывает, что вторая клавиша должна быть нажата при удержанной первой клавише (например, [F6] = [SHIFT] + [F1]). Знак умножения ( • ) означает, что указанную перед ним клавишу нужно нажать заданное число раз.





Последующие маленькие примеры помогут Вам освоиться с клавиатурой компьютера "Ямаха MSX-2".

Примеры:

(1) Очистка экрана
[SHIFT] + [HOME]

(2) Напечатать строку aaaAAAA
[A] x 3, [SHIFT] + [A] x 4

(3) Стирание последнего символа A в строке
[BS]

(4) Перемещение курсора на первый символ A в строке
[←] x 3

(5) Вставка BB между a и A в строке (перейти в режим вставки)
[INS] x 1, [B] x 2

(6) Возвратиться в нормальный режим (отмена режима вставки)
[INS] x 1

(7) Возвратить курсор к первому символу a в строке
[←] x 5

(8) Удаление aaa без перемещения курсора в строке
[DEL] x 3

(9) Перемещение курсора на первый символ A в строке
[→] x 2

(10) Удаление BB со сдвигом оставшейся части строки влево
[BS] x 2

(11) Удаление второго символа A с оставлением на этом месте пробела
[→] x 1, пробел

(12) Возвращение курсора к началу строки и перемещение "слова" на три позиции вправо
[←] x 2, [INS], пробел x 3

Теперь Вы освоили работу основных клавиш-команд клавиатуры.
Давайте сделаем еще один шаг вперед:

* Клавиша [CAPS] позволяет вам перевести клавиатуру в режим ввода заглавных букв. Все буквы, введенные при подсвеченной клавише [CAPS], будут заглавными. Чтобы выключить подсветку, нажмите клавишу [CAPS] еще раз. В отличие от клавиши [SHIFT] клавиша [CAPS] действует только на алфавитные клавиши.

* Когда Вы нажимаете [SHIFT], указатели функциональных клавиш на экране меняют свое значение, указывая на клавиши [F6] ~ [F10].

* Комбинация [GRAPH] + одна из символьных клавиш позволяет ввести графический знак (символ).

* Комбинация [РУС] + символьная клавиша позволяет ввести текст, состоящий из букв кириллицы. Эта клавиша работает так же, как клавиша [CAPS] и может быть подсвечена одновременно с [CAPS].

* Существуют многочисленные "скрытые" команды. Чтобы работать с ними, необходимо нажать одну из символьных клавиш, удерживая при этом клавишу [CTRL]. Например, команда [CTRL] + [G] выдаст короткий звуковой сигнал (как оператор Бейсика **BEEP**, см. далее). Команда [CTRL] + [U] стирает целую строку. Полный перечень таких "скрытых" команд - управляющих кодов - Вы найдете в Приложении А. ОБРАТИТЕ ВНИМАНИЕ: в некоторых программах для компьютера "Ямаха MSX-2" эти управляющие коды могут иметь совершенно другие значения, специфичные для данной программы.





### 1-1-3. Логическая и физическая строки

Существуют два очень важных понятия, которые нуждаются в объяснении:

* **Физическая строка**

Физическая строка - это строка, которая появляется на экране. Обычно строки, которые Вы вводите, имеют длину до 80 символов. В дальнейшем будет показано, как можно изменить количество символов в физической строке.

* **Логическая строка**

Логическая строка содержит все физические строки, которые Вы вводите как один б л о к. Она может содержать до 255 символов. Логическая строка соответствует осмысленной порции текста - т.е. тому, что компьютер попытается "понять", когда Вы нажмете [RETURN].

Пример:
(1) Нажмите [SHIFT] + [HOME], чтобы очистить экран
(2) Введите все, что угодно, чтобы заполнить 5 физических строк, затем нажмите [RETURN].
(3) Не обращая внимания на сообщение об ошибке, нажмите клавишу [HOME] для того, чтобы возвратить курсор в начало логической строки.
(4) Нажатие [CTRL] + [U] удалит логическую строку (см. Приложение А).

Вы видите, что введенный текст полностью исчез. Если Вы ввели более, чем 255 символов, удаляются только первые 255 символов.

Примеры:
(1) Введите любой текст, занимающий половину физической строки, а затем перейдите на следующую физическую строку, нажав клавишу пробела.

(2) Введите несколько символов на этой строке, затем нажмите RETURN.
(3) Обратите внимание, что команда CTRL + U выполнится так, как это описано выше - будут удалены обе физические строки, поскольку они входят в состав одной логической строки.

Если Вы использовали → для перехода к следующей строке вместо клавиши пробела, компьютер рассматривает эти строки как последовательность двух отдельных логических строк. В этом случае операция CTRL + U удалит только одну строку, оставив неизменной другую.

## 1-2. Язык программирования MSX Бейсик

Мы убедились, что компьютер посылает сообщения об ошибках каждый раз, когда он "не понимает", чего Вы хотите. Следовательно, первое, что мы должны сделать сразу же после включения, - это выучить язык компьютера; язык, слова и фразы которого "понимает" компьютер "Ямаха MSX-2". Этот язык называется языком программирования MSX Бейсик.
Когда-то компьютеры могли понимать только машинный язык. Этот язык понятен электронным схемам компьютера, но он не имеет ничего общего с повседневным языком. MSX Бейсик, наоборот, намного ближе к разговорному языку. Слова, используемые в этом языке, пришли из английского языка. Словарь Бейсика настолько ограничен, что выучить сами эти слова не составит ни малейшего труда. Синтаксис MSX Бейсика (грамматические правила) несколько более специфичен, чем синтаксис разговорного языка. Действительно, в разговорном языке небольшие синтаксические ошибки не помешают Вам передать смысл, тогда как малейшие синтаксические ошибки в тексте на Бейсике не дадут компьютеру возможности понять, что нужно выполнить.





### 1-2-1. Арифметические операции

Четыре арифметические операции помогут Вам начать изучение MSX Бейсика.

Введите 1+1 = и нажмите RETURN.
Вы не получите никакого ответа, потому что компьютер пока не понимает, чего Вы хотите. На самом деле Вы хотели бы увидеть результат выполнения этой операции на экране. Чтобы у в и д е т ь на экране ответ, надо приказать компьютеру вывести число на экран; эту операцию выполнит команда PRINT (английское слово НАПЕЧАТАТЬ; оно сохранилось в Бейсике с тех времен, когда вместо экрана и клавиатуры использовалось печатающее устройство - телетайп).
Таким образом, нужно ввести PRINT1+1 и нажмать RETURN.

★ Вместо PRINT вы можете ввести для краткости вопросительный знак.
★ В дальнейшем мы будем просто говорить "ввести ?1+1", опуская указание, что затем должна быть нажата клавиша RETURN.

Для четырех арифметических операций используются следующие символы:

| | |
|---|---|
| + | сложение |
| – | вычитание |
| * | умножение |
| / | деление |

Теперь Вы можете использовать свой компьютер в качестве калькулятора. Говоря о четырех операциях, следует сказать, что MSX Бейсик использует обычный порядок вычислений: сначала выполняются умножение и деление, а затем - сложение и вычитание.

Для проверки этого введите:

```
?2•4+8/2
```

Порядок вычислений можно изменить, используя скобки; при этом действия, взятые в скобки, будут выполнены первыми, в соответствии с правилами, описанными выше.

Проверьте это:

```
?2•(4+8/2)
```

Двойное деление может быть записано двумя способами:

Первый способ короче и лучше.

```
?16/4/2 или ?16/(4•2)
```

К четырем арифметическим операциям может быть добавлена операция возведения в степень: куб от 5 (третья степень) записывается как

```
?5^3
```

(знак ^ вводится при отключенной клавише РУС с нажатием SHIFT).
Возведение в степень имеет приоритет перед другими арифметическими операциями.

Проверьте следующий пример:

```
?4^2•4/3+8
```

Возведение в степень выполнится первым. Результат - 16 - умножается на 4, затем делится на 3, после чего к результату прибавляется 8.





## 1-2-2. Числовые константы и переменные

Числа, которые мы использовали в предыдущих примерах, называются числовыми константами. Так же, как в алгебре, работать с переменными приходится чаще, чем с константами.

Введите следующий пример:

```
A=5
B=3
?A•B
```

Есть существенная разница между алгебраическими переменными и числовыми переменными в языках программирования. В алгебре можно производить действия над переменными (упрощение выражений и т. д.), не зная их значений.

Иными словами, можно написать:

$$A•B+C•A=A•(B+C)$$

Это значит, что вычисление левой и правой частей выражения дадут один и тот же результат, какими бы ни были значения переменных A, B и C. Компьютер, работающий с языком Бейсик, может вычислять значения выражений только тогда, когда у входящих в выражение переменных известны их значения. На самом деле, числовая переменная в Бейсике является только именем, указывающим на ячейку компьютерной памяти. Каждый раз, когда в выражении присутствует числовая переменная, компьютер находит соответствующий "ящик"-ячейку и выбирает из нее текущее значение.

Выражение типа:

```
A=A+1
```

подразумевает в алгебре, что A равно нулю.

Для компьютера это выражение значит:

(1) взять значение из ячейки с именем A
(2) добавить 1 к этому значению
(3) поместить новое значение снова в ячейку с именем A.

Имена переменных подчинены следующим ограничениям:

* Из всего имени для обозначения переменной в памяти компьютера используются только д в а первых символа.

Проверьте это вводом:

```
ALPHA=25
```

Теперь запросите значения переменных **AL**, **ALPH** и **ALXYZ**:

```
?AL
?ALPH
?ALXYZ
```

Каждый раз вы получите 25.

* Имя переменной должно начинаться с буквы л а т и н и ц ы. В остальных позициях можно использовать и цифры, но специальные символы использовать не рекомендуется за исключением предписанных случаев (см. далее).
* Имя переменной не может быть ключевым ("зарезервированным") словом и не может включать в себя такие слова. Ключевыми словами считаются слова, пришедшие в Бейсик из английского языка и принадлежащие уже компьютерному словарю. Мы использовали такое слово (**PRINT**). Полный перечень ключевых слов Бейсика может быть найден в Приложении Г.

Пример:
**OR** является ключевым словом. Имена переменных **ORE**, **ORDER**, **BORE** и тому подобные не могут быть использованы.





* Переменная, которая не получала значений (иными словами, которая не была определена), равна нулю. Следовательно, после включения компьютера следующая команда

```
?AB
```

выводит на экран нулевое значение.

## 1-2-3. Символьные строки

Ваш компьютер может работать не только с числами, но и с символьными строками. Таким образом, теперь мы можем говорить об алфавитно-цифровых константах, переменных и операциях. Алфавитно-цифровая константа вводится между кавычками; имя алфавитно-цифровой переменной записывается с добавлением знака $ в конце имени. Эти имена подчиняются тем же самым ограничениям, что и имена числовых переменных.

Например:

```
A$="BASIC"
B$="MSX"
```

Теперь попросим выдать значения этих переменных:

```
?A$
?B$
```

Символы, которые Вы напечатали между кавычками, честно воспроизведутся (это относится и к пробелу, который Вы ввели после букв MSX).
Знак + служит для "склейки" (или, как говорят, конкатенации) символьных переменных или констант.

Например:

```
?B$+A$
?B$+A$+" "+"Computer"
```

Обратите внимание на ввод пробела.

Конечно же, смешение переменных двух этих типов недопустимо, как лишенное смысла:

```
?A$+A
```

- что бы это значило? Как здесь понимать плюс?..

В заключение попробуйте следующий пример, содержащий как символьную строку-константу, так и числовые значения:

```
A=10
B=21.5
PRINT"TOTAL=";A+B
```

Обратите внимание на точку с запятой, разделяющую "символьную" и "числовую" части команды. Этот знак препинания нужен для того, чтобы на экране слово TOTAL (попробуйте заменить его русским ИТОГ) и число-результат сложения стояли рядом в одной строке.

## 1-3. Что такое программа?

До сих пор мы использовали компьютер в ПРЯМОМ режиме. Прямой режим используется для немедленного выполнения простых команд. В этом режиме можно также вводить простейшие программы разового действия.





Выполнение программ обычно происходит в другом режиме. Программа это последовательность команд, которые выбираются компьютером в определенном порядке. В отличие от команд прямого режима, которые будут потеряны, если экран очистить (например, клавишей-командой CLS), программа хранится в памяти компьютера. Она может быть исполнена столько раз, сколько Вам это понадобится, даже если она исчезла с экрана. Она может быть вызвана на экран для внесения изменений в текст (или, как принято говорить, для редактирования). Программа может быть выведена на принтер (часто употребляется термин "распечатана") и сохранена на дискете или кассете магнитной ленты.

### 1-3-1. Составление и использование программ

Следующий короткий пример поможет Вам понять, как это все работает.

(1) Введите команду NEW, а затем следующую короткую программу:

```
10 A=123
20 PRINT"РЕЗУЛЬТАТ=";A
```

Обратите внимание, что нажатие клавиши RETURN в конце строки не влечет за собой выполнения каких-либо действий: только курсор перемещается в начало следующей строки.

(2) Нажмите клавишу F5 (или введите команду RUN с клавиатуры). Все команды, из которых состоит программа, будут выполнены. Иначе говоря, команда RUN запускает программу.

(3) Нажмите SHIFT + HOME для того, чтобы очистить экран, а затем снова нажмите клавишу F5. Программа выполнится еще один раз, что доказывает ее сохранность в памяти.

(4) Чтобы просмотреть текст программы, нажмите клавишу F4 (или введите команду LIST с клавиатуры).

(5) Давайте немного изменим текст программы. Используя клавиши управления курсором, расположите курсор на цифре 1 (в числе 123) в первой строке и введите 5. Теперь программа выглядит следующим образом:

```
10  A=523
20  PRINT"РЕЗУЛЬТАТ=";A
```

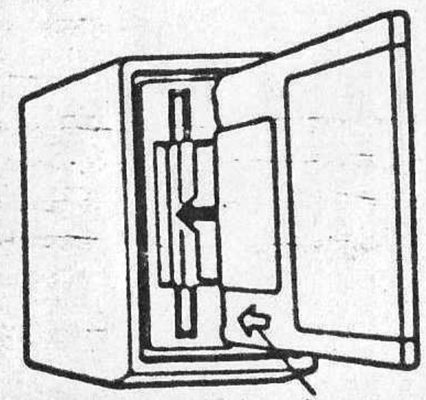
(6) Для того, чтобы ввести измененную строку, нажмите клавишу RETURN (при этом не обязательно, чтобы курсор находился в конце строки).

(7) Переместите курсор вниз, чтобы выйти из той части экрана, на которой виден текст программы, и нажмите F5. Будет выполнена новая (измененная) программа.

(8) Если у Вас есть принтер, и он подключен к компьютеру, вставьте лист бумаги и убедитесь, что горит индикатор ON LINE. Для того, чтобы напечатать текст программы, введите команду LLIST (не LIST!).

(9) Для того, чтобы сохранить программу, Вы можете использовать рабочую дискету, заранее отформатированную (см. описание компьютера). Вставьте дискету в щель дисковода, как показано ниже.

Как вставить дискету в механизм дисковода:



Стрелка





(10) Введите SAVE"TEST".
Индикатор дисковода включится; будет слышен легкий жужжащий звук. Через некоторое время курсор появится вновь - это означает, что программа записана на дискету под именем TEST.

(11) Введите NEW. Эта команда, уже использованная ранее в примере (см. пункт (1)), стирает текст программы в оперативной памяти. Текст на дискете останется цел, теперь, если Вы попробуете выполнить команды LIST или RUN, ничего не произойдет.

(12) Чтобы вызвать программу, которая была сохранена на дискете, введите:

LOAD"TEST"

Как только курсор вновь появится на экране, программа снова будет находиться в оперативной памяти компьютера. Оперативная память подобна классной доске, а дискета выполняет роль записной тетради.

### 1-3-2. Редактирование текста программы

Редактирование программы - это прежде всего ее написание. Программа состоит из определенного количества строк, начинающихся с номера. Номер строки говорит компьютеру, что следующая команда является частью программы и не должна выполняться в прямом режиме. Когда Вы вводите логическую строку, компьютер прежде всего смотрит на начало строки. Если строка начинается с номера, он будет рассматриваться как порядковый номер строки в программе. Если же строка не начинается с номера, то компьютер обработает такую строку как команду прямого режима. Когда это не удается, то на экране появляется сообщение об ошибке.

При вводе строки, начинающейся с номера, могут произойти следующие два события:

(1) Номер строки совпадает с номером строки, уже хранящейся в памяти компьютера. В этом случае существующая строка заменяется на вновь введенную.

(2) Программа не содержит строки с таким номером. Введенная строка будет добавлена к тексту программы.

Перед вводом новой программы мы рекомендуем Вам пользоваться командой NEW. Эта команда стирает содержимое той части памяти, где будет записана новая программа. После ввода программы нажмите F5 для проверки правильности ее выполнения. Новые программы весьма редко начинают работать с первого раза.

Существует два типа программных ошибок:

(1) Ошибки ввода.

Во время выполнения программы компьютер выявляет команды, которые он "не понимает". При этом прерывается выполнение программы, и компьютер выводит на экран сообщение об ошибке, а также, если это необходимо, номер строки, в которой эта ошибка была обнаружена.

Используя команду LIST, выведите программу на экран, после чего замените строку 10 на следующую:

```
10  A=5A3
```

Нажмите клавишу F5. Сообщение

**Syntax error in 10**





указывает тип ошибки и номер программной строки, где она обнаружена. Вы должны снова вывести программу на экран и скорректировать эту строку.

(2) Смысловые (алгоритмические) ошибки.

Исполнение программы не будет прервано, однако она не выдаст ожидаемый результат.
Этот тип ошибок наиболее трудно исправить, особенно в очень больших программах, потому что они часто вызваны неверной постановкой задачи. К этой проблеме мы вернемся позже.

Для просмотра программы Вы всегда должны использовать команду LIST. Вы можете использовать CTRL + STOP для прекращения вывода программы на экран. Нажатие клавиши STOP только приостанавливает вывод. Нажмите ее еще раз для продолжения просмотра.

| | |
|---|---|
| LIST | просмотр программы целиком. |
| LIST 10 | просмотр только строки 10. |
| LIST 10-30 | просмотр программы со строки 10 до строки 30. |
| STOP | приостановка вывода программы на экран, повторное нажатие возобновляет вывод. |
| CTRL + STOP | прекращение вывода программы на экран. |

Итак, Вы можете вводить готовые программы. Некоторые журналы печатают короткие игровые программы, которые можно опробовать. Таким образом Вы сможете потренироваться во вводе программы, исполнении ее, просмотре, а также вывести текст программы на принтер.

### 1-3-3. Операторы программы

Небольшая программа, приведенная выше, могла быть написана в виде одной программной строки:

Оператором Бейсика будем называть ту часть текста, которая

```
10  A=123:PRINT"РЕЗУЛЬТАТ=";A
```

обрабатывается компьютером как единое целое. Чаще всего оператор выполняется, являясь при этом командой. Строка 10 примера содержит два оператора. Когда строка содержит несколько операторов, должны соблюдаться следующие правила:

(1) Операторы должны быть разделены двоеточием.

(2) Программная строка не может содержать более 255 символов (предел длины логической строки).

### 1-4. Порядок выполнения операторов в программе

Программа выполняет операторы в порядке возрастания номеров строк. Если строка содержит несколько операторов, они выполняются в том порядке, в каком они записаны в строке. Порядок выполнения операторов в программе можно изменить различными способами. Рассмотрим два из них.

### 1-4-1. Программные циклы

Программные циклы широко используются в большинстве программ. Они позволяют повторять выполнение последовательности операторов.

```
10  FOR I=0 TO 9
20  PRINT I^2
30  NEXT I
```





Строка 10 определяет значения счетчика цикла I. Строка 20 содержит оператор тела цикла. Строка 30 содержит оператор конца цикла. Каждый раз, когда компьютер достигает этой строки, к счетчику цикла будет добавлена единица, в данном случае - шаг цикла. Компьютер сравнивает значение счетчика с предельным значением (9), указанным в строке 10.
Если значение счетчика достигает этого предела, компьютер переходит к следующей строке (в нашем случае строк больше нет, и компьютер перейдет в прямой режим). Если значение счетчика меньше или равно предельному значению, выполняется тело цикла между FOR... и NEXT...

Разрешены также вложенные циклы:

```
10  FOR I=0 TO 5
20  PRINT I^2
30  FOR J=0 TO 1
40  PRINT
50  NEXT J,I
```

Цикл по J включен в цикл по I. Обратите внимание, что оператор NEXT включает в себя оба счетчика; при этом счетчик внутреннего цикла J находится на первом месте.

Иногда программные циклы используются для организации простых задержек:

```
10  FOR I=0 TO 15
20  COLOR I
30  FOR J=0 TO 1000
40  NEXT J,I
```

Оператор COLOR переопределяет здесь цвет текста. Цикл по переменной J позволяет Вам установить желаемую скорость смены цветов.

### 1-4-2. Подпрограммы

Подпрограммы - это средство организации ясной структуры программ. На практике программы обычно стремятся разделить на несколько небольших программ (подпрограмм), которые можно отлаживать по отдельности. Таким образом можно добиться, чтобы основная программа выглядела как цепочка вызовов подпрограмм - подобно тому, как устройство состоит из деталей. Кроме того, применение подпрограмм, к которым происходит несколько обращений из различных мест основной программы, позволяет существенно сократить общую длину текста программы.

TRIANGLE

```
10  C1=23.4
20  C2=13.89
30  GOSUB 100
40  GOSUB 200
50  END
100 PRINT"ГИПОТЕНУЗА=";SQR(C1^2+C2^2)
110 RETURN
200 PRINT"ПЛОЩАДЬ=";C1*C2/2
210 RETURN
```

Команда GOSUB 100 передает управление строке 100. Когда компьютер выполняет команду RETURN, он должен автоматически возвратиться к той команде, которая стоит в программе сразу за "вызывающей" командой (в данном случае GOSUB 100). Оператор END в строке 50 останавливает выполнение программы. Без этого оператора команды подпрограммы (строки 100 - 200) выполнились бы во второй раз: это повлекло бы за собой появление сообщения об ошибке.

Обратите внимание, что ключевое слово SQR сопровождается выражением, взятым в скобки: это ф у н к ц и я. SQR вычисляет квадратный корень числа,заданного аргументом функции. Более подробно функции будут рассмотрены далее.





Для удобства организации программ необходимо, чтобы любая подпрограмма могла в свою очередь вызывать другие подпрограммы. Можно также включить подпрограммы в тело цикла FOR или воспользоваться циклами FOR в подпрограмме.

Другие способы изменения порядка выполнения команд в программе будут рассмотрены далее.

### 1-5. Принятие решений в программе

Компьютер должен уметь принимать решения, основанные на значениях определенных переменных. Давайте рассмотрим несколько примеров, иллюстрирующих эту возможность и способы ее реализации.

### 1-5-1. Оператор IF... THEN...

Предположим, что мы хотим написать программу для вычисления корней квадратного уравнения ( A*X^2+B*X+C = 0 ). С учетом знака выражения B^2-4*A*C (дискриминанта) уравнение будет иметь либо два вещественных корня, либо два сопряженных комплексных корня, либо двойной корень. В алгоритме вычисления корней необходимо учесть все эти варианты.

QUADEQ1

```
10  A=10
20  B=3/4
30  C=13
40  R=B^2-4*A*C
50  IF R > 0 THEN GOSUB 100:GOTO 90
60  IF R < 0 THEN GOSUB 200:GOTO 90
70  'Двойной корень
80  GOSUB 300
90  END
```

```
100 RR=SQR(R)
110 PRINT"ДВА ВЕЩЕСТВЕННЫХ КОРНЯ:"
120 PRINT"X1=";(-B+RR)/2/A
130 PRINT"X2=";(-B-RR)/2/A
140 RETURN
200 RR=SQR(-R)
210 PRINT"ДВА СОПРЯЖЕННЫХ
    КОМПЛЕКСНЫХ КОРНЯ (a+/-bi):"
220 PRINT"a=";-B/2/A
230 PRINT"b=";RR/2/A
240 RETURN
300 PRINT"ДВОЙНОЙ КОРЕНЬ:"
310 PRINT"X=";-B/2/A
320 RETURN
```

Значения коэффициентов A, B и C записываются в первых трех строках (A должно быть отлично от 0), после чего программа может быть запущена на выполнение.

В строке 40 вычисляется дискриминант. Оператор в строке 50 передает управление подпрограмме 100, если дискриминант положительный. Если дискриминант не положительный, выполняется следующая строка. Следующая программная строка (60) передает управление подпрограмме 200, если дискриминант отрицательный; управление переходит к строке 80, если дискриминант не отрицательный. Если дискриминант не положительный и не отрицательный, т.е. равен нулю, управление передается на подпрограмму 300.

Оператор IF... THEN... определяет в программе условные переходы, т.е. управление передается только после проверки некоторого условия. Оператор, следующий за словом THEN, выполняется, если заданное условие верно; на этом месте допускается любой оператор.





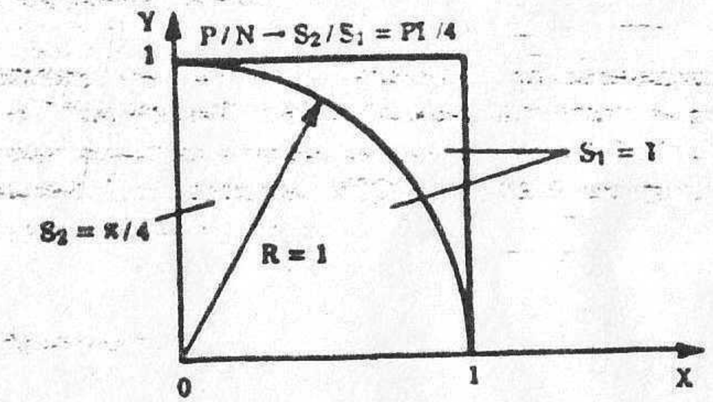
Рассмотрим следующий пример.

RANDPI

```
10 FOR N=1 TO 1000
20 X=RND(1):Y=RND(1)
30 IF X^2+Y^2<1 THEN P=P+1
40 PRINT"ПИ=(приближенно)";4*P/N
50 NEXT
```

Значения, выводимые этой программой, постепенно приближаются к значению числа "пи" (PI). Эта интересная программа использует метод Монте-Карло и работает следующим образом:

На рисунке показана четвертая часть круга радиусом 1, заключенного в квадрат со стороной, равной 1.




Площадь квадрата S1 равна 1. Площадь четверти круга S2 равна PI/4. Следовательно, PI=4*S2/S1. Программа выбирает S1 и S2 следующим образом:

(1) Возьмем случайную точку в пределах квадрата: ее координаты (X,Y) вычисляются двухкратным применением функции RND, которая возвращает случайное число в интервале от нуля до единицы (строка 20).

(2) Квадрат расстояния между началом оси координат и этой точкой вычисляется в строке 30. Если значение меньше 1 (то есть точка расположена внутри четверти круга), то к P прибавляется единица.
N указывает на общее количество попыток, а P - на общее количество точек, попавших внутрь четверти круга. Легко догадаться, что при достаточно большом количестве попыток отношение P/N приблизительно соответствует S2S1.

(3) Строка 40 выводит вычисленное значение PI для каждого прохода.

Эта программа - типичный пример численного метода, рассчитанного на применение компьютера. Программа не включает в себя какого-либо оригинального приема - весь расчет на скорость вычислений. Конечно, требуется достаточно большое количество циклов, чтобы получить приличный результат.

### 1-5-2. Оператор IF... THEN... ELSE

Существует возможность определить два условия перехода при проверке: одно при выполнении условия, а другое - для случая, когда условие не выполняется.

Примером может служить другая небольшая программа, вычисляющая PI.

SERIESPI

```
10  N = 1
20  T = 1/N/(N+2)
30  S = S+T
40  N = N+4
50  PRINT 8*S
60  IF T < .0000001 THEN END ELSE 20
```

Эта программа вычисляет PI следующим образом:





PI=8*(1/1/3+1/5/7+1/9/11+1/13/15...)

Счетчик N выбирает значения 1,5,9,13, и.т.д. используемые для последовательного генерирования компонентов суммы в строке 20. Компоненты суммируются в S в строке 30. Увеличение счетчика осуществляет строка 40.

Промежуточный результат печатается в строке 50. Значение компонента суммы проверяется в строке 60. Если значение меньше .0000001, программа заканчивает работу; в противном случае происходит переход к строке 20 (Вы можете выбрать значение, меньшее, чем .0000001, но это приведет к увеличению времени счета. Однако результат будет более точным).

Обратите внимание, как ELSE используется для возврата к строке 20. Между THEN и ELSE могут быть включены любые операторы или группы операторов, разделенных двоеточиями.

## 1-6. Взаимодействие с работающей программой

Очень часто бывает необходимо взаимодействовать с работающей программой без прерывания ее работы. Игровые программы, например, реагируют на определенные клавиши. Обычно это клавиши управления курсором или функциональные клавиши. Эффект от нажатия этих клавиш при работе подобных программ - совсем иной, чем при работе в прямом режиме. Принято говорить, что данные вводятся в программу в процессе ее работы. Данные можно разместить и непосредственно в тексте программы, но если программа имеет сложную логику, то легче заставить программу запрашивать данные по мере необходимости.

### 1-6-1. Оператор INPUT

Давайте возвратимся к программе решения квадратного уравнения QUADEQ1, описанной в разделе 1-5-1.

Замените строки с указанными номерами на следующие:

```
10  INPUT"A=";A
20  INPUT"B=";B
30  INPUT"C=";C

90  PRINT:GOTO 10
```

Компьютер выводит на экран небольшое сообщение, которое Вы поместили справа от **INPUT**, и ожидает от Вас ввода данных. Вы вводите необходимое значение, которое присваивается переменной, стоящей за сообщением в операторе **INPUT**. После выполнения ввода трех значений одного за другим компьютер производит вычисления, как и ранее.

В строке 90 оператор **GOTO** передает управление строке 10, и компьютер ожидает ввода новых данных. Для того, чтобы прервать исполнение этой программы, нажмите CTRL + STOP.

Таким способом можно вводить и алфавитно-цифровые данные:

```
10  CLS
20  INPUT"КАК ВАС ЗОВУТ";N$
30  CLS
40  PRINT"ПРИВЕТ, ";N$;"!"
```

То, что переменной **N$** будет присвоено символьное (алфавитно-цифровое) значение, указывается символом **$** в конце имени переменной.

Оператор **CLS** производит очистку экрана.

### 1-6-2. Функция INKEY$

Функция **INKEY$** позволяет программе реагировать на нажатие клавиши. Используя оператор **IF...THEN**, можно заставить программу реагировать на нажатие определенных клавиш.





```
10  K$=INKEY$
20  IF K$="" THEN 10
30  PRINT"КЛАВИША";K$;"НАЖАТА"
40  GOTO 10
```

"Имя" нажатой клавиши **K$** определяется в строке 10. Если переменная **K$** пустая, управление возвращается на строку 10. Если клавиша была нажата, изображенный на ней символ (значение) выводится при выполнении строки 30, и затем управление вновь передается строке 10.

**ОБРАТИТЕ ВНИМАНИЕ:** На клавиатуре имеются также клавиши типа ESC или INS, при нажатии которых эта программа ничего на экран не выведет. Приведенная ниже программа показывает, как обращаться с кодами, вводимыми при нажатии таких клавиш.

```
10  COLOR 15,C
20  K$=INKEY$
30  IF K$="" THEN 20
40  IF ASC(K$)=32 THEN C=C+1 ELSE 70
50  IF C=16 THEN C=0
60  GOTO 10
70  IF ASC(K$)=27 THEN COLOR 15,4:END ELSE 20
```

Оператор **COLOR** устанавливает цвет текста с номером 15 (белый), а цвет фона - с номером, равным значению C. Вначале значение **C** равно 0, что означает "прозрачный" цвет. Строка 20 вводит код нажатой клавиши, присваивая его значение переменной **K$**. Если оно пусто, происходит возврат к строке 20 (клавиша не была нажата). Строка 40 сравнивает код нажатой клавиши с числом 32 (код клавиши пробела). Если был нажат пробел, то значение **C** увеличивается; в противном случае производится следующее сравнение (строка 70). После

увеличения C управление передается строке 50. Номера цветов идут от 0 до 15. Если C равно 16, (после очередного увеличения), значение этой переменной снова устанавливается равным нулю. Строка 60 передает управление строке 10, где происходит переустановка цвета фона в соответствии с новым значением C. Если была нажата не клавиша пробела, строка 70 сравнивает код нажатой клавиши с 27 (код клавиши ESC). При нажатии клавиши ESC происходит установка цветов текста и фона в начальное значение, после чего программа прекращает работу (см. оператор END в строке 70). Если была нажата любая другая клавиша, управление автоматически возвратится к строке 20.

Примечание:

Если Вы прервали исполнение программы, используя CTRL + STOP, то Вы можете возвратить начальные значения цветов нажатием SHIFT + F1

Функция ASC возвращает код первого символа аргумента, которым должна быть символьная переменная. Кодовая таблица компьютера MSX-2, основанная на принятом в вычислительной технике коде ASCII и расширенная кодами букв кириллицы, приведена в Приложении A.

В главе 2 Вы найдете дополнительную информацию о различных способах ввода данных в работающую программу с клавиатуры, манипулятора МЫШЬ, джойстика, и т.п.





## 1-7. Запись данных в программе

К этому моменту мы знаем два способа ввода данных:

(1) Запись данных в программе с использованием операторов присваивания, таких, как, например, A=10. Этот способ пригоден при работе с ограниченным количеством данных, которые можно изменять, только изменяя сам текст программы.

(2) Ввод с использованием оператора INPUT. Этот метод позволяет оставлять текст программы неизменным, а изменять только значения требуемых переменных. Следует видеть, что ввод - процедура, требующая повышенного внимания от человека, работающего с программой, и сопряженная с потенциальными ошибками.

### 1-7-1. Операторы READ и DATA

При использовании операторов READ и DATA возможен только упорядоченный ввод данных, помещаемых в тексте программы заранее.

Давайте еще раз возвратимся к примеру из раздела 1-5-1 QUADEQ1 и предположим, что мы хотим вычислить корни большого количества квадратных уравнений (тяжелое испытание для учеников!).

Изменим следующие строки:

QAUADEQ3

```
10  READ A, B, C
15  IF A=0 AND B=0 AND C=0 THEN END
20  PRINT"A=";A
25  PRINT"B=";B
30  PRINT"C=";C:PRINT

90  IF INKEY$=""THEN 90 ELSE
    PRINT:PRINT:GOTO 10
400 DATA 10,.75,13
410 DATA 3,4,12
420 DATA 4.8,32,5
430 DATA 1,2,1
440 DATA 0,0,0
```

Оператор READ в строке 10 присваивает значения, найденные в строках DATA (400-440), переменным A, B и C. Операторы DATA становятся "банком данных", к которым компьютер обращается в том порядке, в каком они набраны в программе. Когда происходит запуск программы, оператор READ выполнит чтение трех значений данных (A, B и C). Программа с операторами READ всегда ищет данные в операторах DATA.

Если оператор READ выполняется повторно, то компьютер "вспомнит", что он уже обращался к 23 "банку данных", и автоматически переключится на ввод содержимого следующей строки DATA. В строке 15 данные проверяются. Если все три значения данных равны 0 (условный признак конца данных), то выполнение программы прекращается. Обратите внимание на использование AND (см. далее раздел 2-3-1).





Три оператора PRINT добавлены к программе для того, чтобы вывести на экран значения коэффициентов A, B и C перед вычислением корней. Вычисления производятся так же, как это было описано ранее; затем строка 90 передает управление строке 10.

Если Вы хотите использовать принтер, достаточно заменить все операторы PRINT на LPRINT.

Примечание:

Операторы DATA не обязательно должны быть написаны в конце программы или подряд. Мы могли бы записать все данные, хранящиеся в строках 400-440, в одной строке с использованием одного оператора DATA.
Мы могли бы также записать одиночные значения данных в разные операторы DATA, "рассыпав" их по всей программе. Для компьютера важен только порядок появления данных. С другой стороны, программисту должно быть удобно проверять эти данные.
Таким образом, наиболее практичный способ записи операторов DATA - подряд и в конце программы.

### 1-7-2. Оператор RESTORE

Оператор READ читает данные только один раз, в порядке их появления в программе. Если Вы попытаетесь использовать оператор READ после того, как все данные уже однажды прочитаны, на экране появится сообщение об ошибке "OUT of DATA".

В некоторых задачах требуется повторное чтение данных. Для организации повторного чтения используется оператор RESTORE, который позволяет как бы "передавать управление" на требуемую строку, содержащую оператор DATA.

Рассмотрим следующую программу:

```
10 INPUT"НОМЕР МЕСЯЦА (1-12)";N
20 IF N < 1 OR N > 12 THEN 10
30 RESTORE 100
40 FOR I=1 TO N
50 READ M$
60 NEXT
70 PRINT M$
80 GOTO 10
100 DATA ЯНВАРЬ,ФЕВРАЛЬ,МАРТ,АПРЕЛЬ,
    МАЙ,ИЮНЬ, ИЮЛЬ, АВГУСТ, СЕНТЯБРЬ,
    ОКТЯБРЬ, НОЯБРЬ, ДЕКАБРЬ
```

В строке 10 программа запрашивает номер месяца (от 1 до 12). Далее, программа проверяет введенное значение и отказывается от него, если оно не соответствует номеру одного из месяцев. Обратите внимание на использование **OR** (см. далее раздел 2-3-1).

Оператор **RESTORE** 100 в строке 30 указывает компьютеру, что следующая порция данных для чтения расположена в строке 100 (или начинается со строки 100).

Небольшой программный цикл 40-60 читает данные из строки 100, начиная с названия первого месяца и до названия месяца с номером **N**, оставляя это "N-ое" название последним значением переменной **M$**.

Напомним, что признаком символьной переменной является использование в ее имени символа **$**.

Программа выводит название указанного месяца, после чего возвращается к строке 10.

Оператор **RESTORE** 100, таким образом, будет выполняться каждый раз перед входом в цикл чтения.

Теперь Вы видите, что оператор **RESTORE** позволяет считывать одни и те же данные столько раз, сколько потребуется.

Для выхода из программы необходимо нажать CTRL + STOP.



# Константы, переменные и выражения

Глава 2

Эта и последующие главы составляют систематическое введение в MSX Бейсик. Объяснения сгруппированы по категориям, а не в алфавитном порядке по ключевым словам, как принято в кратких справочниках. Те пользователи, которые хотят оценить богатые возможности языка в целом, должны внимательно изучить главы с 1 по 8. После этого можно переходить к последующим главам, в которых специфические возможности языка и методы программирования рассмотрены более детально.

## 2-1. Константы

В MSX Бейсике существует два вида констант: символьные (алфавитно-цифровые) и числовые константы.

Числовые константы - это либо положительные, либо отрицательные числа. Они разделены на несколько категорий в соответствии с точностью (максимальным количеством разрядов) и системой счисления.

### 2-1-1. Символьные константы

Символьные константы - это строки длиной до 255 символов. В операторах присваивания они должны быть взяты в кавычки для того, чтобы их можно было отличить от имен переменных. Примеры символьных констант:

```
"ПРИВЕТ"
"#1200"
"Ответ:"
```

В операторах DATA кавычки могут быть опущены, если строка символов не содержит запятых, точек с запятой или двоеточий.
В приведенном ниже примере компьютер различает три отдельных значения символьных данных:

```
DATA год, месяц, день
```

А вот строка "Доброе утро, это почтальон" должна быть взята в кавычки. В противном случае, оператор READ воспримет ее как две отдельных символьных строки:

```
DATA"Доброе утро, это почтальон"
```

Если в следующем примере будут пропущены кавычки, компьютер будет воспринимать только строку "12". Двоеточие здесь рассматривается как признак конца оператора DATA и перехода к следующему оператору, что в данном примере является неверным (мы ведь пытались записать время в часах, минутах и секундах, не так ли?..):

```
DATA"12:14:05"
```

### 2-1-2. Числовые константы с двойной точностью

Константы двойной точности содержат 14 значащих цифр. "Лишние" разряды автоматически округляются компьютером по обычным правилам арифметического округления. Если пятнадцатая цифра равна или больше 5, к четырнадцатой цифре будет добавлена единица; в противном случае, значащая часть числа остается неизменной. По умолчанию в MSX Бейсике числовые переменные обрабатываются с двойной точностью. Как мы увидим далее, существует тип переменной, который позволяет уменьшить количество значащих цифр.





Три способа записи констант с двойной точностью:

(1) Обычное представление:

```
2.2234543
3451100444674
-1.2
.00001
```

(2) Представление, применяемое для научных расчетов (используется в микрокалькуляторах)

```
1.6678D3
-.12355555D-8
55D4
```

Это представление имеет две части: положительное или отрицательное число, содержащее до 14 цифр (мантисса - правильная десятичная дробь), и положительная или отрицательная экспонента (порядок), содержащая до двух цифр с предшествующей им буквой D (Dn эквивалентно *10 ^ n, см. далее знаки операций, раздел 2-3-1).

(3) Использование признака числа с двойной точностью (#)

```
1.234#
345#
```

Это представление позволяет Вам использовать программы, написанные на предыдущих версиях Бейсика, для компьютеров MSX.

### 2-1-3. Числовые константы с одинарной точностью

Константы одинарной точности имеют до 7 значащих цифр. "Лишние" разряды округляются в соответствии со значением седьмой цифры.
Существует два способа записи переменных одинарной точности

(1) Представление, применяемое для научных расчетов:

```
23.89E-5
-3.89876E32
```

Здесь мантисса может содержать до шести цифр, тогда как положительная или отрицательная экспонента (порядок) может содержать максимум две цифры.

(2) Использование признака числа с одинарной точностью (!)

```
2.345!
-321!
```

Примечание:

Представление, применяемое для научных расчетов, называется также "представлением с плавающей запятой"; оно позволяет Вам записывать числа большие, чем 99999999999999. Максимальное число (по абсолютному значению), которое может быть представлено таким способом, равно 9.9999999999999D+63. При попытке работать с числом, большим, чем максимальное, компьютер выведет на экран сообщение об ошибке "OverFlow" (переполнение диапазона представимых в компьютере чисел). Минимальное число (по абсолютному значению), которое может быть представлено таким способом, равно 1D-64. Попытка представления чисел, меньших, чем минимальное, не вызывает сообщения об ошибке, но само число будет округлено до нуля, что подразумевает невозможность его использования в качестве делителя.





## 2-1-4. Целые числовые константы

Целые числовые константы MSX Бейсика принимают значения между −32768 и 32767 (десятичная запятая в них, разумеется, запрещена; кстати, ОБРАТИТЕ ВНИМАНИЕ, что для отделения целой части от дробной в MSX Бейсике используется т о ч к а !). Для записи целых числовых констант используются четыре способа:

(1) По основанию 10 (десятичная система счисления):

```
123%
-4567%
10000%
```

(2) По основанию 16 (шестнадцатиричная система счисления):

```
&H2AEF
&H12
&HFFFF
```

В шестнадцатиричной системе счисления максимальное количество цифр в константе равно четырем, а максимальное значение разряда равно F (15 в десятичной системе счисления). Признак такой константы - буква H.

(3) По основанию 8 (восьмеричная система счисления):

```
&O1453
&O0001
&O177777
```

В восьмеричной системе счисления максимальное количество цифр в константе равно шести, а максимальное значение разряда равно 7. Признак восьмеричного числа - буква O (а не 0 (ноль)!).

(4) По основанию 2 (двоичная система счисления):

```
&B1001
&B001
&B1111111111111111
```

В двоичной системе счисления максимальное количество цифр в константе равно 16, а максимальное значение разряда равно 1. Признак двоичного числа - буква B ( **ОБРАТИТЕ ВНИМАНИЕ**: во всех перечисленных случаях эти буквы вводятся т о л ь к о как латинские!).
Восьмеричная система используется редко. Двоичное представление чисел является удобным для определения образа спрайта или вновь определяемого символа (см., например, разделы с 6-3 по 6-5). Шестнадцатиричная система позволяет записывать данные в очень компактном виде. Мы будем часто использовать ее для записи программ на машинном языке и при определении адресов в памяти.
Читатели, стремящиеся узнать больше об этих системах представления целых чисел, найдут необходимую информацию в Приложении Д.

## 2-2. Переменные

Имя переменной может иметь любую длину. При этом значащимися являются только д в а п е р в ы х с и м в о л а. Использование для именования переменных "осмысленных" слов и аббревиатур помогает, естественно, запомнить назначение самих переменных; однако, нужно учитывать, что имя переменной не должно совпадать с ключевым словом MSX Бейсика. Кроме того, использование букв кириллицы в именах переменных

З А П Р Е Щ Е Н О !





Поэтому пока давайте привыкнем именовать переменные, используя только два символа. Первый символ должен быть буквой; второй может быть как буквой, так и цифрой.

### 2-2-1. Типы переменных

В MSX Бейсике различаются четыре типа переменных. Тип переменной указывается признаком в конце имени. В приведенной ниже таблице перечислены типы переменных и соответствующие им признаки:

| | |
|---|---|
| Целая числовая переменная | % |
| Переменная одинарной точности | ! |
| Переменная двойной точности | # |
| Символьная переменная | $ |

Если тип переменной не указан, считается, что это - переменная двойной точности. Существует возможность переопределения правил записи имен для различных типов переменных с помощью специальных операторов MSX Бейсика:

| | | |
|---|---|---|
| DEFINT | буква(ы) | целое число |
| DEFSNG | буква(ы) | одинарная точность |
| DEFDBL | буква(ы) | двойная точность |
| DEFSTR | буква(ы) | символы |

За этими ключевыми словами должна следовать как минимум одна буква. Использование большего количества букв допускается, но они должны быть разделены запятыми. Можно также определить интервал или несколько интервалов букв для именования переменных (интервальные пары букв также отделяются запятыми).
Интервал записывается с помощью двух букв, разделенных знаком "минус" (—).

**DEFINT A** указывает, что все переменные, имена которых начинаются с A и не имеют признаков !, # или $, являются целыми числами.

**DEFSTR B—I, Z** определяет как символьные все переменные, имена которых начинаются с B по I, или с Z и не имеют признаков %, ! или #.

При определении типа переменной MSX Бейсик действует следующим образом:

(1) Если за именем переменной следует признак типа, то тип переменной определяется в соответствии с этим признаком. Если признак отсутствует, проверяется пункт (2).

(2) Если в программе имеется какой-либо из операторов DEFINT, DEFSNG, DEFDBL или DEFSTR, и его действие затрагивает имя переменной, то ее тип определяется в соответствии с правилами, заданными этим оператором.

(3) Если тип переменной остался не определенным этими двумя проверками, то переменная будет рассматриваться как переменная двойной точности.

**ОБРАТИТЕ ВНИМАНИЕ:**

(1) Операторы **CLEAR** или **RUN** присваивают всем переменным нулевые значения и отменяют эффект использования операторов DEFINT, DEFSNG, DEFDBL и DEFSTR.

(2) Можно иметь в программе переменные с одинаковыми именами, но раз личными типами. MSX Бейсик различает их по признакам.

(3) Во многих программах используются только целые и символьные переменные. В начале таких программ обычно ставится оператор DEFINT A-Z, а символьные переменные указываются индивидуально признаком $. Задание всех числовых переменных как целых значительно сокращает время вычислений, а также уменьшает размер памяти, необходимый для хранения этих переменных.





### 2-2-2. Массивы переменных

До сих пор мы рассматривали только скалярные переменные, т.е. переменные, позволяющие запоминать в программе одиночное число или символьное значение. Массивы переменных, называемые также переменными с индексами, позволяют запоминать несколько значений под одним и тем же именем. Эти переменные используются для записи таких математических объектов, как векторы, матрицы и тензоры. Массив переменных должен быть определен до того, как он будет использоваться. Определение массива выполняется при помощи оператора DIM.

```
DIM A(4,5),B(7),C$(2,6),D
```

Оператор DIM здесь определяет двумерный числовой массив A, одномерный числовой массив B, двумерный массив символьных строк C$ и числовую скалярную переменную D. Все, что было сказано о типах переменных выше, сохраняет силу и для массивов.

Если за именем переменной не стоят круглые скобки, то переменная - скалярная.

Целые числа, взятые в круглые скобки, определяют следующие характеристики массивов:

(1) Размерность массива: одиночная цифра указывает на одномерный массив, две цифры указывают на двумерный массив и т.д. Число индексов (размерность) произвольно, однако следует помнить, что логическая строка ограничена 255 символами.

(2) Количество элементов: B(7) - это одномерный массив с 8 индексированными элементами - индексы от 0 до 7; A(4,5) - это двумерный массив с 30 элементами, т.е. матрица с 5 строками (индексы от 0 до 4) и 6 колонками (индексы от 0 до 5) элементов.

Ниже приведена схема двумерного массива:

DIM A (4,5)

| | | | | | |
|---|---|---|---|---|---|
| A(0,0) | A(0,1) | | | | A(0,5) |
| A(1,0) | | | | | |
| | | | | | |
| | | | | | |
| A(4,0) | | | | | A(4,5) |

При выполнении оператора DIM, в памяти компьютера резервируется пространство для массива, и всем элементам массива присваиваются нулевые значения (проэкспериментируйте с нулевыми значениями символьных строк - что он вам скажет?).
Если массив не может быть размещен в свободной области памяти, то на экран выводится сообщение об ошибке "Out of memory" (или "Out of string memory" для случая массива символьных строк).

При обращении к массиву переменных до его определения его размерность автоматически устанавливается равной 10. Оператор типа A(3)=4, выполненный до того, как массив определен с помощью DIM, будет подразумевать DIM A(10).

Элемент массива определяется именем массива и номером элемента, взятым в круглые скобки (см. схему). Номера (индексы) могут быть заменены выражениями, подразумевающими приемлемые целые значения индексов при их вычислении в ходе выполнения программы.

Для удаления массива A используется оператор ERASE A. Пространство, занимаемое удаляемым массивом, вновь становится доступным.

Попытка переопределения массива без предварительного удаления вызовет сообщение об ошибке "Redimensioned array".





### 2-2-3 Пространство хранения переменных в памяти

В приведенной ниже таблице показано количество байтов, требуемое для хранения скалярных переменных:

| | |
|---|---|
| Целое число | 5 |
| Одинарная точность | 7 |
| Двойная точность | 11 |
| Символьная строка | 6 + пространство для хранения строки |

Для сравнения ниже приведена таблица, показывающая требуемое количество байтов для отдельных элементов массивов:

| | |
|---|---|
| Целое число | 2 |
| Одинарная точность | 4 |
| Двойная точность | 8 |
| Символьная строка | 3 + пространство для хранения строки |

Например, для целой числовой скалярной переменной компьютер запоминает "паспорт", указывающий количество байтов, необходимых для хранения значения переменной (VALTYPE), имя переменной и само ее значение. VALTYPE занимает один байт, имя переменной занимает два байта, само целое число - тоже два байта. Таким образом, общее количество байтов в "паспорте" равно 5.

Для массива же компьютер запоминает имя переменной и различные указатели только один раз. Значения записываются в таком порядке, что элемент может быть найден сразу, как только станет известным его номер.

Каждый раз, когда определяются новые скалярные переменные, компьютер резервирует пространство для хранения переменной. Это влечет за собой перемещение в памяти ранее созданных массивов переменных.

Если таких переменных много (большие массивы), то этот оператор занимает довольно долгое время. Таким образом, настоятельно рекомендуется определять переменные в начале программы, как это показано на следующем примере:

```
10 DIM M(50,50)
20 TIME=0
30 A=1
40 PRINT TIME
```

Теперь заменим строку 10 :

```
10 DIM M(50,50),A
```

и сравним время выполнения программ.

Следующие операторы позволяют Вам проверить размер оставшегося пространства памяти в любой момент выполнения программы:

```
?FRE(0)
?FRE(" ")
```

FRE(0) возвращает количество свободных байтов для хранения программных переменных и для специальной области, называемой стеком. FRE(" ") возвращает количество свободных байтов для хранения строк. Строки хранятся в отдельной области памяти, которая начально имеет размер в 200 байтов. Если Вы хотите расширить пространство для хранения строк до, например, 600 байтов, используйте в начале программы оператор CLEAR 600.





## 2-3. Выражения

Существует три способа присваивания значений переменным:

(1) Оператор READ присваивает значение, найденное в операторе DATA, указанной переменной.

(2) Оператор INPUT присваивает значение, введенное с клавиатуры, указанной переменной.

(3) Специальный оператор присваивания.

Оператор присваивания записывается следующим образом.

LET A=выражение

В существующих версиях Бейсика ключевое слово LET редко является обязательным: оно оставлено из соображений совместимости, и едва ли Вам потребуется использовать его в MSX Бейсике.

В MSX Бейсике Вы можете использовать оператор присваивания в виде:

A=выражение

Выражение может быть просто константой или именем переменной, однако в большинстве случаев оно включает в себя операции и функции.

## 2-3-1. Операции

### (1) Арифметические операции

Арифметические операции всегда выполняются над числовыми операндами. Операция +, соединяющая два символьных операнда, тем не менее, позволяет Вам объединять две строки:

"123"+"456" возвратит "123456"

Следующая таблица содержит список операций в порядке старшинства:

| | |
|---|---|
| ^ | Возведение в степень |
| – | Взятие отрицательного значения (не путать с вычитанием!) |
| * и / | Умножение и деление |
| \ | Целочисленное деление |
| MOD | Вычисление остатка |
| + и – | Сложение и вычитание |

Четыре основные операции и оператор возведения в степень уже рассмотрены в главе 1. Знак минуса не нуждается в каких-либо дополнительных комментариях. Давайте рассмотрим целочисленное деление и операцию вычисления остатка.
Прежде всего, эти операции отбрасывают дробную часть операндов без округления. Если в результате один из операндов выходит за границы допустимой целой константы, то возникает ошибка по переполнению "Overflow".
Затем целочисленное деление сводится к нормальному делению, но частное "усекается" до целого числа. Результатом вычисления остатка также является целая величина, равная остатку от деления целых чисел:

10\4 возвращает 2
22.7\5.1 возвращает 4
5MOD8 возвращает 3 (8\5 равно 1 с остатком 3)





Операция вычисления остатка используется для организации циклических изменений значений целых числовых переменных:

X% = (X% + 1)MOD256

Это выражение позволяет увеличивать X% с 0 до 255. Когда X%=256, значение устанавливается равным нулю.
Два следующих выражения вычисляют старший и младший байты целого числа. После этого их можно запомнить, используя две операции POKE (см. раздел 13-1).

H% = A%\256:L% = A%MOD256

### (2) Операции отношений

Операции отношений применяются для сравнения двух числовых или символьных операндов. Результатом сравнения будет логическая величина "истина" (–1) или "ложь" (0).

Список операций отношений:

| | |
|---|---|
| = | равенство |
| < | меньше чем |
| > | больше чем |
| <> | неравенство |
| <= или =< | меньше либо равно больше либо равно |
| >= или => | |

5 = 4 возвращается 0
5 >= 4.9 возвращается –1
5 <> 5 возвращается 0

Операции отношений наиболее часто используются в операторе IF, но могут быть также использованы в выражениях типа:

IF A=5 THEN A=–2
A=A+7*(A=5)

Оба выражения имеют одинаковый результат: если значение переменной A не равно 5, то оно не изменяется; если A равно 5, то значение A становится равным −2.
Круглые скобки (A>5) необходимы потому, что арифметические операции всегда выполняются перед операциями отношений.
Сравнение строк делает возможной алфавитную классификацию (сортировку). Сравнение производится над ASCII-кодами символов в строках. Перечень кодов ASCII находится в Приложении 3. Помните, что прописные буквы имеют большие коды, чем заглавные буквы.

Все следующие сравнения верны (возвращают −1):

```
"A"<"2"
"A"<"a"
"albert"<"alfred"
```

(3) Логические операции

Логические операции - это операции алгебры логики над каждой парой соответствующих битов двух операндов, которые должны быть целыми числами. Результат логической операции представляется таблицей истинности для пары битов.
Ниже приведен перечень логических операций и соответствующие им таблицы истинности:

| X | 0 | 1 |
|---|---|---|
| NOT X | 1 | 0 |

| X | 0 | 0 | 1 | 1 |
|---|---|---|---|---|
| Y | 0 | 1 | 0 | 1 |
| X AND Y | 0 | 0 | 0 | 1 |

| X | 0 | 0 | 1 | 1 |
|---|---|---|---|---|
| Y | 0 | 1 | 0 | 1 |
| X OR Y | 0 | 1 | 1 | 1 |

| X | 0 | 0 | 1 | 1 |
|---|---|---|---|---|
| Y | 0 | 1 | 0 | 1 |
| X XOR Y | 0 | 1 | 1 | 0 |

| X | 0 | 0 | 1 | 1 |
|---|---|---|---|---|
| Y | 0 | 1 | 0 | 1 |
| X EQV Y | 1 | 0 | 0 | 1 |

| X | 0 | 0 | 1 | 1 |
|---|---|---|---|---|
| Y | 0 | 1 | 0 | 1 |
| X IMP Y | 1 | 1 | 0 | 1 |





Примеры логических операций над целыми числами:

```
10 AND 12 возвращает 8
действительно:          10 = 0000000000001010
                        12 = 0000000000001100
                (10 AND 12) = 0000000000001000 = 8
−1 XOR 3 возвращает −4
действительно:          −1 = 1111111111111111
                         3 = 0000000000000011
                (−1 XOR 3) = 1111111111111100 = −4
```

Если вы хотите узнать побольше о двоичной системе счисления, обратитесь к Приложению Д.
Логические операции наиболее часто применяются в комбинациях двух или большего числа сравнений в операторе IF:

```
IF A > B AND C < B THEN ...
```

Поскольку операции отношений выполняются перед логическими операциями, применение круглых скобок для операций A > B и C < B в этом примере необязательно.
Оператор, следующий за THEN, будет выполнен, если A больше B и C меньше B.
Если A > B, первое сравнение возвращает −1; если C < B, второе сравнение также возвращает −1. −1 AND −1 возвращает −1.

### 2-3-2. Функции

Функции имеют сходство со специальными переменными (см. далее). Компьютер вычисляет значения функций и использует эти значения в тех местах программы, где функции появляются в выражениях. Обычно, аргумент функции также представляет собой выражение.

MSX Бейсик оперирует с разнообразными функциями, а также позволяет пользователю определять свои собственные функции. Сейчас мы рассмотрим "простейшие" функции, то есть функции, близкие к стандартным арифметическим функциям и функции обработки строк. Функции, управляющие экраном, принтером и манипуляторами (джойстик и т.д.), а также внешней памятью, будут рассмотрены в соответствующих главах. В зависимости от типа результата функции подразделяются на два типа:

1. Числовые функции

Большинство числовых функций работают с числовыми аргументами, возвращая в результате числовое значение.
Ниже приведен перечень числовых функций:

**ABS** возвращает абсолютное значение аргумента.
ABS(−5) возвращает 5

**ATN** вычисляет угол, тангенс которого является аргументом данной функции (результат вычисляется в радианах и расположен в первом или четвертом квадранте). 4*ATN(1) возвращает PI (угол, тангенс которого равен 1, состовляет 45 градусов, или PI/4 радиан)

**CDBL** преобразует аргумент в представление с двойной точностью (эта функция сохранена в MSX Бейсике для совместимости)

**CINT** преобразует аргумент в целое значение (аргумент должен лежать в интервале между −32768 и +32767).
CINT(32.9) возвращает 32





**COS** функция вычисления косинуса; аргумент должен быть в радианах.
COS(PI) возвращает −1.

**CSNG** преобразует аргумент в представление с одинарной точностью с округлением последней цифры.

**EXP** возводит число e в степень, указанную аргументом, лежащим в интервале от −149.66803104461 до 145.06286085862.
EXP(1) возвратит значение e (2.7182818284588); заметим, что из-за особенностей машинной арифметики в последних двух цифрах результата - ошибка: мы должны были бы иметь 80 вместо 88)

**FIX** возвращает целую часть аргумента без округления или изменения типа аргумента.
FIX(33.56) возвращает 33 (двойная точность)
FIX(−5.5) возвращает −5

**INT** возвращает максимальное целое число, меньшее аргумента.
INT(5.5) возвращает 5
INT(−5.5) возвращает −6

**LOG** вычисляет натуральный логарифм аргумента (аргумент должен быть положительным).
LOG(100)/LOG(10) возвращает 2 (логарифм по основанию 10)
LOG(X)/LOG(A) возвращает логарифм X по основанию A

**RND** возвращает случайное число в диапазоне от 0 до 1 (1 исключается).

Эта последняя функция должна быть хорошо понята: она является ключевой для многих игр и моделирующих программ. Эта функция генерирует псевдослучайные числа. Хотя такие числа не являются действительно случайными, они обладают свойствами непредсказуемости в достаточной степени.

Аргументом RND может быть как положительное, так и отрицательное число или ноль.

Отрицательный аргумент служит для начальной установки генератора псевдослучайных чисел в RND. Его значение существенным образом определяет свойства последовательности чисел, генерируемых RND.

Нулевой аргумент используется для получения числа, равного последнему числу в последовательности, сгенерированной RND.

Положительный аргумент возвращает число, определяемое последней установкой генератора с помощью отрицательного аргумента. Значение положительного аргумента не важно.

Пример практического использования функции RND:

```
10  FOR I = 0 TO 10
20  PRINT RND(1)
30  NEXT
```

Каждый раз, когда Вы запускаете эту программу, используя RUN, последовательность генерируемых чисел будет той же самой, пока Вы не произведете начальную установку RND при помощи отрицательного числа. Заметим, что обычно для получения псевдослучайных чисел используется именно RND(1) - ведь любой другой положительный аргумент обеспечит такой же результат.

Теперь добавим следующую строку:

```
5 R = RND(-5)
```





Серии чисел, генерируемые этой программой, будут отличаться от серий чисел предыдущей программы, но все запуски измененной программы будут формировать одни и те же серии.

Теперь заменим строку 5 на следующую:

```
5 R = RND(-TIME)
```

Теперь для каждого запуска будут генерироваться различные серии чисел. TIME - это внутренний счетчик MSX Бейсика. Поскольку его значение в каждый момент запуска функции RND(-TIME) оказывается новым, полученные серии будут практически непредсказуемыми.

**SNG** возвращает знак аргумента (1, если аргумент положительный, -1, если аргумент отрицательный и 0, если аргумент равен 0).

| | |
|---|---|
| SGN(45.889) | возвращает 1 |
| SGN(-456) | возвращает -1 |
| SGN(0) | возвращает 0 |

**SIN** функция вычисления синуса; аргумент должен быть в радианах.

| | |
|---|---|
| SIN(0) | возвращает 0 |
| SIN(PI/2) | возвращает 1 |

**SQR** возвращает квадратный корень аргумента (аргумент должен быть положительным).

| | |
|---|---|
| SQR(25) | возвращает 5 |

**TAN** возвращает тангенс аргумента, указанного в радианах.

| | |
|---|---|
| TAN(PI/4) | возвращает 1 |

Следующие функции воспринимают символьные аргументы, хотя и являются числовыми:

**ASC** — возвращает значение ASCII-кода первого символа символьной строки-аргумента, которая не может быть пустой строкой.

ASC("A") возвращает 65
ASC("ABB") возвращает 65 (коды латинских букв!)
ASC(INKEY$) возвращает код любой нажатой клавиши

**INSTR** — возвращает положение подстроки в основной строке.
INSTR(3,"ABCDABEF","AB") возвращает 5

Первый аргумент указывает, что поиск следует начинать с третьего символа. Этот аргумент необязателен. Он должен находиться в диапазоне от 0 до 255. Второй аргумент указывает строку, в которой будет производиться поиск строки, представленной третьим аргументом. Второй и третий аргументы являются обязательными.

INSTR("ABCDABEF","AB") возвращает 1

Поскольку начальная позиция поиска не была указана, поиск начинается с первого символа.

INSTR("ABCDABEF","AZ") возвращает 0

Строка "ABCDABEF" не содержит "AZ".

INSTR("ABCDABEF","") возвращает 1

Пустая подстрока всегда будет найдена в любой строке.

INSTR("","AB") возвращает 0

Непустая подстрока не может быть найдена в пустой строке.





**LEN** — возвращает количество символов, содержащихся в аргументе.

LEN("ABCD") возвращает 4
LEN("") и возвращает 0

**VAL** — возвращает числовое значение числа, записанного символьной строкой.

VAL("123") возвращает 123
VAL("123"+"123") возвращает 123123
VAL("&H"+"10") возвращает 16(=&H10)
VAL("&B"+"10") возвращает 2(=&B10)

ОБРАТИТЕ ВНИМАНИЕ:

VAL("ABC") возвращает 0

Строка "ABC" не может рассматриваться как число.

VAL("12abC") возвращает 12

Начало строки ("12") может рассматриваться как число, остальная часть строки игнорируется.

VAL("abC12") возвращает 0

Части строки - такие, как "a", "ab", "abC", "abC1", "abC12" - не могут быть восприняты как числа.

Примечание:

Ряд выражений для записи математических функций, отсутствующих в MSX Бейсике (обратные тригонометрические, гиперболические функции), приведен в Приложении B.

(2) Функции обработки строк

Результатом действия этих функций являются символьные строки. Имена этих функций всегда сопровождаются признаком $, что указывает на связь с именами символьных переменных. Аргументы здесь могут быть как числовыми, так и символьными.

Ниже приведен перечень символьных функций:

**BIN$** преобразовывает целое число в строку, содержащую двоичное представление данного числа
BIN$(10) возвращает "1010"
BIN$(-1) возвращает "1111111111111111"

Аргумент должен находиться в диапазоне от −32768 до 65535. При использовании отрицательного аргумента будет получено двоичное дополнение (см. Приложение Д).
При использовании положительного аргумента возвращается нормальное двоичное представление.

BIN$(65535) возвращает "1111111111111111", как и при BIN$(−1)

**CHR$** возвращает символ, чей ASCII-код представлен в качестве аргумента; аргумент должен лежать в интервале от 0 до 255.
CHR$(65) возвращает "A" (буква латиницы)

Эта функция применяется и для имитации эффекта некоторых командных клавиш (см. таблицу управляющих кодов в Приложении А):





PRINT CHR$(7) выдает кратковременный звуковой сигнал

★ Для выдачи на экран графического символа, код которого равен 7, введите:

PRINT CHR$(1)+CHR$(7+64)

Этот прием можно использовать для всех символов, имеющих коды от 2 до 31.

**HEX$** преобразовывает целое число в строку шестнадцатиричных кодов; эта функция похожа на BIN$.
HEX$(255) возвращает "FF"
HEX$(-1) возвращает "FFFF"
HEX$(65535) возвращает "FFFF"

**LEFT$** возвращает подстроку указанной длины, начиная с левого края строки, указанной в качестве аргумента; второй аргумент должен лежать в интервале от 0 до 255.
LEFT$("abcdef",3) возвращает "abc"
LEFT$("abcdef",8) возвращает "abcdef"
LEFT$("abcdef",0) возвращает ""
LEFT$("",4) возвращает ""

**MID$** возвращает подстроку указанной длины начиная с указанной позиции в строке, заданной в качестве аргумента; первый аргумент - это исходная строка, второй указывает начальную позицию подстроки, а третий указывает количество символов, которые должны быть включены в подстроку; оба числовых аргумента должны лежать в интервале от 0 до 255; третий аргумент необязателен.

MID$("abcdef",2,3) возвращает "bcd"
MID$("abcdef",8,3) возвращает ""
MID$("abcdef",2) возвращает "bcdef"
MID$("",1,1) возвращает ""
MID$("abcdef",2,0) возвращает ""

Эта часто применяемая функция может быть использована в левой части оператора присваивания; в этом случае первый аргумент должен быть именем символьной переменной (поэтому MID$ иногда называют псевдо-переменной).

```
10 A$ = "ABCDEF"
20 MID$(A$,2,5) = "cdef"
30 PRINT A$
```

**OCT$** преобразовывает целое число в восьмеричный формат; эта функция похожа на BIN$.

OCT$(8) возвращает "10"
OCT$(-1) возвращает "177777"
OCT$(65535) возвращает "177777"

**RIGHT$** возвращает подстроку указанной длины, начиная с правого края строки, заданной в качестве аргумента; функция похожа на LEFT$.

RIGHT$("abcdef",3) возвращает "def"

Заметьте, что порядок символов сохраняется, хотя RIGHT$ вычисляет длину справа.





**SPACE$** создает строку пробелов указанной длины, в диапазоне от 0 до 255.

SPACE$(5) возвращает " "

**STR$** преобразует числовой аргумент в строку цифр (в десятичном виде), оставляя пробел слева.

STR$(123) возвращает " 123"
STR$(&HFF) возвращает " 255"

**STRING$** создает строку повторяющихся символов; первый аргумент указывает длину строки от 0 до 255; второй аргумент может быть как строкой, так и ASCII-кодом; если это строка, то будет использован только первый ее символ.

STRING$(5,65) возвращает "AAAAA"
STRING$(5,"A") возвращает "AAAAA"
STRING$(5,"ABCD") возвращает "AAAAA"
STRING$(0,"A") возвращает ""

### 2-3-3. Функции, определяемые пользователем

Функции, определяемые пользователем, используются так же, как и другие функции. Они должны быть определены внутри программы (никогда не определяйте функцию в прямом режиме). Поскольку компьютер должен иметь эти функции в своем распоряжении, когда они появляются в выражении, возьмите за правило определять функции в начале программы. Ниже приведена короткая функция, вычисляющая двоичный код числа от 1 до 255 и дополняющая строку результата необходимым количеством нулей для получения восьмиразрядного результата:

```
10 DEFFNB$(N) = RIGHT$("00000000" + BIN$(N),8)
```

FNB$(10) возвращает 00001010
FNB$(1) возвращает 00000001

В этой крошечной "программе" определена функция FNB$. Как только эта функция определена, она может быть использована либо в прямом режиме, либо в тексте любой программы. Операторы NEW, CLEAR и RUN отменяют определения функций.

В общем случае, определение пользовательской функции может быть любым выражением: оно может включать как обычные функции, так и другие функции, определенные пользователем.

Если Вы допустили ошибку во время определения функции, эта ошибка будет обнаружена только тогда, когда функция появится в выражении. Компьютер будет указывать как на источник ошибки на ту строку текста, где использована неверно заданная функция, а не на ту строку, где эта функция была определена.

### 2-3-4. Точность числовых выражений

Во время вычисления выражения, все переменные и константы сначала преобразуются к единому виду точности: наивысшей точности какой-либо константы или переменной, присутствующей в выражении.

Промежуточный результат вычисляется в этом виде точности. Последнее преобразование происходит перед присвоением значения переменной: результат преобразуется в вид точности этой переменной.

Тригонометрические функции, квадратный корень (SQR), экспоненциальные функции и логарифм всегда вычисляются с двойной точностью. Это безусловное преимущество MSX Бейсика по сравнению с другими версиями Бейсика, которые вычисляют эти функции в виде с одинарной точностью.





Снижение скорости вычислений в значительной степени оправдывается точностью результата, особенно при большом объеме вычислений, где ошибки округления зачастую делают окончательный результат непригодным для дальнейшего использования.

# Редактирование программ

Глава 3

Редактирование подразумевает как первичный ввод программы, так и внесение в ее текст изменений. В главе 1 мы уже обсуждали простейшие приемы редактирования текстов программ с помощью управляющих, или командных, клавиш. Полный перечень этих клавиш и их комбинаций с пояснением функций редактирования приведен в Приложении А.

## 3-1. Просмотр текста программы

Для просмотра программы на экране пользуйтесь командой **LIST**. На экране умещаются лишь самые маленькие программы; поэтому нам нужно научиться просматривать текст программы по частям:

(1) Используйте **LIST** для просмотра программы; Вы увидите, что текст ее "прокручивается", как рулон пленки. Нажатием STOP Вы можете приостановить просмотр для того, чтобы более внимательно изучить нужный участок текста. Повторное нажатие STOP возобновляет просмотр. Для того, чтобы остановить просмотр и перейти в режим внесения изменений в текст на экране, нажмите CTRL + STOP одновременно.

(2) Существует другой способ - прямого указания того, какая часть программы должна быть просмотрена; используйте оператор **LIST** с указанием двух номеров строк, разделенных знаком (–). Один из номеров при этом может быть опущен.

| | |
|---|---|
| **LIST 10 – 100** | просмотр со строки 10 до строки 100 |
| **LIST 100 – 10** | в обратном направлении просмотра нет |
| **LIST 100** | просмотр строки 100 |
| **LIST 100 –** | просмотр со строки 100 до конца программы |
| **LIST – 100** | просмотр с начала программы до строки 100 |

Ввод несуществующих номеров строк не вызывает появления сообщения об ошибке. Номер строки может быть заменен точкой; этот символ заменяет собой номер последней введенной или модифицированной строки, либо номер строки, где было прервано выполнение программы.

Оператор **LIST** может быть частью программы: он прерывает выполнение программы и переключает MSX Бейсик в прямой режим.

В следующем примере, значение переменной A не будет выведено.

```
10  A = 5
20  LIST
30  PRINT A
```

Оператор **LIST** не присваивает нулевых значений переменным, определенным в процессе выполнения запущенной программы. Вы можете запросить значение переменной A после завершения выполнения этой программы. Однако внесение даже самых малых изменений в программу приводит к инициализации всех переменных.

Все вышесказанное относится и к оператору (команде) **LLIST**, который используется для вывода текста программы на принтер. Если принтер по каким-либо причинам не работает, то для продолжения работы с программой достаточно нажать CTRL + STOP.

Примечание:

Программные операторы могут быть введены как заглавными, так и строчными буквами. Тем не менее, при выводе текста программы на экран ключевые слова и имена переменных будут показаны заглавными буквами.





Только текст между кавычками (константы) и текст в операторах **DATA** остается при этом неизменным. Пробелы, оставляемые обычно для улучшения читаемости программы между ключевыми словами и их операндами, не являются в MSX Бейсике обязательными. При просмотре программы эти пробелы не затрагиваются. Некоторые программисты оставляют пробелы между номером строки и текстом оператора (таким образом, например, выделяют границы подпрограмм). Эти "поля" при просмотре сохраняются.

## 3-2. Отладка программ

Очень часто первая версия большой программы оказывается далекой от совершенства.
В тексте присутствуют как обычные описки, так и принципиальные ошибки (в алгоритме, структурах данных и т.п.). MSX Бейсик содержит ряд "инструментов" для обнаружения ошибок.

### (1) Ошибки написания (описки)

Поспешность при вводе текста программы влечет за собой много описок. Эти ошибки могут быть исправлены довольно легко. Запустите программу при помощи команды **RUN**. Первая же ошибка прервет выполнение программы и переведет компьютер в прямой режим, причем на экран будет выведен номер строки, где ошибка прямо или косвенно была обнаружена. Выведите эту строку на экран, используя **LIST** (или **LLIST** для вывода на принтер).

Ошибки написания наиболее часто вызывают сообщения об ошибках "Syntax error", "Type mismatch", "Subscript out of range" или "Illegal function call". Обычно эти ошибки могут быть немедленно обнаружены. Другие сообщения, такие как "NEXT without FOR" или "RETURN without GOSUB", указывают на ошибку косвенно; она находится не в той строке, номер которой будет присутствовать в сообщении об ошибке. Это затрудняет исправление ошибок.

Полезными здесь оказываются такие операторы и функции, как:

ERL функция, возвращающая номер строки, где ошибка была обнаружена в последний раз

ERR функция, сообщающая код последней ошибки

ERROR оператор, который имитирует ошибку

Для повторного вывода последнего сообщения об ошибке вводится

```
ERROR ERR
```

Для вывода номера строки, где произошла последняя ошибка, вводится

```
PRINT ERL
```

(2) Принципиальные ошибки

Когда исправлены все ошибки написания, программа работает без выдачи сообщений об ошибках. Если Вы скопировали программу, которая ранее была отлажена (например, программа из книги или журнала), то она после исправления описок готова к работе (хотя и в таких программах Вы от ошибок не защищены). Новая программа, с другой стороны, может работать очень "гладко", но, не выполняя в точности ожидаемых действий. Такие программы содержат упомянутые выше





принципиальные ошибки. Для того, чтобы определить этап, на котором происходит ошибка, начинают проверять промежуточные результаты. Структура программы должна быть такой, чтобы она облегчала этот процесс (проверяются обычно передачи управления, условия и т.д.). Чтобы проверить промежуточные результаты, достаточно поместить операторы STOP в нужных местах программы. Эти операторы прерывают выполнение программы, переводят компьютер в прямой режим и выводят сообщение "Break in...", содержащее номер строки (как при нажатии клавиш CTRL + STOP).

На этом этапе лучший способ отладки — это просмотр программы по частям и проверка значений определенных переменных (тех, которые приобрели неверные значения). Отличающееся от ожидаемого значение всегда говорит о том, что произошла ошибка.

```
10  A = 5
20  STOP
30  PRINT A
```

Для продолжения выполнения программы с того места, где она была прервана, используйте команду CONT или нажмите F8 (если Вы не модифицировали программу). Чтобы проверить, как работают остальные участки программы, когда исправлены значения переменных, можно изменять значения переменных в прямом режиме. Если Вы модифицировали программу, сообщение "Can't continue" появится при попытке использовать CONT. Вы можете передать управление программной строке, следующей за STOP, при помощи оператора GOTO.

Трассировка выполнения программы включается с помощью оператора TRON. При выполнении программы теперь выводятся номера программных строк по мере их выполнения.

```
10  FOR I = 0 TO 2
20  FOR J = 0 TO 1
30  NEXT J
40  NEXT I
```

Выполнение этой программы после включения трассировки **TRON** будет продемонстрировано на экране следующим образом:
Протокол трассировки

**[10] [20] [30] [30] [40] [20] [30] [30] [40] [20] [30] [30] [40]**

Эта информация помогает Вам проверить прежде всего правильность выполнения передач управления. Выключить трассировку можно, используя оператор **TROFF**. Этими операторами нужно пользоваться избирательно и крайне осторожно - иначе Вы "утонете" в протоколах трассировки. Стремитесь предельно ограничить область действия **TRON/TROFF** в тексте программы.

Проверка алгоритма работы программы и значений важных переменных возможна также путем включения коротких сообщений в критических точках (точки переходов, возвраты из подпрограмм и т.д.).

```
110 PRINT"LINE 110;A=";A;"B=";B
```

### 3-3. Процедуры ввода и отладки

Как вводить программу с клавиатуры:

(1) Перед вводом новой программы всегда используйте команду **NEW**. Это предотвратит "перемешивание" новой программы со старой.

(2) Номера строк должны быть в диапазоне от **0** до **65529**.





(3) Когда Вы вводите программу, используя уже написанный текст, то можете воспользоваться операцией автонумерации строк. Для этого введите команду **AUTO** или нажмите F2. Если это необходимо, укажите номер начальной строки для автонумерации и величину автоувеличения номеров строк (по умолчанию оба этих значения равны 10). Точка вместо номера строки подразумевает текущий номер строки.

| | |
|---|---|
| **AUTO 10,5** - | начиная со строки 10, увеличивать на 5 |
| **AUTO 100** | начиная со строки 100, увеличивать на 10 |
| **AUTO.** | начиная с текущей строки, увеличивать на 10 |

Эта команда присваивает номера каждой очередной строке автоматически, после нажатия RETURN при вводе предшествующей строки. Если сформированный таким образом номер строки указывает на строку, уже существующую, то после номера строки появится символ звездочки (*), который исчезнет при последующих просмотрах программы. Для отмены **AUTO** нажмите CTRL + STOP.

(4) Для удаления программных строк служит команда DELETE.

| | |
|---|---|
| **DELETE 10-100** | удаляет строки с 10 по 100 |
| **DELETE 100** | удаляет строку 100 |
| **DELETE -500** | удаляет все строки до 500 |
| **DELETE 50-** | удаляет строки с 50 до текущей строки или до конца текста |

Если какие-либо из указанных строк отсутствуют в программе, выводится сообщение **"Illegal function call"**. При этом ничего из текста не удалется.

(5) Номер строки может быть изменен вручную: выведите строку на экран, введите новый номер на место старого и удалите строку со старым номером. Будьте осторожны: если где-нибудь управление передается на строку со старым номером, это также должно быть исправлено.
Для перенумерации группы строк используйте команды **RENUM**.

RENUM 100, 10, 5 перенумерует каждую строку с шагом 5, начиная со строки 10, причем в качестве нового начального номера строки будет взят номер 100.

Формат этой команды следующий:

RENUM новый номер, старый номер, интервал увеличения строк

ОБРАТИТЕ ВНИМАНИЕ: новый номер строки должен быть большим или равным старому номеру.

Значение по умолчанию для всех параметров команды равно 10. Вместо нового или старого номера может быть использована точка; тем самым указывается номер последней использованной строки.

RENUM 5 новый номер = 10 ; старый номер = 5 ; интервал = 10

RENUM 20 новый номер = 20 ; старый номер = 10 ; интервал = 10

Эта команда может быть очень полезна, так как она "подстраивает" все вхождения в интервал передачи управления. Если какие-либо операторы передают управление несуществующим строкам, что вызовет ошибки при запуске программы, перенумерация не произойдет и компьютер выведет сообщение об ошибке. Эта проверка выполняется даже в том случае, если перенумерация невозможна из-за "хитрого" выбора номера строки.

Так, RENUM 65000 не перенумерует ничего, но укажет Вам на все пропущенные номера строк (предполагая, что 65000 больше, чем последний номер строки в Вашей программе).

### 3-4. Функциональные клавиши

Функциональные клавиши используются в прямом режиме. Они позволяют вводить короткие тексты одним нажатием. Тексты, вводимые этими клавишами, выводятся в нижней строке экрана (они могут оказаться "урезанными" из-за недостатка места).





Операторы KEYON и KEYOFF включают и выключают индикацию текстов функциональных клавиш.

KEYOFF выключает тексты функциональных клавиш
KEYON восстанавливает их

Для получения перечня функциональных клавиш и соответствующих им текстов используйте оператор KEYLIST.

| | | |
|---|---|---|
| F1 | color | ожидание ввода новых цветов |
| F2 | auto | переход в режим автонумерации строк |
| F3 | goto | ввод goto и ожидание ввода номера строки |
| F4 | list | |
| F5 | run | ввод list и ожидание параметров |
| F6 | color 15,4,7 | запуск программы |
| F7 | cload" | переустановка цветов на исходные |
| F8 | cont | ввод cload" и ожидание ввода имени файла (для загрузки файла с кассетного магнитофона) |
| F9 | list. | перезапуск программы, остановленной с помощью STOP или CTRL + STOP |
| F10 | run | вывод текущей строки и перевод курсора в начало этой строки |
| | | очистка экрана и запуск программы |

Полный текст не выводится, потому что в некоторых из текстов присутствуют невыводимые символы. Например, полный текст клавиши F10 содержит код CLS, выводимые символы run и код RETURN.

Для переопределения текста клавиши F1 введите следующую команду:

KEY1, текст

Длина текста не должна превышать 16 символов; символы могут быть введены либо обычным способом, в кавычках, либо с помощью функции CHR$ (код символа). Последний способ необходим для невыводимых символов.

Примеры:

KEY1,"CLS"+CHR$(13) очищает экран

Нажатие F1 после ввода этой команды выводит на экран CLS, а затем имитирует нажатие клавиши [RETURN], имеющей код 13 (см. таблицу в Приложении А). Обратите внимание, что обе части текста объединяются с помощью знака (+). Этот знак не рассматривается как символ текста.

Текст этого примера содержит 4 символа. Тот же эффект достигается при использовании:

KEY1,CHR$(12)+CHR$(13)

12 - это код клавиши CLS (или [CTRL] + [L]). В этом случае текст содержит только два символа, оба из которых - невыводимые.

Использование функциональных клавиш очень полезно на этапе редактирования программы. Одна из клавиш может быть специально определена для ускорения ввода большого количества чисел в строках DATA.

KEY1,"DATA0000000"+CHR$(13)

Предположим, что Вы хотите ввести большое количество чисел в восьмибитном виде. Это может быть необходимо при определении образа символа или спрайта (см. главу 6).

Определите клавишу F1 или какую-нибудь другую клавишу, как это описано выше, а затем введите команду AUTO. Нажмите F1 несколько раз для получения начальных "шаблонов" DATA с последовательными номерами строк. Здесь нет никакого риска ошибиться при вводе. Вам остается только отредактировать те строки, в которых необходимо заменить 0 на 1.



# Условные и безусловные переходы



О переходах (передачах управления в программах) уже говорилось в главе 1. Операторы перехода предназначены для изменения последовательности выполнения программных строк; необходимость таких изменений возникает в любом нетривиальном алгоритме.
Операторы переходов подразделяются на два типа:

(1) Безусловные переходы: выполняются с неизменным результатом каждый раз и передают управление в одну и ту же точку программы

(2) Условные переходы: выполняют передачу управления в различные точки программы в зависимости от результатов проверяемых в этих операторах условий

## 4-1. Безусловные переходы

К безусловным переходам относятся операторы GOTO и GOSUB. В GOTO или GOSUB указывается точка перехода - <номер строки>, которая должна существовать в тексте программы. Описанные в главе 3 пользовательские функции можно также рассматривать как одну из разновидностей безусловного перехода.

### 4-1-1. Оператор GOTO

Использование оператора GOTO подвергается критике; созданы языки и приемы программирования, основанные на полном отказе от применения GOTO. Однако, поскольку в большинстве версий и диалектов Бейсика этот оператор сохранен, им пользуются, хотя и с большой осторожностью.

Так, если Вы поместите подпрограммы в начале основной программы, то их следует "обойти" при помощи оператора GOTO.
Пример:

```
10    GOTO 1000
100   REM подпрограмма 1
200   REM подпрограмма 2
300   REM подпрограмма 3
1000  REM основная программа
```

Оператор REM помещает в программный текст комментарии. MSX Бейсик при обнаружении REM не выполняет никаких действий; текст комментариев предназначен для человека, читающего программу, а не для компьютера. Рекомендуется применять REM, в частности, перед подпрограммами для определения их назначения (подобно аннотациям). Ключевое слово REM может быть заменено на символ апострофа (').
В приведенном примере подразумевается, что на месте многоточий находятся сами тексты подпрограмм - выполнимые операторы, которые и нужно "обойти".

Использование оператора GOTO <номер строки> особенно полезно при отладке участков программы в прямом режиме, начиная с указываемой в GOTO строки. Можно использовать и оператор RUN <номер строки>, однако следует помнить, что он инициализирует все переменные.

Пример:

```
10  A=50
20  PRINT A
```





Сначала используйте RUN. Эта программа присвоит переменной A значение 50. Это значение сохраняется в памяти и может быть проверено при помощи команды ?A. Попробуем использовать GOTO 20. Значение переменной A будет снова выведено на экран.

Теперь попробуем RUN 20. Выведенное значение A теперь равно 0, потому что по команде RUN неявно выполняется оператор CLEAR.

Примечание:

В ранних версиях Бейсика оператор GOTO записывался как GO TO. Двухсловная форма GO TO также разрешена к применению в MSX Бейсике, однако ее использовать не рекомендуется, так как она требует большего объема памяти для хранения. Достигаемая экономия на первый взгляд несущественна, однако рано или поздно многие программисты попадают в ситуацию, когда "нескольких байтов не хватает!..".

### 4-1-2. Оператор GOSUB

Оператор GOSUB является важнейшим инструментом построения программ. Во всех хорошо структурированных программах используются операторы GOSUB. Обычно, программы разделяют на подпрограммы

(1) для облегчения понимания логики программ

На практике Вы убедитесь, что внесение изменений в программы занимает гораздо больше времени, чем их составление и запись. Поэтому очень важна хорошо продуманная, ясная структура программы.

Деление программы на подпрограммы делает программу более понятной. В результате основная программа выглядит как цепочка операторов GOSUB. Подпрограммы не должны занимать больше одной страницы экрана. Если это невозможно, постарайтесь разбить подпрограмму на серию более простых подпрограмм.

(2) для однократной записи последовательности операции, которые должны выполняться несколько раз

Простые примеры использования подпрограмм имеются в главе 1.

Примечание:

(1) Подпрограммы обязательно должны оканчиваться оператором RETURN. Этот оператор возобновляет выполнение программы с оператора, следующего за "вызывающим" оператором GOSUB. Оператор продолжения может располагаться в той же самой программной строке, где расположен "вызывающий" GOSUB.

(2) Оператор RETURN может иметь параметр <номер строки>, указывающий, куда нужно передать управление. Этой возможностью следует пользоваться так же осторожно, как и оператором GOTO, поскольку запутывется логика программы.

(3) Если нарушено согласование операторов RETURN и GOSUB, то выводится сообщение об ошибке "RETURN without GOSUB".

(4) Если Вы располагаете подпрограммы в конце основной программы, то отделяйте их от программы оператором END.

(5) RETURN и END могут рассматриваться как безусловные переходы: RETURN для перехода к явно указанной строке или к оператору, расположение которого известно компьютеру; END для перехода в прямой режим.





GOSUB удобно использовать и в прямом режиме. Для проверки новой подпрограммы введите значения переменных, используемых в подпрограмме, в прямом режиме, а затем запустите ее на выполнение с помощью оператора GOSUB. Введите следующую программу и задайте значение R (например, R=1), а затем введите GOSUB 1000. RETURN возвратит систему в прямой режим.

```
1000 "Объем сферы"
1010 VS=16*ATN(1)*R^3/3
1020 PRINT VS
1030 RETURN
```

### 4-1-3. Еще о функциях, определяемых пользователем

Когда необходимо выполнить простые вычисления, которые могут быть записаны в виде выражений, использование функций, определяемых пользователем, более удобно, нежели применение подпрограмм.

Вызов пользовательской функции равносилен передаче управления на то место в программе, где эта функция была определена. Вычисление объема сферы (см. подпрограмму, приведенную выше) может быть выполнено с использованием следующей функции:

```
10 DEFFNVS(R)=16*ATN(1)*R^3/3
```

У этого приема есть и недостатки: невозможно, скажем, включить в пользовательскую функцию операторы типа PRINT.

## 4-2. Условные переходы

Условные переходы позволяют программе реагировать на различные условия (значения определенных переменных). Выполнение такого перехода означает выполнение условия. Существует два способа записи условных переходов:

(1) оператор IF THEN ELSE

(2) оператор ON GOTO (GOSUB)

### 4-2-1. Оператор IF THEN ELSE

Формат этого оператора следующий:

IF<условие>THEN<оператор(ы)1>ELSE<оператор(ы)2>

Если условие выполняется, происходит выполнение операторов группы 1; в противном случае управление передается операторам группы 2. Применение конструкции ELSE <оператор(ы) 2> не является обязательным: если эта конструкция отсутствует и не выполняется условие, то управление переходит к следующей строке. Условием может быть любое числовое выражение. Оно рассматривается как невыполненное (false - ложь), когда значение выражения равно нулю. В противном случае условие считается выполненным.

IF THEN ELSE рассматривается как условный переход только в случае, если операторы 1 и 2 являются операторами переходов, либо группы операторов 1 и 2 заканчиваются операторами переходов.





Пример (1):

```
IF A=1 THEN GOSUB 1000 ELSE GOSUB 2000
```

Здесь вызывается подпрограмма, начинающаяся со строки 1000, если A=1, или подпрограмма,начинающаяся со строки 2000, если A<>1. В обоих случаях после выхода из выполненной подпрограммы управление передается следующей строке после оператора IF THEN ELSE.

Пример (2):

```
IF A THEN GOSUB 1000
```

Условием оценивается по значению переменной A: если оно равно нулю, управление передается следующей строке, в противном случае вызывается подпрограмма 1000.

Пример (3):

```
IF A<0 THEN A=0:GOSUB 1000
```

Перед вызовом подпрограммы 1000, значение A устанавливается равным нулю, если оно было отрицательным. Если A является положительным числом, условие не выполняется, и управление передается следующей строке; значение A при этом остается неизменным.

В любой из групп операторов можно употребить и GOTO: само ключевое слово GOTO может быть опущено, если в группе других операторов нет.

Пример (4):

```
IF A=4 AND B=2 THEN 200 ELSE 150
```

При выполнении условия управление передается строке 200, а в противном случае - строке 150.

Пример (5):

```
IF A<0 THEN A=0:GOTO 200
```

Здесь слово GOTO не может быть опущено, так как в группе 1 есть оператор A=0.

ОБРАТИТЕ ВНИМАНИЕ:

Мы знаем, что возможна запись нескольких операторов в одной логической строке - они разделяются двоеточием. Однако операторы, следующие за IF THEN ELSE в логической строке, игнорируются.

Пример (6):

```
10  IF A=0 THEN GOSUB 100: PRINT A
20  END
100 RETURN
```

В этом примере значение A не будет выведено.

Поэтому дурная привычка "набивать" логическую строку до предела при работе с оператором IF THEN ELSE особенно опасна!

Операторы групп 1 и 2 могут в свою очередь быть операторами типа IF THEN ELSE.





Пример (7):

```
IF A>0 THEN 1000 ELSE IF A<0 THEN 2000
```

Если A>0, управление передается строке 1000; в противном случае продолжается анализ значения A. Если A<0, управление передается строке 2000. Если A=0, происходит выполнение следующей строки.

### 4-2-2. Операторы ON GOTO и ON GOSUB

Операторы ON GOTO и ON GOSUB позволяют организовать сложные разветвления в программе и при этом обычно более понятны, чем последовательность операторов IF THEN ELSE.

Формат этих операторов следующий:

```
ON<выражение>GOTO(илиGOSUB)<N1,N2,N3,N4...>
```

Выражение должно быть числовым и возвращать значения в диапазоне от 0 до 255.

Целая часть этого значения определяет, какой (по порядку) из номеров строк, указанных за ключевым словом GOTO или GOSUB, будет выбран для перехода. Если результат равен нулю или больше, чем количество указанных в списке <N1,...> номеров строк, то ни один из переходов не выполняется.

Пример:

```
ON A GOSUB 1000,2000,3000
```

Если A=0, переход не произойдет. Если A=1, управление будет передано строке 1000; если A=2 - строке 2000; если A=3 - строке 3000. Если A>3, то не будет выполнен ни один из переходов, и управление передается следующей программной строке (как и при A=0).

Сравните эту конструкцию с "лавиной" операторов IF THEN ELSE:

```
IF A=1 THEN GOSUB 1000 ELSE IF A=2 THEN
GOSUB 2000 ELSE
IF A=3 THEN GOSUB 3000
```

Эквивалент приведенной ниже конструкции

```
IF A=1 THEN 1000 ELSE IF A=2 THEN 2000 ELSE
3000
```

с помощью ON GOTO записать гораздо легче:

```
ON A GOTO 1000,2000:GOTO 3000
```

<Выражение> в ON GOTO может быть "условным":

```
ON 2+(A>0) GOTO 1000,2000
```

Выражение равно 1, если A>0. В противном случае оно равно 2.

Примечание:

ON GOTO (GOSUB) воспринимает столько номеров строк в списке, сколько Вы сможете записать в одной логической строке ( помните, что она ограничена 255 символами).

### 4-2-3. Прерывания

Операторы переходов, рассмотренные до сих пор, выполняются в программе "в порядке общей очереди".

Существует другая разновидность условных переходов, когда переход выполняется в зависимости от определенных внешних событий и не связан с общей последовательностью строк в программе. Эта разновидность переходов называется прерыванием и будет рассмотрена позднее.





### 4-2-4. Циклы FOR NEXT

Циклы FOR NEXT уже упоминались в главе 1.

Формат оператора FOR следующий:

```
FOR <счетчик>=<E1> TO <E2> STEP <E3>
```

Переменная <счетчик> должна быть числовой. E1,E2 и E3 - числовые выражения. E1 определяет начальное значение счетчика, E2 - его конечное значение. E3 указывает шаг изменения значения переменной <счетчик>. Если параметр E3 не определен, значение по умолчанию равно 1. Если E1>E2, шаг изменения должен быть отрицательным.

Пример (1):

```
10 FOR I=0 TO 10
20 PRINT I
30 NEXT I
```

Эта программа выводит на экран значения 0,1,2...10.

Пример (2):

```
10 FOR I=0 TO 10 STEP 3
20 PRINT I
30 NEXT I
```

Эта программа выводит на экран значения 0,3,6,9.

Пример (3):

```
10 FOR I=10 TO 5 STEP -2
20 PRINT I
30 NEXT I
```

Эта программа выводит на экран значения 10,8,6

Оператор NEXT увеличивает (или уменьшает) счетчик в соответствии со значением E3. Если новое значение больше (или меньше), чем значение, определенное в E2, управление передается оператору, следующему за NEXT; в противном случае управление передается оператору, следующему за FOR.

Все операторы программного цикла могут быть собраны в одной строке, с учетом ограничения размера логической строки. Такая строка может быть выполнена в прямом режиме.

Пример (4):

```
FOR I=134 TO 142:PRINT BIN$(I):NEXT I
```

Циклы могут быть включены ("вложены") один в другой.

Пример (5):

```
10  FOR I=0 TO 2
20  FOR J=0 TO 3
30  PRINT I+J
40  NEXT J,I
```

ОБРАТИТЕ ВНИМАНИЕ:

(1) Имена счетчиков всех вложенных циклов должны быть различными.

(2) Оператор NEXT должен обрабатывать сначала счетчики внутренних циклов. Вообще имена счетчиков в NEXT необязательны и указываются из соображений наглядности (подобно комментариям).





Строка 30 в примере (3) может быть записана следующим образом:

```
30  NEXT
```

В этом случае оператор NEXT сам выделяет счетчик выполняемого оператора FOR.

Строка 40 в примере (5) может быть записана:

```
40  NEXT: NEXT
```

В соответствии с правилами, оговоренными выше, первый оператор NEXT работает с J, а второй - с I. Оператор NEXT, который не сопровождается именем счетчика, выполняется быстрее, однако делает программу менее понятной.

Примечания:

(1) Информация, необходимая для выполнения цикла, временно запоминается в области памяти, называемой стеком (25 байтов на цикл). Если программа использует слишком много циклов одновременно, может возникнуть сообщение об ошибке "Out of memory". При завершении каждого цикла в памяти освобождаются 25 байтов.

(2) Значения E1, E2 и E3 вычисляются по время выполнения оператора FOR. Если, например, E2 является переменной, изменения этой переменной внутри цикла не будут влиять на количество выполнений "тела" цикла.

Пример:

```
10   A=10
20   FOR I=0 TO A
30   A=A-I:PRINT I
40   NEXT
```

Эта программа выводит на экран целые числа от 0 до 10, хотя A изменяется в процессе выполнения тела цикла.

(3) Исполнение цикла должно заканчиваться последним NEXT, а начинаться с FOR. В противном случае, даже если это не вызывает сообщения об ошибке, поведение программы становится непредсказуемым.



# Работа с внешней памятью

Глава 5

Под словом "данные" здесь подразумевается содержимое памяти компьютера, включая саму программу. Принято различать текст программы, исходные данные для работы программ и результаты работы программ.

Работа с внешней памятью подразумевает запись и чтение данных, размещаемых на внешних носителях информации - кассете магнитной ленты или гибком магнитном диске.

## 5-1. Сохранение и загрузка программ, написанных на MSX Бейсике

### 5-1-1. Сохранение программ

#### (1) команда CSAVE

Команда (или оператор в программе) CSAVE сохраняет программу, записывая ее на кассету магнитной ленты.

Формат оператора:

CSAVE"имя программы";<скорость>

В имени программы допускается использование любых символов. Значащими в имени являются только первые шесть символов.

Второй параметр (который может быть опущен) равен 1 или 2. Он определяет скорость записи данных и по умолчанию равен 1.

```
1 → 1200 бод (1200 бит/сек)
2 → 2400 бод
```

Скорость 2400 бод требует использования хорошего магнитофона и высококачественной ленты. Скорость работы по умолчанию может быть переопределена в операторе SCREEN (см. главу 6). CSAVE сохраняет программу во внутреннем формате (т.е. в том виде, в каком она хранится в памяти компьютера).
Примеры:

| | |
|---|---|
| **CSAVE"PROG1"** | Сохраняет PROG1 во внутреннем формате со скоростью 1200 бод (или со скоростью, определенной оператором SCREEN). |
| **CSAVE"PROG1",2** | Сохраняет PROG2 на кассетной ленте во внутреннем формате со скоростью 2400 бод |

**(2) CLOAD?**

Оператор **CLOAD?** проверяет правильность сохранения программы на кассетной ленте.

После сохранения программы, перемотайте ленту назад и введите:

**CLOAD?"имя программы"**

Записанная программа, имя которой указано в операторе, сравнивается байт за байтом с текстом программы в памяти компьютера. При обнаружении несовпадения эта процедура прекращается и на экране появляется сообщение об ошибке "Verify error". Имя программы может быть опущено. В этом случае с содержимым памяти будет сравниваться первая введенная с ленты программа.
Если имя программы не совпадает с именем, указанным в операторе, поиск программы с указанным именем будет продолжен. На экране появится следующее сообщение:





Skip имя программы

В нем указано имя найденной "по пути" программы или файла данных.

**(3) SAVE**

Этот оператор позволяет сохранить программу как на кассетной ленте, так и на гибком магнитном диске (дискете).

Формат оператора:

**SAVE"имя устройства:имя программы",A**

Имя устройства может быть указано как A (основной дисковод), B (дополнительный дисковод) или CAS (магнитофон). Имя устройства по умолчанию - A. Указание имени программы обязательно.
Если программа сохраняется на кассетной ленте, то она записывается в "формате ASCII", т.е. в том виде, в каком она вводилась с клавиатуры, без специальной кодировки чисел и операторов. В этом формате программа может быть вызвана впоследствии оператором MERGE (см. раздел 5-1-2). Если же сохранение производится на дискете, то используется внутренний формат (чтобы записать программу на дискету в формате ASCII, укажите в операторе SAVE параметр A).

Примеры:

| | |
|---|---|
| **SAVE"PROG"** | Дисковод A;внутренний формат |
| **SAVE"B:PROG1",A** | Дисковод B;формат ASCII |
| **SAVE"CAS:PROG1"** | Лента;формат ASCII |

**Примечание:**

Перед сохранением данных на новой дискете выполните операцию форматирования. Вставьте дискету в дисковод и введите:

CALL FORMAT

или FORMAT. Далее следуйте операторам, появляющимся на экране.

**ОБРАТИТЕ ВНИМАНИЕ:**

*Эта операция разрушает любое прежнее содержимое дискеты!*

Следующий оператор позволяет определить объем свободного пространства на дискете:

DSKF (номер дисковода)

0=текущий дисковод
1=дисковод A
2=дисковод B

Имена программ и файлов данных, сохраненных на дискете (см. ниже), выводятся на экран после ввода команды:

FILES

Команда LFILES выполняет ту же процедуру для вывода этой информации на принтер.
С помощью команд FILES/LFILES удобно проверять наличие тех или иных программ либо файлов на дискете. Введите команду с параметром <имя>; если программа с указанным именем есть на дискете, ее имя будет подтверждено выводом на экран.Для удаления программы с дискеты (например, при высвобождении места на дискете), введите команду:





KILL <имя программы>

**Примечание:**

Всегда помните, что загрузка программы стирает любую программу, находящуюся в памяти компьютера. Кроме того, если Вы сохраняете программу, используя имя, которое уже использовалось при записи на эту дискету, Вы удалите существующий файл с таким именем.

### 5-1-2. Загрузка программ

**(1) CLOAD**

Этот оператор позволяет загружать программу, сохраненную на кассетной ленте во внутреннем формате (с помощью CSAVE). Загрузка выполняется на той скорости, на которой программа была сохранена. Таким образом, Вам не нужно повторно указывать скорость.

Если имя программы не указано в операторе,то будет загружена первая же программа, сохраненная во внутреннем формате. Если Вы указываете имя программы, следите за правильностью ввода имени (т.е., например, если Вы использовали заглавные буквы при написании имени, Вы должны ввести имя программы также заглавными буквами). Программы и другие файлы, чьи имена не указаны или они не во внутреннем формате, будут являться причиной появления сообщения о пропуске "Skip", так же, как это было описано для команды CLOAD?.

* Оператор CLOAD автоматически запускает команду NEW перед началом загрузки в память найденной программы.

### (2) LOAD

Этот оператор производит загрузку программы, сохраненной на кассетной ленте в формате ASCII или на дискете (в этом случае - либо в формате ASCII, либо во внутреннем формате). Указание параметра R (RUN) автоматически запускает программу на выполнение после загрузки.

Примеры:

| | |
|---|---|
| LOAD"CAS:PROG1",R | загружает PROG1 с ленты и запускает ее |
| LOAD"CAS:" | загружает первую найденную программу в ASCII формате с ленты |
| LOAD"PROG1" | загружает PROG1 (дисковод A) |
| LOAD"B:PROG1",R | загружает PROG1 (дисковод B) и запускает ее |

Производя загрузку с кассетной ленты, вводите имя программы в точности так, как это было сделано при ее сохранении ( заглавные/строчные буквы). При использовании дисковода, имя программы автоматически преобразуется в форму с заглавными буквами.

### (3) RUN

Этот оператор, если в нем указаны имя устройства и имя программы, загружает и запускает на выполнение программу, сохраненную в формате ASCII. Он похож на оператор LOAD (с параметром R), за тем исключением, что загружает программы только в формате ASCII и закрывает все открытые файлы (см. ниже). В обоих случаях перед загрузкой программы автоматически выполняется команда NEW.





Примеры:

| | |
|---|---|
| RUN"CAS:PROG1" | загружает и запускает PROG1, сохраненную на ленте (используя SAVE"CAS:PROG1") |
| RUN"PROG1" | загружает и запускает PROG1, сохраненную на диске (используя SAVE"PROG1",A) |

ОБРАТИТЕ ВНИМАНИЕ:

В этой главе предполагается, что Ваш компьютер имеет встроенный дисковод. В таком варианте при включении запускается MSX DISK Бейсик, и приоритетным устройством внешней памяти является дисковод A:. Работая на компьютере без дисковода, Вы запускаете при включении MSX Бейсик, а приоритетным устройством является кассетный магнитофон (CAS:), имя которого может опускаться. Учитывая, что компьютеры КУВТ "Ямаха MSX-2" объединены в локальную сеть, Вы должны также изучить возможности загрузки и сохранения программ на MSX Бейсике, предоставляемые программными средствами локальной сети КУВТ.

### (4) MERGE

Этот оператор объединяет текст программы, сохраненной ранее в формате ASCII, с текстом программы, находящимся в памяти компьютера. Наиболее целесообразно использовать этот оператор при работе с библиотеками подпрограмм на ленте или на дискете.

Предположим, что программа, находящаяся в памяти, занимает строки с 10 по 1290. Для добавления подпрограммы, названной при сохранении ROUT1 и занимающей строки с 10 по 130, сначала Вы должны сохранить программу, находящуюся в памяти.

После этого, Вы вызываете ROUT1, используя LOAD или CLOAD. Теперь используйте RENUM для перенумерации строк подпрограммы, начиная, например, со строки 10000. Далее, сохраните подпрограмму в формате ASCII, назвав ее новым именем (ROUT0). После чего наберите загрузите основную программу. Теперь выполнение команды

MERGE"ROUT0"(может быть указано имя устройства),

приведет к тому, что в конце текста основной программы будет добавлен текст подпрограммы с номерами строк, начинающимися с 10000.

**ОБРАТИТЕ ВНИМАНИЕ:**

Программы, вызываемые оператором MERGE, имеют более высокий приоритет, по сравнению с программами, находящимися в памяти компьютера. Поэтому если в объединяемых программах имеются строки с одинаковыми номерами, то строки программы в памяти компьютера будут заменены соответствующими строками "добавляемой" программы.

## 5-2. Файлы с последовательным доступом.

Файл - это набор данных, записываемый на устройстве внешней памяти. Таким образом, программы, сохраняемые на ленте или дискете, являются файлами. Поиск файла на кассетной ленте называется последовательным, потому что компьютер ищет требуемый файл, последовательно перематывая магнитную ленту. При использовании дискеты достаточно указать имя файла, чтобы компьютер начал поиск с использованием каталога. Все участки дискеты в равной степени доступны для записи и чтения, т. е. для обращения к любому участку требуется примерно одинаковое время. В этом случае говорят, что внешняя память характеризуется п р о и з в о л ь н ы м доступом.





Слова "последовательный" и "произвольный" будут использоваться во вполне определенном смысле. Файл будет называться файлом с последовательным доступом, если компьютер должен просматривать его до момента нахождения нужной части с самого начала. Файл, где информация может быть найдена без полного просмотра носителя, будет называться файлом с произвольным доступом.

Используя кассетную ленту, невозможно создать файл с произвольным доступом.

### 5-2-1. Открытие файла с последовательным доступом

Перед использованием файла Вы должны объявить его открытым. Оператор OPEN описывает имя устройства (CAS:, A: или B:), имя файла, направление потока данных и номер, присваиваемый файлу для операций передачи данных.

Примеры:

(1) OPEN"CAS:ADDRESS" FOR INPUT AS #1

Устройство: CAS:
Файл: ADDRESS
Направление: INPUT (файл, сохраненный на ленте, нужно ввести)
Номер файла: 1

(2) OPEN"ADDRESS" FOR OUTPUT AS #1

Устройство: A: (по умолчанию в MSX DISK Бейсике)
Файл: ADDRESS
Направление: OUTPUT (файл должен быть создан на дискете)
Номер файла: 1

(3) OPEN"B:ADDRESS" FOR APPEND AS #2

Устройство: B:
Файл: ADDRESS
Направление: APPEND (добавление к уже существующему на дискете файлу)
Номер файла: 2

Примечание:

(1) Другие имена устройств используются для работы с экраном и принтером (CRT:, GRP: и LPT:). Их использование будет объяснено далее.

(2) Однажды открытый файл не может быть открыт повторно.

(3) APPEND не может использоваться при открытии кассетных файлов.

(4) Знак # может быть опущен.

(5) Максимальное количество файлов, которые могут быть открыты одновременно, устанавливается следующим оператором:

MAXFILES=<количество> (0-15)

При включении компьютера этот параметр равен MAXFILES=1. Таким образом, если Вы хотите открыть только один файл, использование оператора MAXFILES необязательно. Файл #0 всегда открыт. MSX DISK Бейсик использует этот файл для своих нужд (операции LOAD, SAVE и т.п.). При использовании MAXFILES автоматически выполняется CLEAR. Если Вы используете оба оператора в программе, располагайте MAXFILES перед CLEAR.





### 5-2-2. Запись в файл

Для записи данных в файл используется следующий оператор:

PRINT #<номер>,<данные>

Данные могут быть числового или символьного типа. Использование расширения USING позволяет Вам определить собственный формат данных. Использование USING будет рассмотрено в главе, посвященной работе с экраном.

Пример:

```
10 OPEN"TESTF" FOR OUTPUT AS #1
20 FOR I=0 TO 10
30 PRINT #1,I
40 NEXT
50 CLOSE
```

Эта программа создает на диске A: файл TESTF и записывает в него числа от 0 до 10.

### 5-2-3. Чтение из файла

Данные из созданного в предыдущем примере файла могут быть использованы для организации небольшого массива переменных в памяти и его вывода на экран.

Пример:

```
NEW
10 OPEN"TESTF" FOR INPUT AS #1
20 DIM X(10)
30 FOR I=0 to 10
40 INPUT #1,X(I)
50 PRINT X(I)
60 NEXT
70 CLOSE
```

Оператор INPUT# используется для ввода данных из файла. Тип вводимых данных определяется типом переменной, используемой в операторе ввода (в данном случае X(1)).

Для ввода символьных данных можно также использовать следующие операторы:

(1) <Символьная переменная>=INPUT(<длина>,#<номер>);

Первый аргумент определяет количество символов для ввода (до 255).

(2) LINE INPUT #<номер>,<символьная переменная>

В этом случае символьной переменной автоматически присваивается значение в виде строки длиной до 255 символов. Заметим, что этот оператор может использоваться для ввода текстов программ, сохраненных в формате ASCII.

### 5-2-4. Закрытие файла

Закрытие файла позволяет Вам использовать тот же номер для открытия другого файла. При выводе (записи) данных закрытие обеспечивает вывод кода конца файла (End-of-File) в файл, который был открыт ранее в режиме "FOR OUTPUT".

CLOSE <номер>

Если оператор не указывает на какой-либо номер, происходит закрытие всех открытых файлов. Несколько номеров, разделенных запятыми, могут быть указаны одновременно.

Пример:

```
100 CLOSE 1,2
```

Номер закрытого файла может использоваться для открытия другого файла; этот прием особенно удобен, когда Вы хотите использовать в программе только один открытый файл в любой момент (MAXFILES=1).





### 5-2-5. Определение конца файла

Длина файла в примере ниже известна (14 чисел). Однако длину файла не всегда можно определить заранее: представьте себе файл адресов, который все время приходится открывать в режиме добавления данных (FOR APPEND).

Функция EOF позволяет вам решить эту проблему: специальный символ с кодом (1AH) добавляется к концу файла, когда этот файл закрывается. Этот символ помечает конец файла. Функция EOF сравнивает первый нечитаемый символ с кодом 1AH:

X=EOF(<номер>) возвращает −1 (true), если найден код конца файла, и 0 (false) в противном случае.

В следующем примере продемонстрирован способ использования EOF:

```
10  OPEN "TESTF" FOR INPUT AS #1
20  IF EOF(1) THEN 60
30  INPUT #1,A
40  PRINT A
50  GOTO 20
60  CLOSE
```

### 5-3. Файлы с произвольным доступом

Файлы с произвольным доступом не могут быть созданы на кассетной ленте. Они отличаются от файлов с последовательным доступом тем, что они разделены на пронумерованные участки, которые можно вызывать по адресам (номерам).

Операторы, позволяющие работать с такими файлами, приведены ниже:

**(1) OPEN**

Этот оператор открывает файл с произвольным доступом, если спецификации режимов **FOR INPUT, OUTPUT** или **APPEND** опущены.

**(2) FIELD**

Этот оператор выделяет буфер, называемый буфером полей, для переменных. Он должен выполняться перед любыми обращениями к файлу, использующими операторы **PUT** и **GET** (см. ниже).

Пример:

```
FIELD #1,200 AS X$,12 AS Y$
```

Здесь отведено 200 символов для переменной **X$** и 12 - для переменной **Y$**. Заметим, что этот оператор будет выполнен только в случае, если файл был предварительно открыт.

**(3) PUT**

Этот оператор передает данные из буфера полей в файл с произвольным доступом.





Пример:

```
PUT #1,3
```

Второе число (3) определяет номер записи в файле. Если этот номер будет пропущен, то будет использовано число, на единицу большее, чем в последнем выполненном операторе **PUT** или **GET**. Номер записи может находиться в диапазоне от 1 до 4294967295.

**(4) GET**

Этот оператор вводит данные из файла с произвольным доступом.

Пример:

```
GET #1,3
```

Здесь выполняется передача записи с номером 3 из файла #1 в буфер полей. Номер записи может быть опущен, как и в случае **PUT**.

**(5) CVI, CVS и CVD**

Эти функции преобразуют строки символов в числа (в файлах с произвольным доступом информация хранится в виде строк). Аргумент состоит из 2, 4 или 8 байтов и преобразуется в целое число, либо число с одинарной или двойной точностью, в зависимости от вида используемой функции (**CVI, CVS** или **CVD**).

Примеры:

**CVI ("AA")** возвращает 16705

ASCII-код для буквы A – 65. Два байта строки "AA" записываются следующим образом:

01000001 01000001(B) = 16705(D)
CVS("AAAA") возвращает 4.14141

Последние три байта строки "AAAA" имеют следующее представление:

01000001 или 01000001 = 41(двоично-десятичный код BCD)

Первый байт, имеющий такое же представление, соответствует знаку мантиссы (+) и экспоненте 10(1).

Дополнительная информация о представлении чисел в памяти компьютера см. в Приложении Д.

### (6) MKI$, MKS$ и MKD$

Эти функции принимают целое число, число с одинарной точностью и число с двойной точностью соответственно в качестве аргумента.
Они выполняют обратное преобразование по отношению к CVI, CVS и CVD.

Примеры:

MKI$(16705) возвращает строку "AA"
MKS$(4.14141) возвращает строку "AAA"

### (7) LOC

Функция LOC (<номер файла>) возвращает номер записи, использованный в последнем выполненном операторе PUT или GET.





### (8) LSET, RSET

Оба этих оператора передают данные из памяти в буфер полей. Они имеют следующий формат:

LSET (или RSET) <символьная переменная> =
<символьное выражение>

Если выражение содержит меньше байтов, чем выделенный участок буфера полей для этой переменной, то эти байты выравниваются по левой (LSET) или правой (RSET) границе поля. Неиспользованные байты заполняются пробелами.

### (9) VARPTR

VARPTR (#номер файла) возвращает адрес блока управления файлом (FCB).

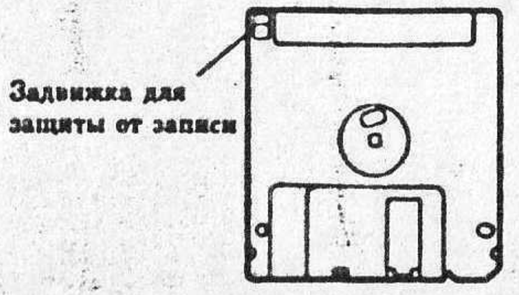
## 5-4. Работа с файлами на дискетах

Несколько советов:

(1) Для защиты файлов, сохраненных на дискете, позаботьтесь о создании копий файлов на другой дискете или на ленте.

(2) Создайте архив файлов программных текстов и снабжайте дискеты либо ленты этого архива этикетками.

(3) Защищайте окончательные версии программ от неосторожного обращения путем выламывания предохранительного ушка на кассете или установкой предохранительной защелки на диске.



Имеются специальные операторы для обслуживания библиотек файлов:

**(1) (L)FILES**

Этот оператор выводит на экран каталог файлов, сохраненных на дискете, независимо от того, в каком формате они сохранены. При указании имени файла, имя будет воспроизведено на экране, если файл с таким именем найден на дискете.

**(2) _FORMAT**

Этот оператор выполняет разметку дискеты, которая необходима для работы с файлами, стирая предварительно всю информацию с этой дискеты.

**(3) KILL**

KILL с указанием имени файла удаляет этот файл с дискеты.

**(4) LOF**

LOF (<номер файла>) возвращает длину (в байтах) указанного файла.

**(5) NAME**

NAME <старое имя> AS <новое имя> изменяет имя файла. Старое имя должно быть именем файла, хранящегося на дискете; новое имя может быть любым, отличным от существующих на этой дискете.

**(6) COPY**

COPY <имя файла-источника> TO <имя файла-приемника> позволяет копировать файлы. Если TO и второй аргумент будут опущены, файл будет скопирован на активный дисковод (обычно A:).



# Работа с экраном

Глава 6

## 6-1. Введение

### 6-1-1. Выбор режима экрана

Режим экрана выбирается при помощи оператора SCREEN. Этот оператор имеет несколько параметров, но мы будем здесь рассматривать только параметры, устанавливающие режим экрана.

| | |
|---|---|
| SCREEN 0 | Текстовый экран 40x24 или 80x24 символов |
| SCREEN 1 | Текстовый экран 32x24 символов |
| SCREEN 2 | Графический экран 256x192 точек |
| SCREEN 3 | Графический экран 64x48 блоков |
| SCREEN 4 | Графический экран 256x192 точек |
| SCREEN 5 | Графический экран 256x212 точек |
| SCREEN 6 | Графический экран 512x212 точек |
| SCREEN 7 | Графический экран 512x212 точек |
| SCREEN 8 | Графический экран 256x212 точек |

ОБРАТИТЕ ВНИМАНИЕ:

Режимы SCREEN 4-8 разрешены только для компьютеров MSX-2. В режиме SCREEN 0 для компьютеров серии MSX разрешен только режим 40x24 символов. Это следует иметь в виду при рассмотрении совместимости КУВТ "Ямаха MSX-2" и КУВТ "Ямаха" поставки 1985 г. Необходимо также учитывать, что некоторые программные средства КУВТ "Ямаха" используют возможности видеоконтроллера 9938, общего для компьютеров этих двух серий.

Примечание:

(1) При включении питания компьютера, неявно выполняется оператор SCREEN 0.

(2) При вводе оператора SCREEN n (n=0-8) всегда очищается экран.

(3) В прямом режиме Вы находитесь в SCREEN 0 или SCREEN 1. Возвращаясь в прямой режим, Вы снова находитесь SCREEN 0 или в SCREEN 1, в зависимости от последнего выполненного оператора. Соответственно, ввод SCREEN 2 в прямом режиме очищает экран перед возвратом в SCREEN 0 или 1. Выполнение оператора SCREEN 2 внутри программы сохранит режим SCREEN 2 на все время выполнения программы, если не будут выполнены другие операторы SCREEN n. По окончании выполнения программы экран очищается (исключение составляют программы, работающие в SCREEN 0 или 1) и переключается в режим SCREEN 0 или 1.

### 6-1-2. Компоненты изображения

Выбор режима работы экрана зависит от того, как Вы собираетесь использовать компьютер.

Примеры:

(1) Когда Вы вводите программу или используете клавиатуру для ввода текстов, Вы хотите видеть как можно большую часть текста. Выбор цвета определяется личным вкусом.





(2) Для игровых программ часто требуется более сложный режим экрана. Возможность выбора из большого количества цветов позволяет Вам наиболее точно передавать различия экранных областей, придать программе привлекательный вид. Вам могут также понадобиться динамические примитивы (спрайты).

(3) В программах научных расчетов очень удобным может оказаться графическое представление результатов. В определенных случаях разрешающая способность графического экрана может быть решающим фактором.

Давайте детально разберемся, какие существуют отличия между режимами экрана. В каждом отдельном случае будут учитываться следующие параметры:

(1) Количество цветов
(2) Параметры отображения текста
(3) Графические возможности (только для случая графических экранных режимов)
(4) Динамические примитивы (спрайты); эта возможность отсутствует в SCREEN 0.

## 6-2. Режим SCREEN 0

Этот режим предназначен исключительно для отображения текстов. Никакие графические операторы здесь не могут быть использованы. Запрещено также использование спрайтов.

При включении питания, компьютер автоматически устанавливается в режим экрана SCREEN 0. Этот режим обычно используется для ввода программ или для работы с программами редактирования текстов.

### 6-2-1. Выбор цвета

Компьютер имеет 16 определенных цветов. Оператор **COLOR** позволяет выбирать два цвета из 16 - один для текста, а другой для фона.

Цвета текста и фона выбираются следующим образом:

**COLOR <Т>,<Ф>**

где <Т> и <Ф> - числа (или выражения) в диапазоне от 0 до 15. Для возможности использования одной из 256 возможных комбинаций (16х16) отображаемых цветов, изучите таблицу в конце раздела.

**Примечание:**

(1) Если цвет текста равен 0 или такой же как и цвет фона, текст исчезает с экрана. В этом случае нажмите F6 для установки цветов в начальное состояние.

(2) Если Вы не имеете в своем распоряжении RGB-монитора, некоторые комбинации цветов на мониторе могут сделать текст неразборчивым.

(3) Для изменения только одного цвета используйте:

```
COLOR,6   Цвет текста не изменяется; красный фон
COLOR ,1  Цвет текста черный, цвет фона остается
          прежним
```

Цвета, пронумерованные от 0 до 15, соответствуют определенным пропорциям красного, зеленого и синего цветов (Red, Green, Blue). Таблица в конце этого раздела содержит сведения о пропорциях RGB в изначально установленных цветах экрана.





В компьютере серии MSX-2 эти пропорции могут быть изменены при помощи следующего оператора:

**COLOR=(<Ц>,<К>,<З>,<С>)**

где <Ц> - номер изменяемого цвета, а <К>, <З>, и <С> - соответственно пропорции красного, зеленого и синего цветов. Последние три цифры должны находиться в диапазоне от 0 до 7. В порядке примера, возьмите для Ц число, соответствующее текущему цвету фона и попробуйте различные пропорции. Для получения чистого красного цвета фона, введите:

**COLOR=(4,7,0,0)**

(предполагается, что текущий цвет фона равен 4).

Таким образом можно получить 512 различных оттенков цвета.

С учетом этого приема нет смысла говорить об абсолютном значении цифр, соответствующих цветам, а более удобно оперировать термином "номер ПАЛИТРЫ".

Оператор **SCREEN** устанавливает палитры в начальное состояние. Для получения того же самого результата без стирания экрана используйте оператор:

**COLOR=NEW** или **COLOR**

**ОБРАТИТЕ ВНИМАНИЕ:**

Палитра не может быть изменена в случае использования компьютеров серии MSX. Операторы типа **COLOR=(4,7,0,0)**, **COLOR=NEW** или **COLOR** без параметров здесь являются запрещенными.

Таблица предустановленных палитр

| Номер | Цвет | К | З | С |
|---|---|---|---|---|
| 0 | Прозрачный | 0 | 0 | 0 |
| 1 | Черный | 0 | 0 | 0 |
| 2 | Средне-серый | 1 | 6 | 1 |
| 3 | Светло-зеленый | 3 | 7 | 3 |
| 4 | Темно-синий | 1 | 1 | 7 |
| 5 | Светло-синий | 2 | 3 | 7 |
| 6 | Темно-красный | 5 | 1 | 1 |
| 7 | Голубой | 2 | 6 | 7 |
| 8 | Средне-красный | 7 | 1 | 1 |
| 9 | Светло-красный | 7 | 3 | 3 |
| 10 | Темно-желтый | 6 | 6 | 1 |
| 11 | Светло-желтый | 6 | 6 | 4 |
| 12 | Темно-зеленый | 1 | 4 | 1 |
| 13 | Лиловый | 6 | 2 | 5 |
| 14 | Серый | 5 | 5 | 5 |
| 15 | Белый | 7 | 7 | 7 |

### 6-2-2. Отображение текста

Режим SCREEN 0 допускает применение большого количества операторов, позволяющих детально управлять отображением текста на экране. Оператор PRINT уже рассматривался в простейшем варианте. Однако, существует большое количество его вариантов, которые мы рассмотрим в этой части главы.

Сначала давайте представим координатную систему экрана, размер отображаемых символов и ширину экрана.

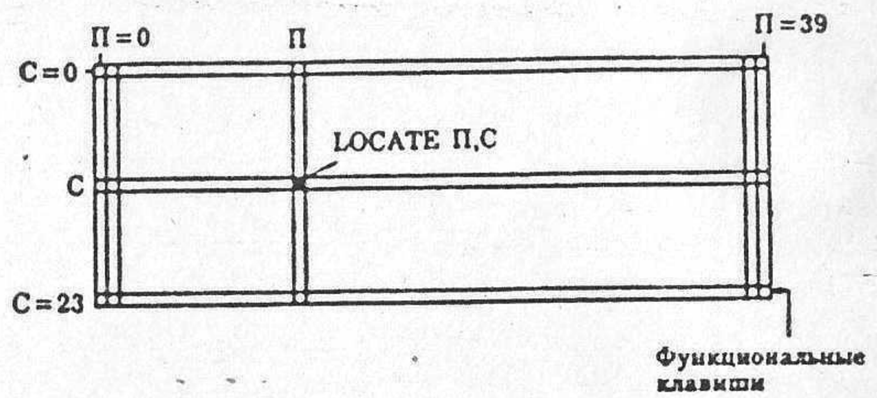




(1) WIDTH

Этот оператор определяет как размер символов, так и ширину экрана. После включения питания, режим экрана - SCREEN 0, а координатная система выглядит следующим образом:

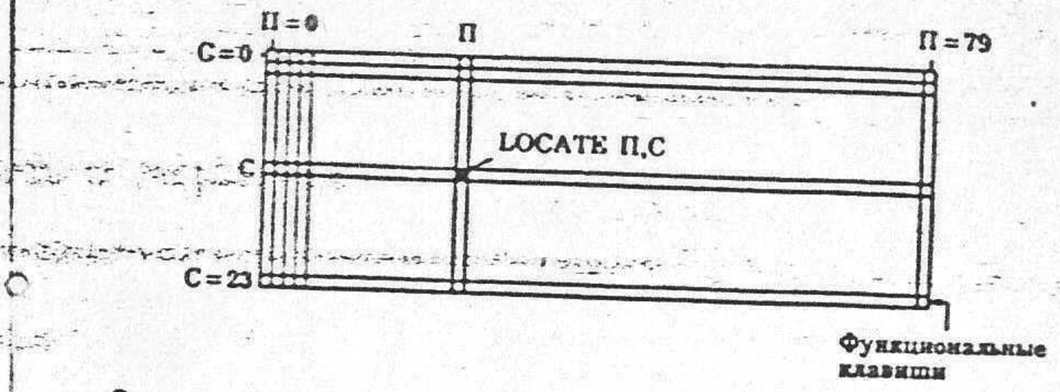
Координатная система в режиме SCREEN 0 (WIDTH 40-ширина экрана равна 40 символам).




24 строки, каждая из которых состоит из 40 символов, отображаются в данном режиме. Заметим, что если функциональные клавиши отображаются на экране, общее количество текстовых строк сокращается до 23. Операторы KEY ON/OFF позволяют вам управлять отображением фукциональных клавиш.

Оператор WIDTH 80, разрешенный только для компьютеров серии MSX-2, позволяет производить отображение 80 символов в одной экранной строке. Ширина символов при этом уменьшается наполовину. Новая координатная система представлена ниже:

Координатная система в режиме SCREEN 0 (WIDTH 80).




Заметим, что разборчивое чтение символов в режиме WIDTH 80 возможно только при использовании RGB-монитора.

Программы обычно редактируются в этом режиме, в особенности, если они имеют длинные строки. Выбор цвета при этом производится исходя из соображений наилучшей разборчивости.

WIDTH позволяет Вам определять ширину экрана.

Пример (1):

```
WIDTH 20
```

20 символов, имеющих нормальный размер в центральной части экрана, могут быть отображены в одной строке.

Пример (2):

```
WIDTH 60
```

60 символов, имеющих половинную ширину, могут быть отображены в одной строке в центральной части экрана.





Аргумент оператора WIDTH должен лежать в диапазоне от 1 до 40 для компьютеров серии MSX и от 1 до 80 для компьютеров серии MSX-2. Если число меньше или равно 40, символы отображаются в нормальном размере; они уменьшаются вдвое по ширине, если число лежит в диапазоне от 41 до 80.

(2) LOCATE

Этот оператор производит установку курсора в требуемую позицию на экране. Его формат следующий:

```
LOCATE <П>,<С>
```

где <П> - это номер позиции в строке (номер колонки), а <С> - номер строки, где Вы хотите напечатать символ. <П> должна находиться в диапазоне от 0 до значения аргумента последнего оператора WIDTH минус 1. <С> может лежать в диапазоне от 0 до 22 (23, если перед этим был выполнен оператор KEY OFF).

Пример:

```
10 SCREEN 0
20 WIDTH 80
30 LOCATE 36,11
40 PRINT"ПРИВЕТ"
```

Третий аргумент (0 или 1) может быть добавлен в операторе LOCATE. Этот аргумент управляет отображением курсора (1-курсор есть, 0-курсор не отображается). Курсор будет отображен при опускании третьего аргумента, а также при возврате в прямой режим и исполнении оператора INPUT.

Оператор LOCATE очень часто используется вместе с оператором INPUT.

Пример:

```
10  SCREEN 0
20  WIDTH 40
30  LOCATE 2,0
40  INPUT"ВАШ ОТВЕТ (0-10)";R
50  LOCATE 2,5
60  PRINT"ВЫ ВЫИГРАЛИ"
```

### (3) POS и CSRLIN

Для определения позиции курсора используйте функции **POS** и **CSRLIN**: **C=POS** (<фиктивный аргумент>) возвращает номер позиции в строке; аргумент необходим, однако он может быть любым, так как игнорируется. **L=CSRLIN** возвращает номер строки.

Пример:

```
10  CLS
20  PRINT RND(1)
30  IF CSRLIN < 20 THEN 20
40  STOP:GOTO 10
```

Эта программа выводит на экран одну страницу случайных чисел и предотвращает скроллинг (рулонный режим перемещения строк снизу вверх) экрана, который стирает верхние строки одну за другой. Для того, чтобы вывести вторую страницу, нажмите F8 .

### (4) PRINT

Этот оператор необходим для отображения любого текста на экране. Мы использовали его, когда рассказывали о файлах. Простейший формат этого оператора следующий:

**PRINT E1;E2;E3..**





Позиция на экране, где E1,E2,E3... будут напечатаны, может быть определена при помощи оператора **LOCATE**. По умолчанию - это начало строки, следующей за строкой, где был отображен предыдущий текст. Перечень выражений, управляющих выводом на экран, может разделяться символами точки с запятой.

**PRINT E1; E2;**

В этом случае последующий текст будет слит с предыдущим:

```
10  PRINT"ПРИВЕТ";
20  PRINT" ЛЮДИ"
```

Для разделения можно также использовать запятую (вместо точки с запятой). В этом случае данные последовательно выводятся на экран, с разделением на зоны шириной 14 символов без слияния. Определение этих зон будет учитывать ширину экрана, установленную при помощи **WIDTH**, до тех пор, пока Вы не перейдете в режим работы с логическим устройством **"CRT:"**

```
10  WIDTH 40:DEFSNG A:A=ATN(1)
20  OPEN"CRT:"AS #1
30  PRINT#1,A,A,A
```

Тем не менее, из соображений практичности мы будем использовать оператор **WIDTH 80**, если нам понадобится отобразить несколько колонок чисел.

В предложенных примерах Вы могли заметить, что числовые значения снабжаются предшествующим им пустым знакоместом (пробелом). Для того, чтобы обойти это, попробуйте выполнить следующее:

```
10  A$=STR$(125)
20  PRINT"A=";RIGHT$(A$,LEN(A$)-1)
```

Если числовое значение, которое должно быть напечатано - целое число и меньше 10, достаточно ввести

PRINT HEX$(значение)

Для оставления пробелов между двумя полями, используйте функции SPC и TAB. Использование их разрешено только в пределах оператора PRINT. Сравните результаты выполнения следующих операторов:

```
PRINT"AAA";SPC(3);"BBB"
PRINT"AAA";TAB(6);"BBB"
```

Оператор PRINT имеет расширение USING, позволяющее более точно контролировать формат отображения. Это расширение можно использовать как для числовых, так и для символьных аргументов. Однако его никогда нельзя использовать для отображения смеси полей этих двух типов.

(1) Для вывода числовых данных:

| | |
|---|---|
| PRINTUSING"####";2/3<br>PRINTUSING".####";2/3<br>PRINTUSING"#.##";-2/3 | Числовое значение отображается с определенным количеством цифр до и после десятичной запятой. Автоматически происходит округление. Знак % появляется, если все значащие, не цифровые разряды не укладываются в поле формата (знак минус занимает одну из значащих позиций). Перед знаком # возможно написание сообщения (PRINTUSING "РЕЗУЛЬТАТ= #.##";2/3) |





| | |
|---|---|
| PRINTUSING"+##.##";35/3<br>PRINTUSING"+##.##";-35/3<br>PRINTUSING"##.##+";35/3<br>PRINTUSING"##.##+";35/3 | Значение выводится так же, как показано выше, однако либо указывается знак. Если Вы замените знак (+) на знак (-), перед положительным значением будет напечатан пробел. |
| PRINTUSING+"••####";13<br>PRINTUSING"••###";-13 | Пробелы будут заполнены знаком •. |
| PRINTUSING" ###";2000<br>PRINTUSING" ###";-2000<br>PRINTUSING" ###";2000<br>PRINTUSING" ###";-2000 | Так отображается знак перед каждым значением (второй знак указывает цифру, так же как и каждый #). |
| PRINTUSING"•• ##";2 | Знак денежной единицы ( ) отображается перед значением, а пробелы заполняются знаками •. |
| PRINTUSING"#,#######";1200 | Установить запятую для отделения групп по три разряда |
| PRINTUSING"##.##^^^^";23456 | Отобразить значение в представлении для научных расчетов. Количество цифр перед десятичной запятой равно количеству знаков # минус 1. |

(2) Для вывода символьных значений:

| | |
|---|---|
| PRINTUSING"!";"AAAA" | Напечатать только первый символ. |
| A$="Доброе утро &&"<br>PRINTUSING A$;" тов.","Като" | Напечатать определенные строки начиная с позиции, указаной знаками &. Вы можете указать столько строк, сколько символов &, при этом строки разделяются запятыми. Это расширение оказывается особенно удобным для создания стандартных текстов, в которых лишь небольшие части могут изменяться. |
| PRINTUSING"\ \";"AA" | Количество пробелов между двумя знаками \ указывает минимальную длину строки. Если строка короче, она будет удлинена, путем добавления к ней пробелов справа. |

Расширение USING относится ко всему оператору PRINT; это означает, что комбинацию числовых и символьных констант нельзя использовать в одном операторе PRINTUSING. Для вывода данных обоих типов в одной строке используйте следующий прием:





```
PRINTUSING"#.##:";10/3;:PRINT"Рублей"
```

Расширение USING может быть использовано для пересылки данных на активное устройство (например, для создания файла на дискете). Для этого необходимо только указать номер открытого файла:

```
PRINT #1,USING...
```

Это относится и к устройству "CRT:" для вывода на экран.

## 6-3. Режим SCREEN 1

Этот режим отличается от режима SCREEN 0 следующими тремя особенностями:

(1) Длина строки уменьшается до 32 символов, однако графические символы отображаются полностью. В режиме SCREEN 0 нет эквивалента 80-символьному режиму.

(2) Возможна установка цвета границы экрана.

(3) Разрешено использование спрайтов.

### 6-3-1. Выбор цвета

Как и в режиме SCREEN 0, выбор палитр осуществляется при помощи оператора COLOR, который в данном режиме воспринимает три аргумента:

```
COLOR <Т>,<Ф>,<Г>
```

<Г> - это цвет части экрана, лежащей ниже и выше доступной пользователю области. Начально цвета установлены как:

```
COLOR 15,4,7
```

Для возврата к начальным цветам после их переустановки, нажмите [F6].

Используя компьютер серии [MSX]-2, можно переустанавливать палитры (как и в режиме SCREEN 0), используя операторю:

COLOR=(Ц,К,З,С)

### 6-3-2. Отображение текста

Текст отображается на экране в выбранном для него цвете. Тем не менее, как это будет показано далее, возможно отображение групп символов разными цветами. Максимальная длина отображаемой строки - 32 символа.

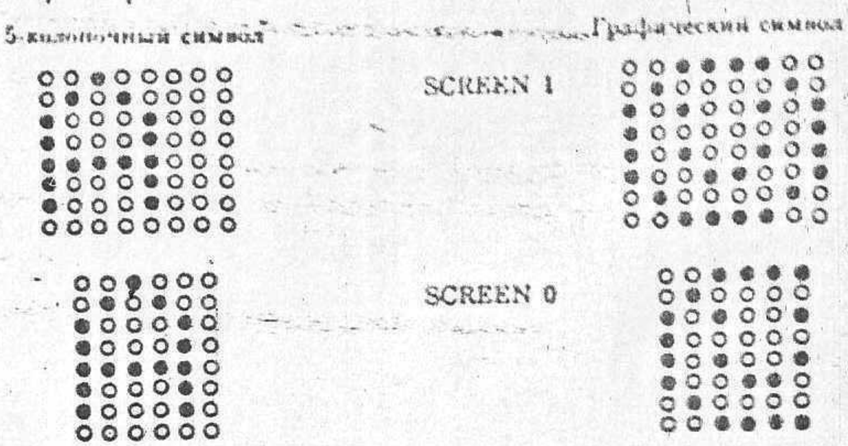
Образ символа представляется матрицей 8x8 точек. В режиме SCREEN 0 последние две колонки матрицы подавляются. Таким образом, 40-символьная строка занимает 40x6=240 пикселов (точек на экране).

В режиме SCREEN 1 все 8 колонок символа отображаются, что соответствует 32x8=256 пикселам в одной строке. Графические символы (вводимые с помощью клавиши [GRAPH]) также имеют эффективную ширину, равную 8 колонкам матрицы знакоместа. Именно поэтому они не отображаются полностью в режиме SCREEN 0.



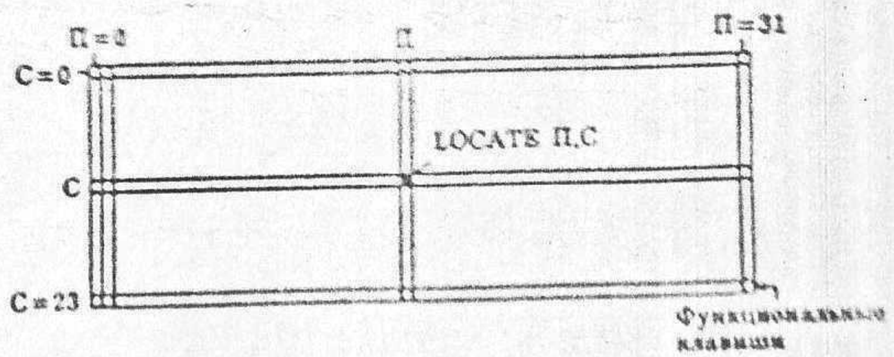


Примеры образов символов:




Все операторы управления отображением текста остаются такими же, как и в режиме SCREEN 0. Однако, координатная система несколько изменяется:

Координатная система для отображения текста в режиме SCREEN 1(WIDTH 32)




Максимальный аргумент для WIDTH равен 32.

### 6-3-3. Спрайты

Спрайты - это графические объекты, образ которых может быть определен пользователем. MSX Бейсик позволяет отображать спрайты в любой точке экрана, а также определять их цвет.

Работа со спрайтами происходит в два этапа:

(1) Определение образа

В наиболее простой форме образ спрайта определяется матрицей 8x8 точек.

Обычно эти образы определяются в программе с помощью рассмотренных ранее операторов DATA для обеспечения наилучшей визуализации. Они запоминаются в определенной области памяти компьютера, определяемой псевдопеременной SPRITE$.

Следующий пример демонстрирует определение двух образов:

SPRITE1

```
10    SCREEN 1
20    FOR I=0 TO 1
30    S$=""
40    FOR J=0 TO 7
50    READ A$
60    S$=S$+CHR$(VAL("&B"+A$))
70    NEXT J
80    SPRITE$(I)=S$
90    NEXT I

1000  DATA 11111111
1010  DATA 10000001
1020  DATA 10000001
1030  DATA 10000001
1040  DATA 10000001
1050  DATA 10000001
1060  DATA 10000001
1070  DATA 11111111
1080
```





SPRITE1

```
1100  DATA 10000000
1110  DATA 11000000
1120  DATA 11100000
1130  DATA 11110000
1140  DATA 11111000
1150  DATA 11111100
1160  DATA 00110000
1170  DATA 00110000
```

Эта программа использует два вложенных цикла. Внешний цикл осуществляет подсчет номера спрайта, служащего в качестве аргумента для функции SPRITE$. Переменная S$ инициализируется в строке 30, после чего начинается внутренний цикл. Этот цикл последовательно читает 8 блоков символьных данных, представляющих собой текстовый эквивалент побитного (двоичного) представления спрайтов (строка 50). Данные присваиваются переменной A$, затем преобразовываются в числовое значение, используя функцию VAL, и, наконец, в символ, соответствующий данному коду (строка 60). В этой же строке этот символ сливается с переменной S$. После того как S$ будет содержать 8 символов, при помощи функции SPRITE$, эта строка запоминается в памяти образов спрайтов (строка 80). Функция или псевдопеременная SPRITE$ может рассматриваться как массив символьных данных, который предварительно был определен (подобно DIM) и содержит по 8 символов на элемент. Этот массив имеет размерность 255 и позволяет Вам определять до 256 образов спрайтов: от SPRITE$(0) до SPRITE$(255).

Обратите внимание на способ записи данных, а также на преобразование их в определяющие образ символы. Единицы соответствуют засвеченным пикселам, тогда как нули указывают на пикселы, остающиеся прозрачными.

Существуют другие способы для определения образа спрайта. Мы могли бы определить SPRITE$(0), используя следующий оператор:

SPRITE$(0)=CHR$(255)+STRING$(6,129)+CHR$(255)

Однако этот способ в ряде случаев оказывается утомительным.

SPRITE$ является псевдопеременной, что подразумевает возможность прочтения ее значения.

Запустите программу, приведенную выше, после чего введите:

? SPRITE$(0)

Вы получите удивительный результат. Чтобы его интерпретировать, давайте попробуем еще один пример; введите:

SPRITE$(2)="AAAAAAAA"

Если Вы запросите значение SPRITE$(2), Вы снова получите AAAAAAAA.

Какому образу это будет соответствовать ? Мы знаем, что код символа A равен 65, что в двоичном представлении равно 01000001. Следовательно, образ будет соответствовать двум вертикальным линиям, занимающим вторую и последнюю колонки матрицы 8x8.

**(2) Отображение спрайта**

Добавьте следующие строки к предыдущей программе:





SPRITE1

```
100 PUT SPRITE 0,(50,10),1,0
110 PUT SPRITE 1,(60,50),7,1
```

Запустите программу. Должны появиться два спрайта, один голубой и один черный.

Оператор

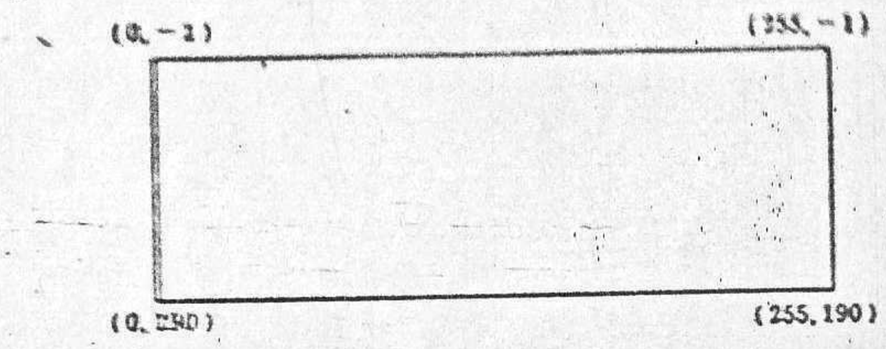
PUT SPRITE <H>,(X,Y), <Ц>,<О>

используется для отображения спрайта. Аргументами данной функции являются:

H-номер отображаемого спрайта: 32 спрайта могут быть отображены на экране одновременно (H изменяется от 0 до 31), однако в одной горизонтальной строке может лежать не более 4 спрайтов, а пикселы остальных спрайтов с более высокими номерами останутся невидимыми.

(X,Y)-координаты верхнего левого угла матрицы 8x8 спрайта, который должен быть отображен; эту систему координат не нужно путать с системой координат при отображении текстов. Координаты спрайта могут лежать в диапазоне от -32768 до 32767, однако при отображении спрайта будут браться координаты, пересчитанные как модуль от деления X и Y на соответствующие размеры экрана.

Координатная система при отображении спрайтов



Ц-цвет отображения спрайта: лежит в диапазоне от 0 до 15 и в точности соответствует таблице номеров цветов, приведенной ранее для текстов. Отображаемый цвет также зависит от переустановки палитр, которые являются особенностью компьютера серии MSX-2.

О-номер образа отображаемого спрайта (от 0 до 255).

Н (номер отображаемого спрайта) и О (номер образа) необходимо различать: один и тот же образ может соответствовать двум различным спрайтам. Более того, образ, соответствующий отображаемому спрайту, может быть в любое время изменен.

Н является необходимым аргументом; остальные могут быть опущены. Значения, используемые при очередных выполнениях **PUT SPRITE** именно для данного спрайта (Н), являются значениями по умолчанию опущенных значений. Проверьте это вводом оператора:

```
PUT SPRITE 0,,15,1
```

Цвет и образ спрайта 0 изменились, но координаты остались теми же самыми.

Когда два спрайта перекрываются, цветные пикселы спрайта с более низким номером скрываются пикселами других спрайтов, чьи номера выше. Это позволяет создавать многоцветные спрайты. Попробуйте, например, наложить спрайты, используя пример, приведенный выше.

Для того, чтобы избавиться от спрайта, определите его цвет как прозрачный (0) или заполните нулями соответствующий образ.





Существует возможность изменить как ФОРМАТ ОБРАЗА, так и МАСШТАБ спрайтов, что выполняется с помощью оператора **SCREEN** (второй аргумент):

**SCREEN 1,N**

| | | |
|---|---|---|
| N = 0 (по умолчанию) | 256 матриц 8x8, | масштаб: x1 |
| N = 1 | 256 матриц 8x8, | масштаб: x2 |
| N = 2 | 64 матрицы 16x16, | масштаб: x1 |
| N = 3 | 64 матрицы 16x16, | масштаб: x2 |

Вы можете проверить эффект использования масштабирования, заменив строку 10 на следующую:

```
10  SCREEN 1,1
```

Тем не менее, для отображения спрайта, имеющего матрицу 16x16, недостаточно только заменить строку на **SCREEN 1,2**. Образ спрайта также необходимо соответственно изменить.
Пример:

SPRITE2

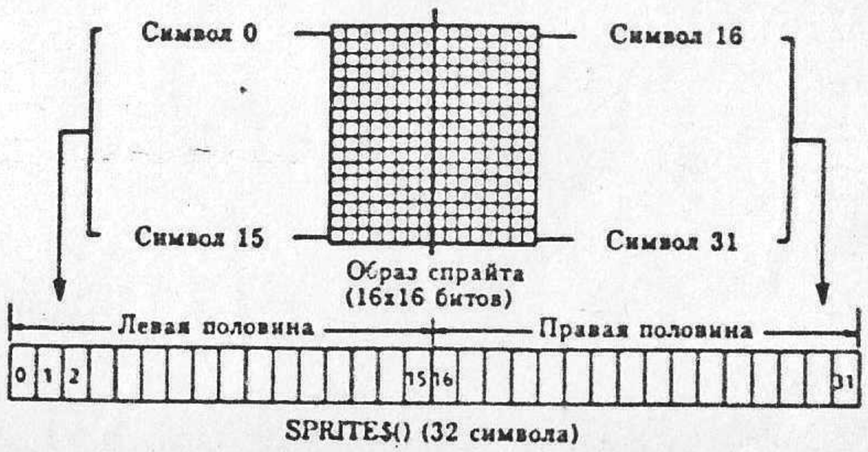
```
10   SCREEN 1,2
20   SL$=""
30   SR$=""
40   FOR I=1 TO 16
50   READ A$
60   AL$=LEFT$(A$,8)
70   AR$=RIGHT$(A$,8)
80   SL$=SL$+CHR$(VAL("&B"+AL$))
90   SR$=SR$+CHR$(VAL("&B"+AR$))
100  NEXT
110  SPRITE$(0)=SL$+SR$
120  PUT SPRITE 0,(50,50),15,0
130  '10SCREEN 1,2
1000 DATA 1000000000000000
1010 DATA 1100000000000000
1020 DATA 1110000000000000
```

```
1030  DATA 1111000000000000
1040  DATA 1111100000000000
1050  DATA 1111110000000000
1060  DATA 1111111000000000
1070  DATA 1111111100000000
1080  DATA 1111111110000000
1090  DATA 1111111111000000
1100  DATA 1111111111100000
1110  DATA 1111111111110000
1120  DATA 0000111100000000
1130  DATA 0000111100000000
1140  DATA 0000111100000000
1150  DATA 0000111100000000
```

Пример:

Данные образа 16x16 запоминаются как 32-х байтная строка: первые 16 байтов определяют левую половину спрайта, а другие 16 байтов правую половину. Если операторы **DATA** должны быть представлены таким образом, чтобы образ был непосредственно виден, Вы должны разделить данные на две части и затем поместить их в соответствующие строки (**SL$** и **SR$**). Следующий рисунок поясняет это.

Как данные загружаются в SPRITE$()








Вы можете изменить масштаб этого спрайта, заменив следующую строку:

```
10  SCREEN 1,3
```

Примечания:

(1) **SCREEN M** (M указывает на формат и масштаб) не очищает экран, однако стирает образы спрайтов в памяти компьютера.

(2) **SCREEN P** (P указывает на режим работы экрана) очищает экран без стирания образов спрайтов из памяти компьютера.

(3) Для перемещения спрайта на экране просто изменять координаты.

Для проверки этого, изменить программу следующим образом:

SPRITE3

```
115  FOR X=0 TO 500
120  PUT SPRITE 0,(X,50),15,0
125  FOR I=0 TO 100:NEXT I
130  NEXT X
```

Вы можете видеть, что X-координата определяется как целое число, равное результату деления X на ширину экрана (256).

Строка 125 служит для замедления движения. Для ускорения замените в этой строке число 100 на 10.

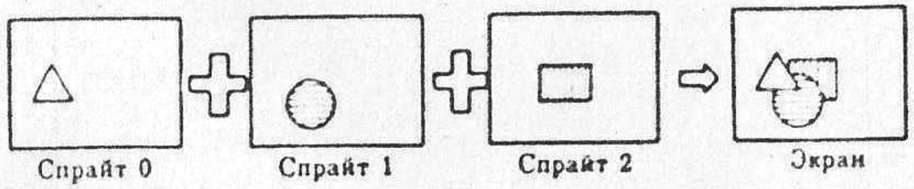
(4) Можно определять координаты расположения спрайта в зависимости от предыдущего положения, используя расширение STEP. Опробуйте это, заменив строки со 115 по 130 на следующие:

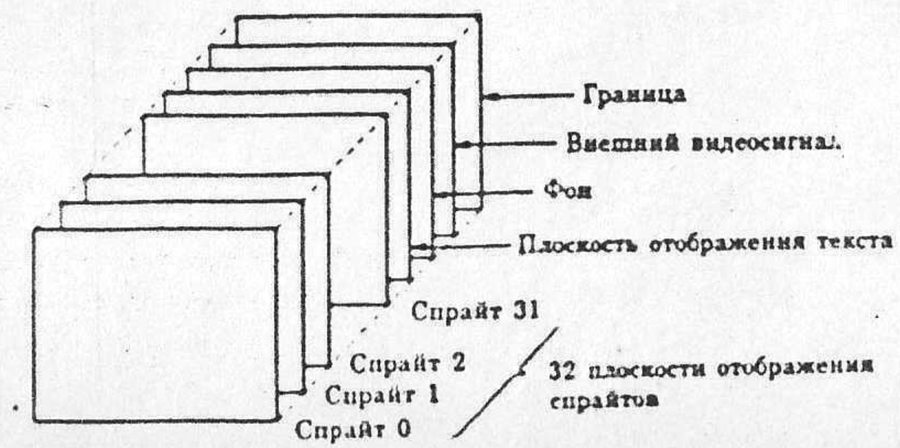
```
115 PUT SPRITE 0,(0,50),15,0
120 FOR I=0 TO 500
125 PUT SPRITE 0,STEP(1,0)
130 FOR J=0 TO 100 : NEXT J
135 NEXT I
```

Строка 115 определяет начальное положение, цвет и образ спрайта 0. Строка 125 добавляет 1 к старой координате X и 0 к координате Y. Цвет или образ могут быть опущены, если их значения не должны изменяться.

(5) Если Вы удалите строку 10, Вы увидите, что при своем движении спрайт ничего не стирает. Он только скрывает все, что находится в данный момент под ним.

Плоскости отображения спрайтов на экране








## 6-4. Режим SCREEN 2

Режим SCREEN 2 отличается от предыдущего режима следующим:

(1) разрешается использование всех графических операторов, а не только операторов работы со спрайтами.

(2) цвет изображения и цвет фона могут быть определены для каждой секции шириной 8 пикселов по горизонтали в дополнение к трем цветам, разрешенным в режиме SCREEN 1.

(3) для изображения текста необходимо указание на устройство "GRP:".

### 6-4-1. Выбор цвета

Цвет текста, фона и границ выбирается так же, как и в режиме SCREEN 1 - при помощи оператора:

COLOR <Т>,<Ф>,<Г>

Тем не менее, существуют важные различия:

(1) Во время выполнения вышеуказанного оператора изменяется только цвет границы экрана, а не цвет изображения или фона.

(2) Цвет фона изменяется только в том случае, если после выполнения оператора COLOR будет выполнен оператор CLS.

(3) Новый цвет текста будет изображен для той части текста и графических операторов, которые были введены после выполнения операторы COLOR (см. ниже).

Для проверки этого, сравните следующие программы:

```
10  SCREEN 2:COLOR 1,15,6
20  GOTO 20

10  SCREEN 2:COLOR 1,15,6:CLS
20  GOTO 20

10  COLOR 1,15,6:SCREEN 2
20  GOTO 20
```

Запустите каждую из этих программ после предварительного нажатия F6. В первом случае, изменяется только цвет границы экрана. Вторая и третья программа изменяют также цвет фона (оператор SCREEN выполняет операцию, аналогичную CLS).

Строка 20 удерживает режим экрана. Для возврата в прямой режим, выполните CTRL + STOP.

Далее мы будем рассматривать более подробно возможности выбора цвета.

Для изменения палитры (при использовании компьютера серии MSX-2) выполните такие же операции, как и в режимах SCREEN 0 и SCREEN 1.

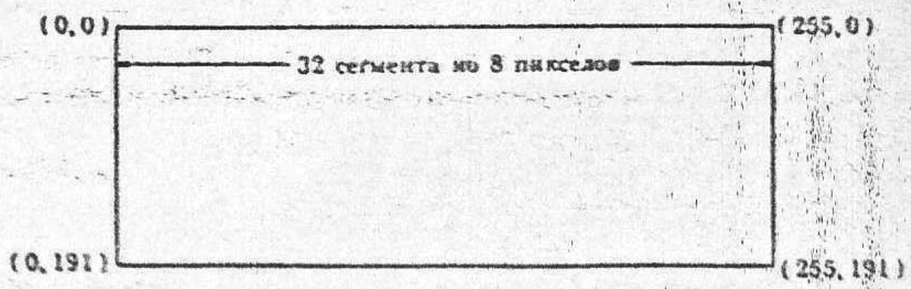
### 6-4-2. Графические операторы

В режиме SCREEN 2 можно использовать большое количество графических операторов, позволяющих изображать графические образы на экране. Среди них можно выделить оператор DRAW, имеющий большое количество расширений: они составляют специальный графический макро-язык.
Перед тем, как мы приступим к рассмотрению этих операторов, необходимо изучить детально координатную систему и способ, при помощи которого MSX Бейсик присваивает цвет экранному пикселу.



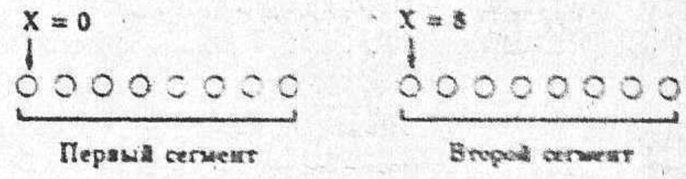


Координатная система в режиме SCREEN 2.




Каждая из 192 линий экрана поделена на 32 8-ми пиксельных сегмента. Каждому сегменту присваивается цвет изображения и цвет фона. Отображаемый пиксел появляется на экране в присвоенном цвете, тогда как отсутствие пиксела (не изображение) означает появление на его месте цвета фона. Следовательно, для 8-ми пиксельного сегмента возможно наличие всего двух цветов. Это поясняется следующей иллюстрацией:

Сегментное деление строки




Графические операторы, устанавливающие цвет фона и цвет изображения сегментов, определяют, какие пикселы будут "высвечены", а какие нет.

Список графических операторов:

**(1) PSET и PRESET**

Присвоить пикселу цвет фона или цвет изображения.

PSET (X,Y),<Ц>

Оператор присваивает пикселу, имеющему координаты (X,Y), цвет Ц. Если другие пикселы, принадлежащие этому же самому сегменту были высвечены, им также присваивается цвет изображения Ц. Если цвет не был явно указан, цвет изображения будет таким, каким он был при выполнении последнего оператора **COLOR**, даже если за этим не последовал оператор **CLS**.

**PRESET (X,Y)**

Присвоить пикселу с координатами (X,Y) цвет фона сегмента.

Если цвет указан так же, как и в случае использования оператора **PSET**, оператор **PRESET** имеет такой же результат, как и **PSET**.

Расширение **STEP** может быть указано перед координатами, для определения значения приращения координат по отношению к последнему выполненному графическому оператору.

**PSET STEP (X,Y)**

Координаты могут быть опущены. В этом случае цвет будет таким же, как и цвет последнего определенного пиксела.

Чаще всего эти операторы используются для рисования графиков.

Пример: нарисовать синусоиду и косинусоиду





SINE

```
10  SCREEN 2
20  PI=4*ATN(1)
30  FOR I=0 TO 200
40  X=I+20
50  YS=60*SIN(I*PI/50)
60  PSET (X,90-YS),1
70  YC=60*COS(I*PI/50)
80  PSET(X,90-YS),15
90  NEXT
100 GOTO 100
```

Обратите внимание на смену цветов в местах пересечения кривых (во избежание этого выберите одинаковые цвета для двух кривых).

Операторы **PSET** и **PRESET** используются также для позиционирования курсора перед выводом на экран (см. ниже).

(2) **LINE**

Этот оператор рисует линии и замкнутые прямоугольники.

Для рисования линии:

**LINE** (X1,Y1)-(X2,Y2),<Ц>

Этот оператор нарисует линию от точки с координатами (X1,Y1) до точки с координатами (X2,Y2). Цвет линии указывается <Ц>. Если цвет не определен, цвет изображения будет выбран таким же, каким он был во время выполнения последнего оператора **COLOR**.

Расширение STEP может быть указано перед координатами (X2,Y2).

```
LINE (X1,Y1)-STEP(5,6)
```

эквивалентно

```
LINE (X1,Y1)-(X1+5,Y1+6)
```

Координаты (X1,Y1) могут быть опущены:

```
LINE -(X2,Y2)
```

Этот оператор нарисует линию от точки, указанной в последнем графическом операторе до точки с координатами (X2,Y2).

```
LINE -STEP(5,6)
```

Этот оператор рисует линию от точки, указанной в последнем графическом операторе, до точки, расположенной на 5 пикселов правее и 6 пикселов ниже.

Пример:

```
10  SCREEN 2
20  LINE (16,20)-STEP(31,40),1
30  LINE-STEP(50,-22),15
40  GOTO 40
```

Для вывода на экран замкнутого прямоугольника:

```
LINE(X1, Y1)-(X2, Y2), <Ц>, B
```

Результатом выполнения этого оператора будет прямоугольник, чья диагональ была бы нарисована, если бы не было указано расширение оператора B (буква латиницы).





Пример:

Введите расширение B в строки 20 и 30 программы, приведенной выше.
Для вывода на экран раскрашенного прямоугольника введите:

```
LINE(X1,Y1)-(X2,Y2),Ц,BF
```

Примечания, относящиеся к пропуску <Ц> и использованию расширения STEP, остаются в силе и для случая использования расширений BF и B.

Пример:

Введите расширение BF для строк 20 и 30 в программу, приведенную выше.

Оператор LINE с расширением B или без него изображает все указанные пикселы и присваивает цвет фона (указанный или по умолчанию) сегментам, содержащим эти пикселы. С этой точки зрения оператор LINE может рассматриваться как серия операторов PSET. Вы можете проверить это с помощью следующей программы:

```
10  SCREEN 2
15  '
20  LINE (8,10)-STEP(7,8),15,B
30  LINE (16,10)-STEP(7,8),1,B
35  '
40  LINE (8,20)-STEP(3,8),15,B
50  LINE (12,20)-STEP(11,8),1,B
55  '
60  LINE (8,30)-STEP(11,8),15,B
70  LINE (20,30)-STEP(3,8),1,B
75  '
80  GOTO 80
```

В первом случае оба прямоугольника относятся к различным сегментам. Во втором и третьем случаях один из прямоугольников расширяется до сегментов, принадлежащих другому прямоугольнику. Цвет фона переустанавливается вторым оператором LINE для сегментов, участвующих в данном изображении.

Это совсем не означает, что невозможно создать многоцветные рисунки в режиме SCREEN 2. Однако, расположение элементов рисунка должно быть тщательно выбрано, в зависимости от изображаемых сегментов.

Оператор LINE с расширением BF выполняется несколько иначе:

Если это возможно, пикселы, задействованные при выполнении этого оператора, гасятся, указанный цвет становится цветом фона для сегмента, тогда как цвет изображения сегмента устанавливается равным 0 или остается неизменным.

При возникновении конфликтов между цветом и конфигурацией, расширение BF выполняется как и PSET: оно изображает все задействованные пикселы, переустанавливает цвет изображения сегмента, тогда как цвет фона остается неизменным.

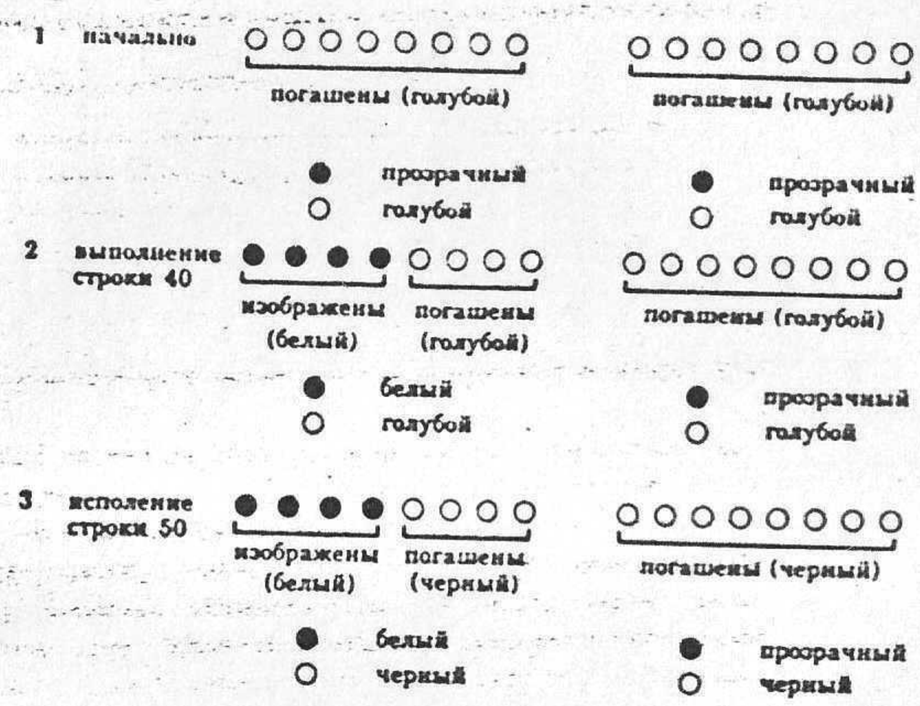
Для проверки этого вернитесь к программе, приведенной выше и замените расширения B на BF.





В первом случае, оба замкнутых прямоугольника задействуют разные сегменты.
Все их пикселы изображаются цветом фона, а цвет изображения устанавливается равным 0. В результате первый прямоугольник белый, а второй - черный.


Во втором случае меньший прямоугольник рисуется первым. Так как только левая половина сегмента задействована, только 4 пиксела оказываются изображенными. Цвет изображения устанавливается белым, а цвет фона остается нетронутым. Затем рисуется черный прямоугольник. Он расширяется на 2 сегмента. В первом сегменте задействовано только 4 пиксела,они гасятся: цвет фона устанавливается черным. Белый цвет изображения остается неизменным. Что касается второго сегмента, все происходит точно так же, как и в первом случае.

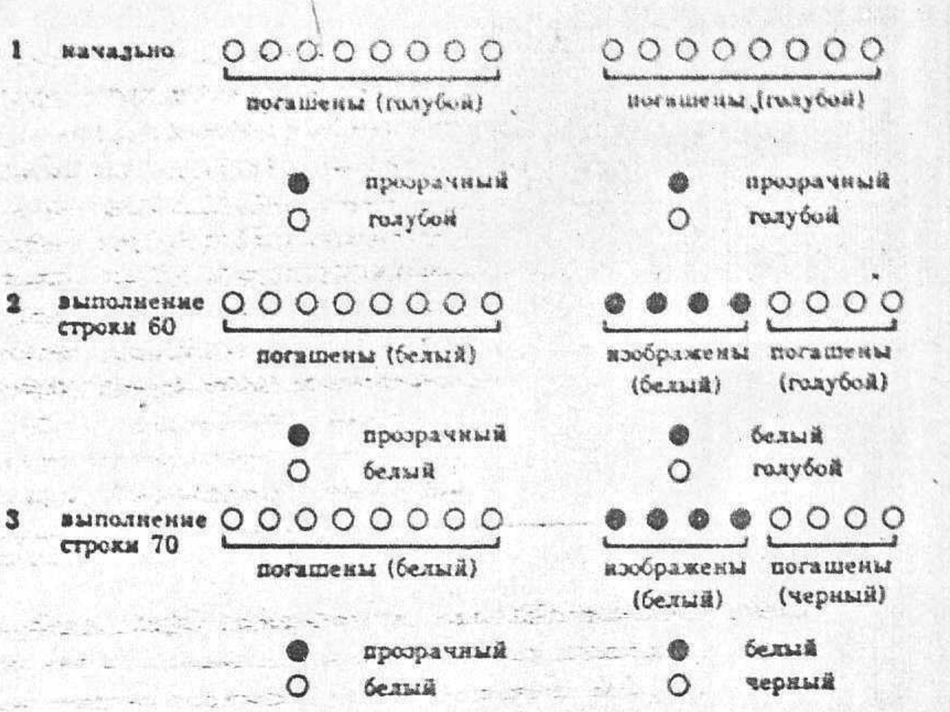



В третьем случае белый прямоугольник рисуется первым. Этот прямоугольник принадлежит двум сегментам. Для первого сегмента все происходит так же, как и в первом случае. Для второго сегмента только первые 4 пиксела оказываются изображенными. Цвет фона остается голубым, а цвет изображения становится белым.

Затем рисуется черный прямоугольник. При этом задействуется вторая половина второго сегмента. Последние четыре пиксела остаются погашенными, а первые четыре - изображаются. Цвет фона становится черным, а цвет изображения - белым.








Таким образом, расширение BF обычно генерирует более четкий рисунок, чем расширение B.

Хотя это несколько более длинная процедура, замкнутые прямоугольники указанного цвета получить намного легче.

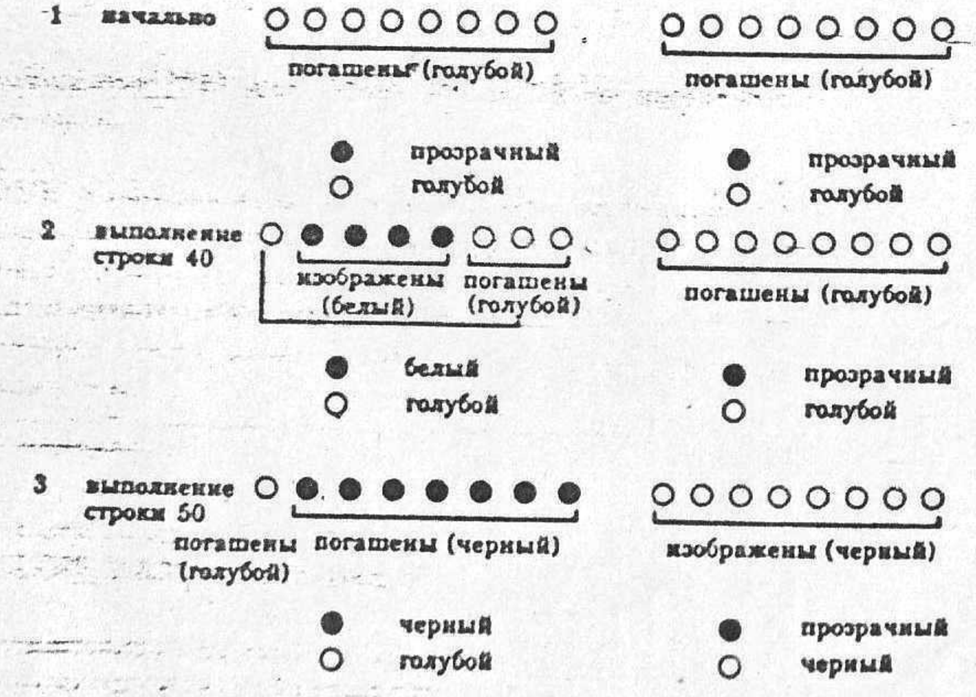
В некоторых случаях, тем не менее, смешение цветов, при использовании расширения BF, невозможно обойти. Это можно проиллюстрировать, заменив строку 40 программы, приведенной выше, на строку:

```
40 LINE (9, 10) - STEP(3, 8), 15, BF
```

Никаких белых прямоугольников на экране не появится. Причина этому следующая:

Белый прямоугольник использует для изображения пикселы 2,3,4 и 5 первого сегмента. Цвет фона остается голубым, тогда как цвет отображения устанавливается белым. Затем мы пробуем нарисовать черный прямоугольник, использующий последние три пиксела первого сегмента. Эти точки должны быть изображены. В противном случае, цвет фона стал бы черным и первые три погашенных пиксела сегмента (задействуемые в случае использования оператора LINE BF) были бы изображены черным. Таким образом, последние три пиксела сегмента изображаются, при этом цвет изображения становится черным.

Цвет фона остается голубым. Как только изображаются пикселы 2,3,4 и 5, они становятся черными и белый прямоугольник полностью исчезает.








(4) CIRCLE

Оператор CIRCLE рисует круги и эллипсы:

CIRCLE (X,Y),<P>,<Ц>,<Н>,<К>,<А>

Координаты (X,Y) определяют начальную точку. Координатам может предшествовать расширение STEP.

<P> указывает на длину радиуса (в случае, если фигура - круг).

<Ц> определяет цвет изображения. По умолчанию используется цвет, определенный в последнем операторе COLOR.

<Н> и <К> определяют начальный и конечный угол, указываются в радианах и вычисляются в соответствии с тригонометрическими правилами. По умолчанию <Н> равен 0, а <К> - 2•PI.

<А> - коэффициент отношения между горизонтальной и вертикальной осями. По умолчанию он принимается равным 1 (круг).

Пример:

```
10  SCREEN 2
20  CIRCLE (50, 60), 100
30  CIRCLE STEP (-10, 30), 40, , , , 2
40  GOTO 40
```

Какую роль играет параметр <P> в случае рисования эллипса? Правило достаточно простое: значение P всегда присваивается более длинной оси.

Примечание:

Пропорции между координатами X и Y не обязательно сохраняются при изображении на экране. Настройка отклоняющей системы мониторов может оказывать определенное влияние. Для получения действительного круга иногда может быть необходимо несколько изменять параметр A (устанавливать его, например, равным 1.2).

Для того, чтобы отобразить дугу или эллипс, используются параметры <H> и <K>. Эти значения могут лежать в диапазоне от 0 до 2•PI. Для соединения крайней точки дуги с центром используйте отрицательное значение для <H> и <K> (−0 является запрещенным значением; используйте, например, −0.001).

ОБРАТИТЕ ВНИМАНИЕ:

Дуги вычисляются так же, как и в тригонометрии, от крайней правой точки по часовой стрелке. Они должны быть указаны в радианах (см. Приложение B).

Оператор CIRCLE не позволяет ориентировать эллипс под любым углом по отношению к горизонтальной оси. Если Вы хотите отобразить такую фигуру, используйте метод, показанный в примере ниже:





ELLIPS

```
10  SCREEN 2
20  PI = 4•ATN(1)
30  FOR I = 0 TO 200
40  X = 120 + 70•SIN(I•PI/100 + PI/4)
50  Y = 90 − 50•COS(I•PI/100)
60  PSET (X, Y)
70  NEXT
80  GOTO 80
```

Сдвиг PI/4 (45 градусов) в строке 40 определяет угол между осями и горизонталью. Обратите внимание, что коэффициенты 50 и 70 в строках 40 и 50 не оказывают влияния на абсолютное значение осей, однако влияют на высоту и ширину окна, в котором изображается эллипс. Так как вычисление тригонометрических функций занимает несколько больше времени, этот метод менее эффективен, чем использование оператора CIRCLE для изображения неориентированного эллипса.

### (4) PAINT

Этот оператор закрашивает определенную часть экрана.

PAINT (X,Y),<Ц>

Координаты X и Y указывают точку, с которой начинается закраска. Эта точка должна находиться внутри области, которая должна быть закрашена. Расширение STEP может использоваться вместо указания координат.

<Ц> указывает на цвет. Он должен быть таким же, как и цвет границ (в режиме SCREEN 3 он может отличаться). Граница должна быть замкнута. Оставление малейшего зазора в контуре приведет к полной закраске всего экрана.

Пример:

```
10  SCREEN 2
20  CIRCLE (120,90),50,7,,,2
30  PAINT STEP (0,0),7
40  GOTO 40
```

Цвет для оператора PAINT определяется так же, как и для оператора LINE BF.

Если ваша область имеет замысловатую форму, очень трудно расположить границу таким образом, чтобы оператор PAINT закрасил каждый пиксел используемых сегментов. Когда несколько операторов PAINT используются для закраски смежных областей, зачастую невозможно получить чистый рисунок.

(5) POINT

Эта функция возвращает цвет указанной точки.

C = POINT(X,Y)

Если точка выходит за границы экрана, возвращается значение −1.

Если точка изображена, возвращается ее цвет. В противном случае возвращается цвет фона.

Если точка расположена внутри экранного окна, Вы всегда получите значение цвета.





Примечание:

Эта функция игнорирует спрайты. Если точка изображается в белом цвете потому, что скрыта белым спрайтом, возвращаемый цвет соответствует действительному цвету точки, когда спрайт находится где-нибудь еще.

(6) DRAW

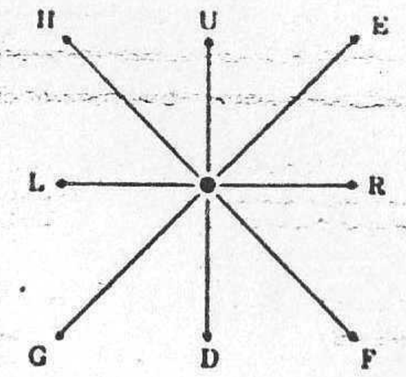
Параметрами оператора DRAW является текст на так называемом "Графическом Макро-Языке" (GML) MSX Бейсика. Эти параметры содержат строки символов, которые интерпретируются как графические операторы.

Линии указанной длины и ориентации, расположенные под углами, кратными 45 градусам с привязкой к осям координат, могут быть выведены на экран с помощью этого оператора.
Пример:

```
10  SCREEN 2
20  PRESET (120,90)
30  DRAW "C15 R20 U30 C1 L12"
40  GOTO 40
```

Пробелы могут быть опущены. C,R,U и L являются примерами операторов GML; числовые значения являются параметрами.

C определяет цвет и используется со всеми последующими операторами.
Другие символы приведены на следующем рисунке:




**Оператор M позволяет вам нарисовать линию между точкой, указанной последним графическим оператором и точкой, чьи координаты указаны в операторе:**

**DRAW"C14 M20,30"** рисует линию между точкой, указанной предыдущим графическим оператором и точкой с координатами (20,30).

**Если координатам предшествует знак + или -, значения с учетом знака добавляются к координатам последней указанной точки.**

**Все операторы, указанные выше, могут быть модифицированны с использованием префиксов B или N.**

| | |
|---|---|
| **B** | **двигает точку, но не рисует** |
| **N** | **рисует и возвращается к стартовой точке** |

**Существует два дополнительных оператора:**

**A (вращение): этот параметр может иметь значения, лежащие в диапазоне от 0 до 3.**





| | |
|---|---|
| 0 → | нормальное |
| 1 → | 90 градусов |
| 2 → | 180 градусов |
| 3 → | 270 градусов |

**S (масштаб): параметр должен быть в диапазоне от 0 до 255.**

| | |
|---|---|
| 0 → | нет изменения масштаба |
| больше 0 → | параметр, деленный на 4, служит множителем для всех размеров (таким образом, 4 умножает на 1). |

Оба оператора [illegible] со всеми последующими операторами (до тех пор, пока не будет выполнена переустановка).

Во все операторах, рассмотренных до этого момента, параметры могут быть заданы выражениями, переменными или массивами переменных. В этом случае, элементы массива могут быть переменными, но не выражениями. Формат применется следующий:

Оператор=выражение;

Пример:

```
DRAW"C15 R=I+10*J;"
```

Точка с запятой необходима после каждого выражения, служащего в качестве параметра. Оператор M требует точки с запятой после обеих координат.

```
M+=X;,-=Y;
```

Наконец, существует оператор, позволяющая создавать подпрограммы:

X <имя символьной переменной>;

Содержимое переменной читается и интерпретируется как строка Графического Макро-Языка.

Ниже приведен пример, суммирующий все, о чем мы говорили:

ДОМ

```
10  SCREEN 2
20  A$="R20 U20 L20 D20"'ФАСАД
30  B$="E14 F14 BL20"'КРЫША
40  C$="R5 U10 L5 D10"'ДВЕРЬ
50  D$="R4 U4 LA D4"'ОКНО
60  E$="XA$;BR4 XC$;BM+10,-5 XD$; BM-18,-12
        XB$;"'ДОМ
70  DRAW"BM20,20C1S2XE$;BM100,100S6C15XE$;"'
    МАЛЕНЬКИЙ ЧЕРНЫЙ ДОМ ВВЕРХУ СЛЕВА И
    БОЛЬШОЙ БЕЛЫЙ ДОМ В ЦЕНТРЕ
80  GOTO 80
```

Примечание:

Оператор DRAW изображает точки в цвете так же, как и инструкция PSET.

### 6-4-3. Спрайты

Все происходит так же, как и в режиме SCREEN 1, за исключением того, что изображение спрайтов исчезает, когда Вы возвращаетесь в прямой режим.





Существует два существенных отличия между координатными системами спрайтов и графических операторов:

(1) Все координаты могут быть в диапазоне от −32768 до +32767 (за исключением оператора PAINT). В случае спрайтов, берется целая часть результата деления указанного значения на размер экранного окна; для графических операторов "за пределами" экранного окна ничего не будет нарисовано.

(2) Координата Y спрайта сдвинута на одну позицию, что сказывается на графических операторах: для изображения спрайта, левый верхний угол которого будет иметь координаты такие же, как и пиксел, изображенный с использованием PSET(X,Y), необходимо использовать PUT SPRITE H,(X,Y-1),Ц,О.

### 6-4-4. Отображение текста

Для изображения текста в режиме SCREEN 2 используйте устройство
"GRP:".

В этом случае, следующий оператор должен быть использован в начале программы:

OPEN"GRP:" AS #N

где N указывает на присвоенный номер "файла"

Если Вы одновременно собираетесь использовать ввод/вывод на дискету (например, PRINT# или INPUT#), не забудьте соответственно определить MAXFILES.

Определив однажды устройство "GRP:", Вы можете изображать текст, используя оператор:

PRINT#N,"текст"

где N- номер файла.

Тем не менее, позиционировать курсор при помощи оператора LOCATE Вы больше не можете. Точнее, в режиме SCREEN 2 вообще нет курсора текста.

Курсор существует только при использовании графических операторов.

Перед изображением текста позиционируйте курсор, используя операторы PSET или PRESET.

Пример:

```
10  SCREEN 2
20  OPEN"GRP:"AS#1
30  PRESET(20,12)
40  PRINT#1,"ABCDE"
50  GOTO 50
```

Текст будет напечатан на позиции, указанной оператором PRESET (верхний левый угол матрицы 8x8 символа "A" расположен на (20,12)).
Другие символы автоматически сдвигаются на 8 пикселов вправо, как и в режиме SCREEN 1.

Для более компактной записи попробуйте следующее:

```
10  SCREEN 2: OPEN"GRP:" AS #1
20  A$="ABCDEF"
30  FOR I=1 TO LEN(A$)
40  PRESET (12+(I-1)*6, 20)
50  PRINT#1, MID$(A$, I, 1)
60  NEXT
70  GOTO 70
```





В режиме SCREEN 2 изображение текста происходит так же, как и при работе графических операторов. Таким образом, Вы можете рассматривать этот процесс как изучение образа символа перед тем, как вывести его на экран, и выполнения операторов PSET, когда необходимо отобразить пикселы этого образа. Соответственно:

(1) Цвет изображения символа зависит от последнего выполненного оператора COLOR, так что цвет каждого символа может быть установлен отдельно; однако опасайтесь смешения цветов, если более одного символа находится внутри одного сегмента.

(2) Оператор PRINT# никогда ничего не удаляет. Отображение пустых пробелов вызывает только передвижение курсора без стирания чего-либо. Для удаления текста используйте LINE BF, чтобы установить цвет области экрана равный цвету фона и погасить все пикселы. Цвет изображения также может быть переустановлен в значение локального цвета фона, после чего тот же самый текст может быть повторно выведен на экран на том же самом месте.

(3) Оператор WIDTH не берется в расчет.

### 6-5. Режим SCREEN 3

Этот режим характеризуется низкой разрешающей способностью (блоки, эквивалентные 4 x 4 пиксела в SCREEN 2).

#### 6-5-1. Выбор цвета

Цвета текста, фона и границы выбираются и работают таким же образом, как и в SCREEN 2: новый цвет фона появляется после CLS, и новый цвет текста не влияет на уже напечатанный.

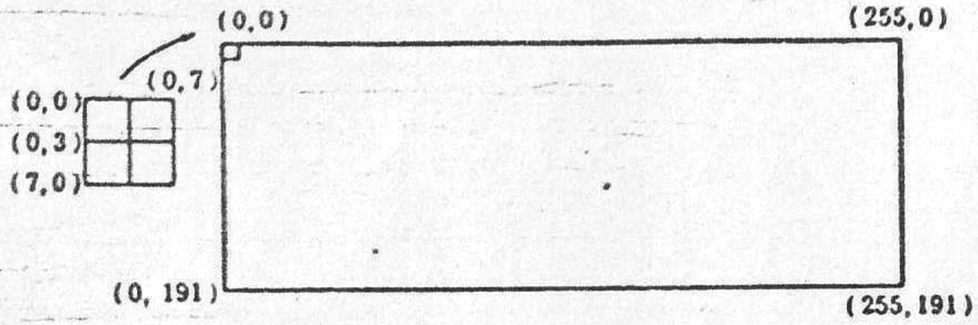
### 6-5-2. Спрайты

Спрайты функционируют, как в **SCREEN 2**, и в той же системе координат. Их разрешение не зависит от разрешения других операторов.

### 6-5-3. Графические операторы

Координатная система та же, что и в **SCREEN 2**, но имеется только 64 x 48 блока. Каждый из этих блоков имеет только один цвет (тот, что отображен). Можно считать, что графические операторы явно переопределяют цвет блоков.

Система координат в SCREEN 3




Поскольку координатная система та же, что и в **SCREEN 2** и **3**, графические операторы, описанные для **SCREEN 2**, работают также и в **SCREEN 3** и выполняются в основном тем же образом.

Пример:

```
10  SCREEN 3
20  LINE(8,0)-(128,128)
30  GOTO 30
```





Для сравнения, выполните эту программу сначала в **SCREEN 3**, а затем в **SCREEN 2**. За исключением разрешения, линия имеет одну и ту же длину в обоих случаях. Другими словами, хотя пиксели больше, длина линии не изменяется.

В режиме **SCREEN 3** **PSET (8,0),15** эквивалентен **PSET (X,Y),15**, если (X,Y) принадлежат одному блоку 4 x 4.

Поскольку каждый блок имеет единственный цвет, искажения цветов между смежными областями не происходит.

Сравните изображения, выводимые этой программой в **SCREEN 2** и **3**:

CIRCLES

```
10  SCREEN 3
20  DEFINT A - Z : R = 1
30  CIRCLE(82,90), R, RND(1) * 16
40  CIRCLE(164,90), R, RND(1) * 16
50  R = R + 4
60  GOTO 30
```

Оператор **PAINT** может заполнить область отличным от границы цветом, что невозможно в SCREEN 2.

```
10  SCREEN 3
20  CIRCLE(128,90), 50, 6
30  PAINT STEP(0,0), 15, 6
40  GOTO 40
```

В **SCREEN 2** весь экран постепенно захватывается цветом границы, если он отличается от цвета, использованного для заполнения области.

### 6-5-4. Текстовое изображение

Текстовое изображение осуществляется посредством "GRP:". Для установки курсора перед выводом на экран, используйте PRESET или любой другой оператор, определяющий позицию графического курсора. Верхний левый угол матрицы отображаемого символа совпадет с блоком, где помещен графический курсор.

Символ отобразится на матрице 8 x 8, т.е. в масштабе, который в 4 раза больше, чем в SCREEN 2.

Пример:

```
10  SCREEN 3 : OPEN"GRP:" AS#1
20  PRESET(50,70)
30  PRINT #1, "ABC"
40  GOTO 40
```

## 6-6. Режим SCREEN 4 (имеется только на компьютерах MSX-2)

Этот режим идентичен SCREEN 2 в том, что касается выбора цвета, графических операторов и отображения текста. Он отличается, однако, обработкой спрайтов: он предлагает новые возможности относительно цвета, и на одной линии может находиться 8 спрайтов.

### (1) COLOR SPRITE$

Этот оператор дает возможность изображать многоцветный спрайт (один цвет на линию). В SCREEN 2 для получения такого же результата должны накладываться несколько спрайтов. Этот оператор имеет следующий формат:





COLOR SPRITE$(NS) = строка максимум 16 символов где NS - номер отображаемого спрайта

Каждый строковый символ соответствует одной линии спранта. Поэтому следует использовать 8 символов, когда образы спрайтов кодируются в 8 байтах (SCREEN 4,0 и SCREEN 4,1) и 16 символов при кодировании в 32 байтах (SCREEN 4,2 и SCREEN 4,3).

Этот оператор рассматривает двоичную структуру, соответствующую коду ASCII каждого символа, и действует на спрайтовой линии в соответствии со значением битов, составляющих эту структуру:

**• БИТ 7 (крайний левый)**

Если этот бит равен 1, спрайтовая линия отображается на 32 точки левее координаты X, заданной в PUT SPRITE.

**• БИТ 6**

Если этот бит 1, пересечение этого спрайта с другим не будет обнаружено. Правило приоритета (когда два спрайта накладываются, спрайт с меньшим номером перекрывает другой) более не соблюдается. Вместо этого, пересечение имеет другой цвет, определяемый результатом оператора OR, примененного к наложившимся цветам.
Линии, не затронутые пересечением, исчезают.

**• БИТ 5**

Если этот бит 1, пересечение не обнаруживается.

**• БИТ 4**

Этот бит не используется.

• БИТЫ 3-0

Четыре младших бита позволяют закодировать номер палитры (0-15), присвоенный этой линии.

Пример 1:

```
10  SCREEN 4,1
20  SPRITE$(0) = STRING$(8,255)
30  SC$ = CHR$(&B10000001)
40  PUT SPRITE 0, (32, -1), 15, 0
50  IF INKEY$ = "" THEN 50
60  COLOR SPRITE$(0) = SC$
70  GOTO 70
```

Спрайт изображается белым. При нажатии клавиши, его первая линия отображается черным на 32 точки левее.

Пример 2: измените предшествующую программу

```
30  SC$ = STRING$(8,&B01000111)
40  PUT SPRITE 0, (32, -1), 8, 0 : PUT
    SPRITE 1, (32, -1), 7, 0
60  COLOR SPRITE$(1) = SC$
```

Спрайт 0 изображается красным (8) и спрайт 1 - темно-синим (7). При нажатии клавиши, все линии спрайта 1, находящиеся под спрайтом 0, сохраняются; остальные исчезают. Цвет пересечения белый (7 OR 8 = 15). Обратите внимание, что COLOR SPRITE$ в строке 60 должен быть применен к спрайту плоскости 1.

### (2) COLOR SPRITE

Этот оператор очень похож на первый, но действует на всех линиях:





COLOR SPRITE (NS) = значение

Значение должно лежать в пределах от 0 до 127. Это делает невозможным передвижение спрайта на 32 точки влево, поскольку для этого требуются значения от 128 до 255.
Когда значение находится между 0 и 15, такой же эффект достигается следующими операторами:

PUT SRITE 0, , значение

COLOR SPRITE$(0) = STRING$(8,значение)

COLOR SPRITE(0) = значение

Когда значение находится между 0 и 127, два последних оператора эквивалентны.

Когда значение варьирует от 128 до 255, или когда необходимо воздействовать на отдельные линии, следует использовать второй оператор.

## 6-7. Режим SCREEN 5 (имеется только на компьютерах MSX-2)

Этот режим предлагает те же возможности, что и SCREEN 4, в том, что касается спрайтов. Он отличается от предыдущего в трех пунктах:

(1) Разрешение 256x212

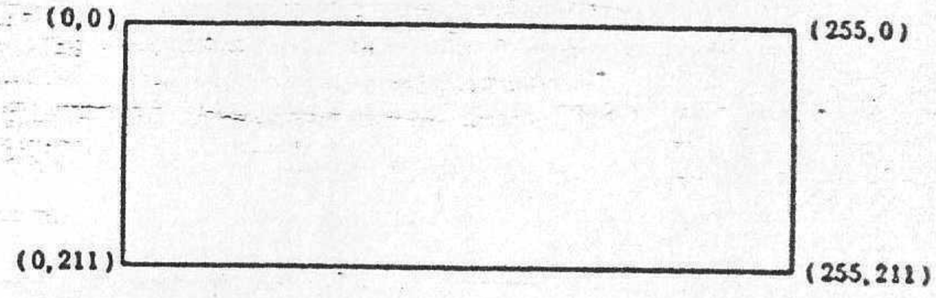
(2) Выбор цвета

(3) Несколько экранных страниц

### 6-7-1. Разрешение и выбор цвета

Экран разделен на 212 строк по 256 точек. Каждая группа из 2 точек в строке может рассматриваться в качестве сегмента. Это довольно близко к SCREEN 3, где имеется 48 строк из 32 сегментов, составленных из двух больших блоков 4x4. Здесь эти " блоки" сведены в один пиксель.

Система координат в SCREEN 5




Цвет присваивается каждой точке. Следовательно, сегменту присваивается два цвета.

Палитры определяются как в SCREEN 0. Для каждой точки можно выбрать любой цвет, содержащийся в 16 палитрах.

### 6-7-2. Экранные страницы

В SCREEN 5 в Вашем распоряжении имеется несколько экранных страниц.
Их количество зависит от объема видеопамяти.

| | | |
|---|---|---|
| 64 | килобайта → | 2 страницы (0 и 1) |
| 128 | килобайтов → | 4 страницы (от 0 до 3) |





### (1) SET PAGE

Посредством оператора SET PAGE можно определять страницу для отображения на экране и страницу для воздействия (графическими операторами и т.д.).

Пример:

SETPAGE

```
10  SCREEN 5
20  OPEN"GRP:" : AS #1
30  SET PAGE 0, 0 : CLS
40  PSET(100, 100) : PRINT #1," ABCD"
50  SET PAGE 1, 1 : CLS
60  FOR I = 0 TO 7 : FOR J = 0 TO 127
70  PSET(2*J, I + 8), I + 2 : PSET STEP(1,
    0), I + 3
80  NEXT J, I : P = 1
90  IF INKEY$ ="" THEN 90
100 P = P XOR 1
110 SET PAGE P, P
120 GOTO 90
```

Первый аргумент SET PAGE указывает номер отображаемой страницы, а второй - номер активной страницы. В строке 30 страница 0 отображается и активизирована (это ситуация по умолчанию). Затем небольшой текст записывается на страницу 0.

В строке 50, страница 1 отображена и активизирована: нарисованы 8 линий. Программа ожидает нажатия клавиши в строке 90. Если клавиша нажата, номер (P) отображенной страницы изменяется оператором XOR (если это 0, он становится 1, и наоборот). Все изображение может быть изменено простым нажатием клавиши. Это составляет один из самых мощных операторов MSX Бейсика версии 2.0. Ниже следует еще один весьма мощный оператор:

**(2) COPY**

Оператор COPY пересылает блок экрана в другое место экрана (на этой же или другой странице).
Фактически, имеется 6 вариантов этого оператора:
• Вариант 1

COPY(X1,Y1)-(X2,Y2), страница-источник TO (X3,Y3),

страница-приемник, логическая операция

Первая группа параметров указывает позицию копируемого блока и работает как в операторе LINE. Эти данные обязательны.

Второй параметр относится к номеру страницы-источника (по умолчанию - активная страница).

Третий параметр необходим и указывает место, где должно быть выполнено копирование.

Номер страницы-приёмника необязателен (по умолчанию - страница-источник).

Следующий параметр - ключевое слово для определения комбинации цвета экрана с цветом точек копируемого блока посредством логической операции.





Используемые слова перечислены ниже:

| | |
|---|---|
| XOR | SC = (NOT C) AND SC OR (NOT SC) AND C |
| OR | SC = SC OR C |
| AND | SC = SC AND C |
| PSET | SC = C (по умолчанию) |
| PRESET | SC = NOT C |
| SC | = цвет экрана (до и после копирования) |
| C | = цвет копируемого рисунка |

Этим словам может предшествовать префикс T. Тогда прозрачный цвет не будет действовать.

Пример:

COPY

```
10  SCREEN 5
20  SET PAGE 0, 0
30  LINE(100, 0)-(130, 30), 7, BF
40  LINE(100, 40)-(130, 70), 6, BF
50  IF INKEY$= "" THEN 50
60  SET PAGE 1, 1 : CLS
70  COPY(0,0)-(255,100), 0 TO (100,0), 1, PSET
80  GOTO 80
```

Испытайте логические операции, изменяя последний параметр в строке 70.
• Вариант 2

COPY(X1,Y1)-(X2,Y2) страница-источник TO переменная массива/спецификация файла

Это дает возможность копировать блок изображения в цифровой форме либо в в переменную заданного размера или в указанный файл. Размер массива определяется следующей формулой:

INT((4•(ABS(X1-X2) + 1)•(ABS(Y1-Y2) + 1) + 7)/8) + 4

• Вариант 3

**COPY переменная массива/спецификация файла, направление TO (X3,Y3), страница-приемник, логическая операция**

Параметр направления может быть пропущен, но он даст возможность "перевернуть" блок изображения. Он может принимать следующие значения:

| | | |
|---|---|---|
| 0 | верхний/левый | →нижний/правый |
| 1 | верхний/правый | →нижний/левый |
| 2 | нижний/левый | →верхний/правый |
| 3 | нижний/правый | →верхний/левый |

• Вариант 4

**COPY массив переменных TO спецификация файла**

• Вариант 5

**COPY спецификация файла TO массив переменных**

• Вариант 6

ОБРАТИТЕ ВНИМАНИЕ:

Этот оператор применяется на компьютерах MSX-2, оснащенных видеоинтерфейсом.

**COPY SCREEN режим**

Это позволяет преобразовать изображение, полученное с внешнего источника видеосигнала, в цифровые данные, записанные на экранной странице.





Параметр <режим> может принимать два значения:

0: (по умолчанию) поле записывается на отображенной экранной странице
1: 2 поля (кадр) записываются на странице, номер которой равен номеру отображенной минус 1. В этом случае номер отображенной страницы должен быть нечетным.

Предполагается, что экран подготовлен к работе с изображением, приходящим от внешнего источника видеосигнала. Для этой цели имеется оператор SET VIDEO.

**SET VIDEO режим,ym,cb,синх,аудио,видеовход,команда AV**

Режим может принимать следующие значения:

0: (по умолчанию) изображение от компьютера
2: смешивание
3: ТВ-сигнал

ym может иметь следующие значения:

0: интенсивность ТВ-сигнала уменьшена наполовину
1: нормальная интенсивность

cb может иметь следующие значения:

0: вывод цветных данных
1: ввод цветных данных

синх может иметь следующие значения:

0: внутренняя синхронизация
1: внешняя синхронизация (невозможна, когда режим = 0)

аудио может иметь следующие значения:

```
0: только звук от компьютера
1: смешивание с внешним сигналом (правый
   канал)
2: смешивание с внешним сигналом (левый
   канал)
3: смешивание по обоим каналам
```

видео может иметь следующие значения:

```
0: выбирает разъем RGB
1: выбирает разъем RCA
```

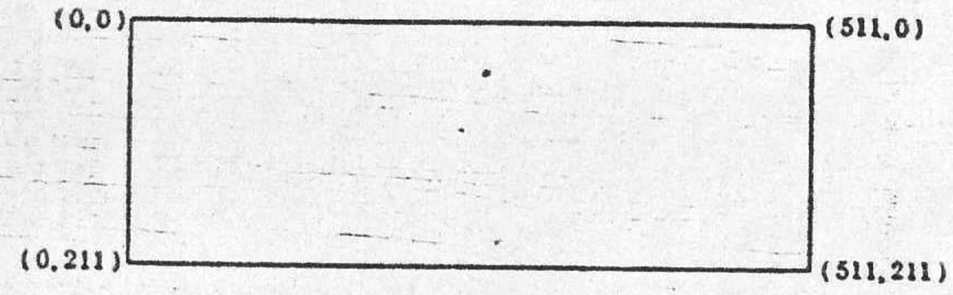
команда AV может иметь следующие значения:

```
0: выхода через разъем RGB нет
1: выход разрешен
```

## 6-8. Режим SCREEN 6 (имеется только на компьютерах MSX-2)

### 6-8-1. Разрешение и выбор цвета

Экран состоит из 212 строк по 512 точек, но поскольку экранная память не больше, чем в SCREEN 5, может использоваться только 4 цвета.








Сегмент состоит из четырех пикселов. Ширина сегмента та же, что и в SCREEN 5. Поскольку цвета этих точек по-прежнему кодируются в одном байте, для каждой точки имеется только два бита.
Именно поэтому возможен выбор только четырех различных цветов.
Оператор COLOR должен быть выполнен после SCREEN 6.

CLS устанавливает цвет экрана на 0 (прозрачный: появляется только цвет границы). Для цвета границы можно выбирать номер палитры, лежащий в пределах от 0 до 31, хотя имеется только 16 палитр.

От 0 до 15:

Цвета создается значением двух младших битов номера палитры, введенных оператором COLOR (15 = 00001111 возвращает 3); цвет границы однообразный.

От 16 до 31:

Принимаются в расчет четыре младших бита цвета границы. Первые два определяют цвет точек с четными координатами, а два последних - точек с нечетными координатами (22 = 00000110 возвратит 01 и 10 в двоичном виде; появятся черные и зеленые вертикальные линии).

### 6-8-2. Графические операторы.

Все графические операторы работают, как и в SCREEN 3 (они переопределяют цвет точек, но в новой системе координат); цвет границы может быть задан оператором PAINT.

```
10  SCREEN 6:COLOR 15,17,1
20  CIRCLE(255, 106), 100, 3
30  PAINT STEP(0, 0), 2, 3
40  GOTO 30
```

Получается окружность! Радиус по вертикали (автоматически) сжимается для соответствия разнице масштаба между абсциссой и ординатой.

### 6-8-3. Вывод текста

Во-первых, должно быть активизировано устройство "GRP:". Символы изображаются как в SCREEN 0, WIDTH 80 (сжатие), но писать можно в любом месте, передвигая курсор с помощью PSET. Нулевые биты образа символа сбрасывают соответствующую точку экрана на 0.

### 6-8-4. Применение экранных страниц и видео

Все действует как в SCREEN 5. Но не забывайте следующее: формула, дающая размер массива переменных, копирующего блок экрана, теперь выглядит:

```
INT((2*(ABS(X1-X2) + 1)*(ABS(Y1-Y2) + 1) + 7)/2) + 4
```

## 6-9. Режим SCREEN 7 (имеется только на компьютерах MSX-2 с видео памятью 128 Кбайт)

### 6-9-1. Разрешение и выбор цвета

Система координат та же, что и в SCREEN 6, но для каждой точки экрана имеется в два раза меньше видеопамяти. Первоначально, цвет экрана устанавливается на цвет фона оператора COLOR.

Этот режим наиболее удобен для многоцветных рисунков.





### 6-9-2. Вывод текста

Для вывода текста необходимо активизировать "GRP:". Символы появляются, как в режиме SCREEN 0, WIDTH 80, но пробелы ничего не стирают.

### 6-9-3. Применение экранных страниц

Здесь имеются только страницы 0 и 1, поскольку одна страница занимает огромное пространство в памяти.

Формула, необходимая для вычисления размера массива, принимающего данные блока экрана, та же, что и в SCREEN 5.

## 6-10. Режим SCREEN 8 (имеется только на компьютерах MSX-2 с видео памятью 128 Кбайт)

Этот режим характеризуется количеством цветов, которые могут выводиться одновременно.

### 6-10-1. Разрешение и выбор цвета

Имеется 212 строк по 256 точек, как в SCREEN 5. Цвет каждой точки кодируется в одном байте, и следовательно, имеет значения, лежащие в пределах от 0 до 255.

Они кодируются следующим образом:

```
байт цвета = GGGRRRBB
```

Первые три бита (G) представляют значения от 0 до 7 и определяют интенсивность зеленого цвета. Следующие три бита (R) устанавливают интенсивность красного, и последние два (B) - интенсивность синего; они представляют значения от 0 до 3.

Цвета текста, фона и границы определяются как в SCREEN 2 посредством COLOR (или графических операторов для цвета текста), но значения лежат в пределах от 0 до 255.

Введенное значение и пропорции цвета соотносятся следующим образом:

цвет = 32•G + 4•R + B

В SCREEN 8 палитры не используются: желаемые оттенки задаются непосредственно.

Пример:

```
10  SCREEN 8
20  FOR I = 0 TO 255
30  COLOR, I : CLS
40  FOR J = 0 TO 300
50  NEXT J, I
```

### 6-10-2. Применение экранных страниц

Имеется только две экранные страницы, как в SCREEN 7.

Формула, необходимая для вычисления размера массива, принимающего копию блока экрана, имеет следующий вид:

INT((8*(ABS(X1-X2) + 1)*(ABS(Y1-Y2) + 1) + 7/8)



# Обработка прерываний в программе пользователя

Глава 7

Прерывания представляют собой особый тип условных переходов. Для выполнения "обычного" условного перехода, где-либо в программе располагается оператор, проверяющий условие и передающий управление на указанный номер строки в соответствии с результатами проверки. Переход происходит только в том случае, если при выполнении программы в последовательности операторов был обнаружен оператор перехода.

Что же касается прерываний, то компьютер непрерывно проверяет условия их возникновения в то время, как программа пользователя может выполнять "полезную работу".

Имеется два типа обработки прерываний:
(1) Обработка по событию
(2) Обработка по ошибке

### 7-1. Обработка прерывания по событию

Обработка событий выполняется с помощью операторов двух типов:

(1) ON <событие> GOSUB <номер строки>

Этот оператор определяет событие, вызывающее проверку, а также место в программе, куда передается управление при наступлении события.

(2) <событие> ON/OFF/STOP

Если событие включено (ON), происходит переход, определенный оператором (1). Этот оператор, разумеется, должен быть предварительно выполнен.

Если событие выключено (OFF) (по умолчанию), переход не происходит.

Если событие остановлено (STOP), переход при наступлении события не происходит, но информация о том, что оно произошло, записывается, и переход происходит при обнаружении в программе оператора <событие> ON.
Отметим, что операторы (1) или (2) не выполняют "сами по себе" переход. Когда выполнен оператор типа (1), относящийся к данному событию, компьютер начинает проверять, не произошло ли данное событие во время выполнения программы. Если событие произошло, текущий оператор все равно выполняется до конца. Затем компьютер проверяет, выполнен ли оператор типа (2). Возможны четыре варианта:

(1) Оператор "<событие> ON/OFF/STOP" не был выполнен. Программа выполняется, как если бы событие не произошло.

(2) Оператор "<событие> ON" был выполнен и является последним оператором этого типа, относящимся к данному событию.

Происходит переход к подпрограмме, указанной в операторе "ON <событие> GOSUB". При входе в эту подпрограмму неявно обрабатывается "<событие> STOP". Когда выполняется оператор RETURN подпрограммы, автоматически происходит "<событие> ON", и выполнение возвращается к оператору, следующему за тем, при выполнении которого было обнаружено событие.

(3) Был выполнен оператор "<событие> STOP", и он является последним оператором обработки данного события.
программа выполняется дальше как будто не прото не произошло никакое событие.

Это событие отмечается и записывается, и выполнение продолжается нормально. Если в дальнейшем встречен оператор





"<событие> ON", относящийся к тому же событию, происходит переход, и все выполняется далее как в (2).
Ниже перечислены события, прерывания по которым могут обрабатываться в MSX Бейсике.

### 7-1-1. Нажатие 'CTRL' + STOP

Как известно, программа может быть прервана посредством нажатия 'CTRL' + 'STOP'. Это событие может быть отменено следующим оператором:

ON STOP GOSUB <номер строки>

Пример:

```
10   ON STOP GOSUB 100
20   STOP ON
30   GOTO 30
40
100  RETURN
```

Строка 10 определяет ожидаемое событие и номер строки начала подпрограммы обработки прерывания. Строка 20 устанавливает это событие ON. Строка 30 является бесконечным циклом. Она, разумеется, может быть заменена любой последовательностью операторов. Строка 100 - начало подпрограммы обработки события. Здесь следует разместить все операторы, связанные с обработкой прерывания. "Пустая" подпрограмма завершается оператором RETURN.

Как реагирует программа? После нажатия клавиши [F10] (RUN) выполнение программы "доходит" до строки 30. Поскольку строка 10 к этому моменту уже выполнена, компьютер проверяет событие [CTRL] + [STOP]. Как только включенное событие обнаружено, компьютер заканчивает выполнение строки 30 ("бесконечный" цикл). Затем выясняется, выполнялся ли оператор типа STOP ON/OFF/STOP. Поскольку

**STOP ON** был выполнен в строке 20, выполнение переходит на строку 100.

Происходит неявное событие **STOP STOP**, а затем выполняется подпрограмма. Эта подпрограмма содержит только **RETURN**, поэтому выполняется неявный оператор **STOP ON**, и управление вновь переходит на строку 30 (она содержит оператор, который выполняется после перехвата события указанного типа). И мы оказываемся в бесконечном цикле.

Все произошло так, как если бы событие [CTRL] + [STOP] было отключено. Для остановки программы следует выключить компьютер. Способ прерывания такой программы с клавиатуры будет обсужден позже.

Еще один пример:

```
10  ON STOP GOSUB 100
20  STOP ON
30  PRINT N
40  N = N + 1
50  GOTO 30
60  '
100 IF N > 500 THEN END
110 PRINT"[CTRL] + [STOP]"
120 RETURN
```

В данном случае [CTRL] + [STOP] остановит программу, если N > 500. При меньших значениях N выводится своеобразное "эхо".

Обратите внимание, что обрабатывается только комбинация [CTRL] + [STOP]. Нажатие одной клавиши [STOP] просто "задержит" программу, как обычно. Эта клавиша в MSX Бейсике - "перехватываться" не может. Однако можно ее "отключить", как это будет показано ниже.





Обработка [CTRL] + [STOP] позволит избежать любого вмешательства в программу, выполняющуюся в Ваше отсутствие.

### 7-1-2. Прерывания в фиксированные интервалы времени

Псевдопеременная **TIME** делает возможным контроль выполнения программы в определенные интервалы времени. Этот счетчик получает приращение 50 раз в секунду и автоматически сбрасывается на 0 при достижении значения 65535 (два байта).

Пример:

```
10  SCREEN 0
20  TIME = 0
30  IF TIME < 100 THEN 30
40  F = (F + 1) MOD 16
50  COLOR, F
60  GOTO 20
```

Эта программа меняет цвет экрана каждые две секунды. Строка 20 сбрасывает счетчик **TIME** на 0. Строка 30 является "закрытым" циклом, который заканчивается, когда счетчик достигает 100 или больше. В этом случае управление переходит на строку 40, которая переопределяет номер цвета; строка 50 меняет цвет экрана. Затем мы возвращаемся на строку 20, где счетчик сбрасывается на 0. Этот метод, однако, вызывает много хлопот, если Вы хотите, чтобы программа делала что-либо еще в то время, пока она не занимается изменением цвета. Действительно, это "что-либо" должно занимать меньше времени, чем указанный период, и нужно постоянно возвращаться к проверке счетчика.

Обработка события по интервалу помогает решить эту проблему. Оператор обработки события по интервалу имеет следующий формат:

ON INTERVAL = <длина> GOSUB <номер строки>

Длина может лежать в пределах от 1 до 65535 (21 минута 50,7 секунд).
Пример:

```
10  SCREEN 0
20  ON INTERVAL = 100 GOSUB 100
30  INTERVAL ON
40  GOTO 40
50  '
100 F = (F + 1) MOD 16
110 COLOR, F
120 RETURN
```

Эта программа выполняется точно так же, как и предыдущая, но теперь строка 40 легко может быть заменена целой последовательностью операторов, поскольку проверка интервала реализуется автоматически.

Обработка интервала оказывается очень удобной в игровых программах, где она, например, позволяет "бросать бомбу" через регулярные интервалы.

Примечание:

Убедитесь в том, что интервал не короче, чем время выполнения подпрограммы обработки, иначе она будет бесконечно вызывать саму себя, поскольку в конце ее неявно выполняется INTERVAL ON.

### 7-1-3. Столкновения спрайтов

Прерывания по столкновению спрайтов широко используются в игровых программах. Они дают возможность, например, воспроизводить звук взрыва при всяком столкновении ракеты и мишени, если оба этих объекта являются спрайтами.





Пример:

```
10  SCREEN 4
20  SPRITE$(0) = STRING$(8,255)
30  PUT SPRITE 0, (200,100), 15, 0
40  ON SPRITE GOSUB 100
50  SPRITE ON
60  FOR X = 0 TO 240
70  PUT SPRITE 1, (X,100), 1, 0
80  NEXT
90  GOTO 60
100 SCREEN 0
110 RETURN 10
```

Эта программа вызывает "взрыв" на экране, когда спрайт 1 (черный) встречается со спрайтом 0 (белый).

Примечание:

Столкновение обнаруживается, когда накладываются непрозрачные части двух спрайтов. Имеются два способа сделать некоторые части спрайта "прозрачными":

(1) Нулевые биты образа спрайта всегда прозрачны.
(2) Биты со значением 1 образа также прозрачны, если цвет спрайта установлен в 0 операторами PUT SPRITE, COLOR SPRITE$ или COLOR SPRITE, даже если палитра 0 была переопределена посредством COLOR = (0,...). Обнаруживаемые столкновения вызываются совпадением битов 1, за исключением тех случаев, когда цвет каждого спрайта установлен в 0, или если обнаружение совпадения было отменено с помощью COLOR SPRITE$ или COLOR SPRITE.

### 7-1-4. Функциональные клавиши

Следующий оператор обрабатывает события, связанные с нажатием функциональных клавиш:

**ON KEY GOSUB N1, N2,..., N10**

Он напоминает оператор ON <переменная> GOSUB N1, N2,... В этом операторе может быть записано максимум 10 номеров строк. Нажатие F1 отсылает программу на строку N1, нажатие F2 - на строку N2 и так далее. Если будут использоваться только некоторые функциональные клавиши, просто пропустите номера строк.

Пример:

**ON KEY GOSUB 100,, 200**

Здесь программа отсылается на строку 100, если нажата клавиша F1, и на строку 200, если нажата клавиша F2. Остальные клавиши не вызывают подобного эффекта "перехвата".
Определенные ниже операторы определяют для каждой отдельной клавиши, будет она обрабатываться или нет:

**KEY (N) ON/OFF/STOP**

N относится к номеру клавиши. Не путайте это с оператором KEY ON/OFF (без скобок), который влияет на отображение текстов функциональных клавиш в режимах SCREEN 0 и 1.

### 7-1-5. Триггеры

Клавиша пробела является для многих программ триггером ("спусковым крючком"). Другие периферийные устройства, такие, как джойстик или манипулятор МЫШЬ, также имеют по одному или по два триггера. Таким образом, всего имеется до 5





триггеров. Оператор, определяющий прерывание по триггеру, имеет синтаксис, похожий на синтаксис оператора, определяющего прерывания по функциональным клавишам:

**ON STRIG GOSUB N1,N2,...N5**

Определенные ниже операторы определяют условия обработки события, связываемого с каждым из триггеров:

**STRIG (N) ON/OFF/STOP**

где N - номер требуемого триггера.

| N | Триггер |
|---|---|
| 0 | Клавиша пробела на клавиатуре |
| 1 | Триггер 1 на джойстике 1 |
| 2 | Триггер 1 на джойстике 2 |
| 3 | Триггер 2 на джойстике 1 |
| 4 | Триггер 2 на джойстике 2 |

### 7-1-6. Особенности одновременной обработки событий различных типов

Обработка нескольких событий различных типов в одной программе вполне допускается. Однако для того, чтобы структура программы оставалась ясной, необходимо следовать некоторым рекомендациям:

(1) Перед входом в подпрограмму обработки события всегда неявно происходит "<событие> STOP". Если в процессе выполнения этой подпрограммы происходит это же событие, то подпрограмма по завершении будет сразу же вызвана снова. Чтобы избежать этого, помещайте в начале подпрограммы оператор <событие> STOP или OFF; таким

образом Вы предотвратите неявное возникновение ситуации "<событие> ON".

(2) Применение операторов <событие> ON/OFF/STOP помогает отметить те участки программы, в которых разрешены либо запрещены прерывания. Подпрограммы обработки событий, как правило, "маскируются" от прерываний (хотя формально это не обязательно).

(3) Если событие остановлено (STOP), то факт возникновения этого события запоминается. Если же событие того же типа возникает еще раз, то компьютер запоминает только последнее событие. Поэтому особенно важно четко отмечать участки программы, связываемые с событиями каждого типа.

## 7-2. Обработка ошибок

Обработка ошибок предотвращает остановку программы и появление сообщения об ошибке, когда происходит ошибка ожидаемого типа. При обработке ошибок в программе выполнение переходит на подпрограмму обработки, в которой определена реакция на ошибку. Несмотря на сходство этих ситуаций с обработкой событий, имеются некоторые различия в синтаксисе соответствующих операторов.

Оператор обработки прерывания по ошибке выглядит следующим образом:

ON ERROR GOTO <номер строки>

Такого оператора, как "<событие> ON/OFF/STOP", при обработке ошибок не существует. Для отмены действия оператора ON ERROR достаточно записать в нем 0 вместо номера строки. Если номер строки отличен от 0, управление при возникновении ошибки передается на эту строку. Выполнение продолжается начиная с этой строки до обнаружения оператора RESUME.





Этот оператор отмечает конец подпрограммы обработки ошибки и возобновляет выполнение основной программы с того места, где оно было прервано. В операторе RESUME может быть указан номер строки.

Следующие примеры показывают, как это происходит.

Пример 1:

```
10   ON ERROR GOTO 100
20   PRINT 1/SIN(RND(1)) * 60
30   GOTO 20
40   '
100  PRINT"Число слишком велико"
110  RESUME
```

Деление на 0 должно вызывать ошибку. Фактически в этом случае деление на 0 весьма маловероятно, поскольку RND(1) должно было бы возвращать 0. Помните, однако, что компьютер выполняет округление до 0, когда числа слишком малы, и тем самым возможно "случайное" деление на 0, из-за того, что числа, представимые в компьютере, должны находиться в заданных пределах (См. Приложение Е). Обработка прерываний по ошибке предотвращает остановку программы в подобных случаях.

Разумеется, этот пример намеренно упрощен. В более сложных программах происходят самые разнообразные ошибки, что выявляется чаще всего при отладке программ.

Пример 2:

```
10  ON ERROR GOTO 100
20  'что угодно
30  GOTO 20
40  '
100 RESUME
```

В этом примере строка 20 представляет тело программы, т.е. может быть заменена любой последовательностью операций. Здесь очевидно, что никакая ошибка не вызовет появления сообщения о ней. Отладка такой программы становится тем самым невозможной.

Для обработки только ожидаемых ошибок, в то время как остальные ошибки будут вызывать обычные сообщения, используются два приема:

(1) Области программы, где ожидаются ошибки, должны быть четко определены. Если ожидаются ошибки между строками 110 и 130, введите в программу следующие операторы:

```
100 ON ERROR GOTO 1000
140 ON ERROR GOTO 0
```

В этом случае ошибка вызовет переход к строке 1000 (подпрограмма обработки) только тогда, когда она произойдет в ограниченной операторами ON ERROR области.

(2) В подпрограмме обработки осуществляется проверка типа и места возникновения ошибки с помощью функций ERR и ERL. Например, если должно обрабатываться только деление на 0, в начале подпрограммы обработки записывается следующий оператор (см. Приложение Б):





```
IF ERR <> 11 THEN END
```

(3) RESUME возвращает управление оператору, вызвавшему переход. Поэтому если в подпрограмме обработки ошибки никакие переменные не изменены, выполнение этого оператора будет снова и снова вызывать одну и ту же ошибку. RESUME NEXT и RESUME <номер строки> позволяют избежать такого бесконечного цикла. Первый вариант RESUME возвращает управление оператору, следующему за оператором, вызвавшим ошибку; второй вариант позволяет передать управление на любой номер строки.

# Ввод с клавиатуры и с помощью манипуляторов

Глава 8

Операции ввода с клавиатуры были представлены в Главе 1. Прерывания по 'CTRL + STOP', функциональным клавишам и клавише пробела, упомянутые в предыдущей главе, также относятся к операциям ввода с клавиатуры. Триггерные прерывания 1 - 4 соответствуют вводу, осуществляемому с помощью манипуляторов (джойстик, МЫШЬ). В данной глава возможности ввода данных рассмотрены более подробно.

## 8-1. Операции ввода с клавиатуры

### 8-1-1. INKEY$

**INKEY$** является функцией, возвращающей строку из одного символа. С помощью INKEY$ можно прочитать только те клавиши, которым соответствует код какого-либо символа. Ниже приведен перечень клавиш, при нажатии которых ввод с помощью этой функции невозможен:

| | | | |
|---|---|---|---|
| CTRL | SHIFT | CAPS | GRAPH |
| CODE/РУС | STOP | | |

Коды остальных клавиш перечислены в Приложениях А и 3. Некоторые из этих кодов не соответствуют выводимым на экран символам, однако вводятся с помощью INKEY$. Соответствующие этой ситуации клавиши перечислены ниже:

| | | | |
|---|---|---|---|
| ESC | TAB | RETURN | SELECT |
| HOME | CLS | INS | DEL |
| Клавиши курсора | | CTRL + буква | |

Для обнаружения нажатия какой-либо из этих клавиш, используйте следующий оператор:

```
IF INKEY$ = CHR$(K) THEN...
```

где K является кодом ожидаемой клавиши. Если клавиша соответствует выводимому на экран символу, пишется просто:

```
IF INKEY$ = "символ" THEN...
```

Символ записывается в том же виде, как он выглядит на экране, но в кавычках. Этот способ избавляет от необходимости обращаться к кодовой таблице.

При выполнении программы на MSX Бейсике клавиатура постоянно сканируется, и коды записываются в области памяти, называемой БУФЕРОМ клавиатуры. Функция INKEY$ извлекает из буфера код первого символа. Если буфер пуст, эта функция возвращает пустую строку.

Функциональные клавиши имеют ту особенность, что нажатие любой из них вызывает ввод нескольких кодов символов, составляющих ее текст (который можно переопределить), в буфер клавиатуры.

Например:

```
10  IF INKEY$ ="" THEN 10
20  N = N + 1 : PRINT N : GOTO 10
```

Запустите эту программу клавишей [F5]. Коды символов текста этой клавиши - "RUN" + CHR$(13) - автоматически извлекаются из буфера в момент использования их по назначению. Затем программа входит в цикл строки 10 до тех пор, пока не будет нажата какая-либо клавиша. Если это - функциональная





клавиша, то поочередно на экран будет выведено несколько значений переменной N, поскольку эти клавиши вводит в буфер последовательности символов.
Если нажатая клавиша должна проверяться несколько раз, значение INKEY$ присваивается символьной переменной.

```
10  K$ = INKEY$
20  IF K$ = "A" THEN PRINT "A" ELSE IF K$
    = CHR$(13) THEN PRINT "RETURN"
30  GOTO 10
```

### 8-1-2. INPUT$

Функция INPUT$ ожидает нажатия определенного числа символьных клавиш и возвращает введенную таким образом строку.

Пример:

```
10  A$ = INPUT$(4)
20  PRINT A$ : GOTO 10
```

Функция INPUT$(4) в данном примере извлекает 4 символа из буфера клавиатуры и присваивает их переменной A$. Если введено меньше символов, чем задано, функция будет ожидать дальнейшего ввода. INPUT$(1) возвращает один символ, как и INKEY$, но если буфер пуст, эта функция будет находиться в состоянии ожидания, тогда как INKEY$ возвращает пустую строку. Два следующих оператора кажутся эквивалентными:

```
A$=INKEY$: IF A$ = "" THEN 10
A$=INPUT$(1)
```

Однако между ними есть два важных различия:

(1)Нажатие клавиши STOP задерживает программу в цикле INKEY$, тогда как во время ожидания оператором INPUT$ нажатия клавиши клавиша STOP временно отключается. Кстати, это единственный способ временно отключить эту клавишу в MSX Бейсике.

(2) INPUT$ всегда переходит в SCREEN 0 или 1 и выводит подсказку - вопросительный знак, что ограничивает его использование в других режимах SCREEN.

Ниже приведена короткая программа, позволяющая найти код ASCII тех клавиш, для которых он существует.

```
10  A$ = INPUT$(1)
20  PRINT ASC(A$)
30  GOTO 10
```

### 8-1-3. INPUT

Оператор INPUT нам уже встречался. Помните, что он используется для ввода данных из последовательных файлов.

INPUT"сообщение";A$,B$,C

Этот оператор переводит программу в режим SCREEN 0 или 1, поэтому использование его невозможно, если необходимо держать экран в другом режиме. При выполнении оператора выводится <сообщение> (необязательное), заканчивающееся вопросительным знаком. Затем могут вводиться данные (записанные по тем же правилам, что и в строке DATA), разделяемые запятыми и заканчивающиеся нажатием клавиши RETURN. Если символьные данные содержат запятые, то они должны вводиться в кавычках. Если нажать RETURN без данных, указанные в операторе переменные не будут изменены.

Если тип данных не совпадает с типом переменной, появляется сообщение "Redo from start" (Повторить сначала), и ввод





данных следует начать заново.

Обратите внимание:

Ситуация "Redo from start" (Повторить сначала) относится к ошибкам, прерывание по которым не обрабатывается. Если вводится недостаточное количество данных, выводится два вопросительных знака, и компьютер ожидает продолжения ввода. Если вводится слишком много данных, появляется сообщение ?Extra ignored (Лишнее игнорируется).

Заключительный совет:

Используйте INPUT только в случае необходимости и вводите только по одному элементу данных. Скоро обнаружится, что этот оператор имеет много недостатков.

### 8-1-4. LINE INPUT

Этот оператор предоставляет наилучший способ ввода достаточно больших объемов символьных данных. Помните, что он позволяет также вводить последовательные файлы. Для ввода с клавиатуры он имеет следующий формат:

LINE INPUT"сообщение"; символьная переменная

Этот оператор переводит программу в режим SCREEN 0 или 1 и выводит на экран <сообщение> (необязательное). Вводиться может любой текст (даже с командных клавиш). При нажатии клавиши RETURN весь текст (максимум 255 символов) будет присвоен указанной переменой.

Проверьте следующую программу вводом с различных клавиш (например, CLS):

```
10  LINE INPUT "Ваш текст?"; A$
20  PRINT A$
30  GOTO 10
```

### 8-1-5. STRIG(0)

Эта функция проверяет, нажата клавиша пробела или нет. Она возвращает -1, если пробел нажат, и 0 - если нет. Проверка производится в момент выполнения функции STRIG(0) в программе.

В ожидании нажатия этой клавиши опрос будет выполняться постоянно, что делает возможной немедленную реакцию программы. Нажатие клавиши пробела может также проверяться с помощью INKEY$, но в этом случае компьютер будет проверять текущее состояние буфера клавиатуры.

Функция STRIG(0), однако, возвращает -1, если клавиша пробела нажата во время вычисления этой функции.

Пример:

```
10  FOR I = 0 TO 500
20  PRINT I
30  NEXT
40  IF STRIG(0) THEN 10 ELSE 40
```

Обратите внимание, что если клавиша пробела нажата во время вывода информации на экран, это событие не будет перехвачено функцией STRIG(0), как произошло бы в случае замены строки 40 следующей:

```
40  IF INKEY$ = CHR$(32) THEN 10 ELSE 40
```





### 8-1-6. STICK(0)

Эта функция, в сущности, работает так же, как и STRIG(0), но возвращает значения, соответствующие нажатию курсорных клавиш. Возвращаемые значения перечислены ниже:


Эта функция используется в первую очередь для управления перемещениями спрайтов.

Пример:

STICK

```
10  SCREEN 2
20  SPRITE$(0) = STRIG$(8,255)
30  X=120:Y=90
40  PUT SPRITE 0,(X,Y)
50  S=STICK(0):IF S=0 THEN 50
60  X=(X-(S>1 AND S<5)+(S>5))MOD 200
70  Y=(Y+(S<3)+(S=8)-(S>3 AND S<7))MOD 180
80  GOTO 40
```

## 8-2. Манипуляторы

В этом руководстве мы называем манипуляторами устройства, которые могут быть подключены к разъемам компьютера

JOYSTICK 1 и 2 (джойстик, МЫШЬ, графический планшет, аналоговый манипулятор...). Этот список, очевидно, не может быть полным, но поскольку при покупке такого манипулятора Вы получите к нему специальную инструкцию, мы не будем останавливаться на специфических особенностях манипуляторов.

### 8-2-1. STRIG

Когда речь идет о клавиатуре, эта функция действует как STRIG(0). Аргумент изменяется от 1 до 4 и имеет то же значение, что и для прерываний (см. Главу 7). Функция используется для джойстиков и, с некоторыми ограничениями, для МЫШИ.

| | | |
|---|---|---|
| STRIG(1) | джойстик 1 | триггер 1 |
| STRIG(2) | джойстик 2 | триггер 1 |
| STRIG(3) | джойстик 1 | триггер 2 |
| STRIG(4) | джойстик 2 | триггер 2 |

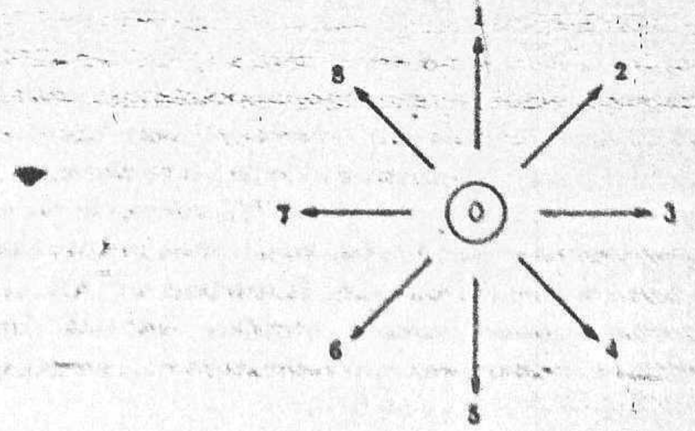
### 8-2-2. STICK

Аргумент может принимать значения 1 или 2, что соответствует номеру разъема для джойстика. Эта функция используется для джойстика и различных разновидностей МЫШИ почти так же, как и для клавиатуры.

Обратите внимание, что МЫШЬ имеет два режима функционирования, и в одном из них перемещения МЫШИ могут быть зафиксированы только подпрограммами в машинных кодах.








### 8-2-3. PAD и PDL

Эти функции позволяют вводить данные соответственно с графических планшетов и аналоговых игровых манипуляторов. Специфика применения этих функций изложена в соответствующих инструкциях.

# Музыкальные возможности MSX Бейсика

Глава 9

MSX Бейсик включает в себя "Музыкальный макроязык", позволяющий управлять звукогенератором компьютера. Этот генератор работает как независимый процессор: если музыкальные или другие звуковые эффекты запрограммированы, то они запускаются параллельно с исполнением основной программы.

Звукогенератор имеет три независимых выходных канала, допуская тем самым 3-голосный полифонический выход. Звуковой сигнал формируется генератором регулярных сигналов, частота которых (высота звука) может программироваться, либо широкополосным генератором шумов.
Амплитуда сигнала (громкость) может устанавливаться для каждого канала отдельно или модулироваться генератором огибающей.

Программирование звукогенератора осуществляется следующими операторами:

(1) PLAY
(2) SOUND

## 9-1. Оператор PLAY

Этот оператор можно сравнить с оператором DRAW, используемым в графике: его параметры составляют язык, довольно близкий к нотной записи. Общий формат этого оператора имеет следующий вид:

PLAY A$, B$, C$

где A$, B$ и C$ - строки символов, представляющих звуковые

"партии" каналов A, B и C соответственно. Символы в строках являются командами Музыкального макроязыка, перечисленными ниже.

**9-1-1. Высота звука**

**(1) Названия нот**

Каждая буква обозначает одну ноту:

**C D E F G A B**
**до ре ми фа соль ля си**

Пример:

```
PLAY"C D E F G A B"
```

Эта команда проиграет по каналу A гамму 4-ой октавы. Пробелы между символами необязательны, но они делают текст более ясным.

(2) Вспомогательные знаки

Для записи вспомогательных знаков используйте после символа ноты один из следующих символов:

| | |
|---|---|
| # или + | знак диеза |
| – | знак бемоля |

Пример:

```
PLAY"C C# D D# DD"
```

(3) Вариации октав
Имеется 8 октав (1-8). Если октава специально не указана, компьютер будет играть в 4 октаве. Она может быть изменена





символом O (не ноль, а буква O) с последующим номером желаемой октавы. Команда O влияет на все последующие ноты, пока октава не будет переопределена.

Пример:

```
PLAY"O2 C D O3 C D O4 C D"
```

(4) Номера нот

Вместо указания высоты ноты посредством обозначающей ее буквы, вспомогательных знаков и номеров октав, может использоваться компактная запись N с последующим номером. Имеющиеся ноты и соответствующие символы перечислены ниже:

| Октава | Название ноты/номер | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| | • | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2 | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 3 | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| 4 | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 5 | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 53 | 57 | 58 | 59 |
| 6 | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |
| 7 | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 |
| 8 | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 9 | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| | 96 | • | • | • | • | • | • | • | • | • | • | • |

В команде N, нота C (до) октавы 1 отсутствует; при использовании названий нот, отсутствует нота C (до) октавы 9.
Пример:

```
PLAY"N36 N38 N40 N41 N43 N45 N47"
```





### 9-1-2. Длительность нот

(1) Точка

Точка, следующая за нотой, увеличивает длительность ноты наполовину. Вторая точка добавляет половину длительности, полученной с первой точкой. Несколько точек после одной ноты вполне приемлемы.
Пример:

```
PLAY"C D . E .."
```

(2) Запись индивидуальной длительности ("вес" ноты)

За нотой может следовать признак длительности. Если нота записана в виде ее номера N, оба числа должны быть разделены точкой с запятой. Длительность ноты лежит в пределах от 1 до 64.
Основные веса нот и их значения перечислены ниже:

Пример:

| | |
|---|---|
| 1 | Целая нота |
| 2 | Половина |
| 4 | Четверть |
| 8 | Восьмая |
| 16 | Шестнадцатая |
| 32 | Тридцать вторая |
| 64 | Шестдесят четвертая |

Нестандартная длительность нот допускается. Если длительность не задана, компьютер играет "в четвертях".

```
PLAY"C1 D2 E3"
```

(3) Команда задания длительности группы нот(L)

Эта команда задает те же значения, что и приведенная выше форма записи. В отличие от нее команда L действует на все последующие ноты. По умолчанию ее значение - 1/4.

```
PLAY"L1 C D L64 C D E F"
```

(4) Темп (T)

Эта команда устанавливает темп. За ней следует значение, лежащее в пределах от 32 до 255, по умолчанию - 120. Это значение задает количество четвертей, проигрываемых в минуту.

Пример:

```
PLAY"T32 C D E F T255 C D E F"
```

### 9-1-3. Паузы (R)

Для введения паузы вместо символа ноты используется символ R. Длительность паузы задается так же, как длительность ноты.
Пример:

```
PLAY"C R1 C R32 C32"
```

### 9-1-4. Динамические отметки

Динамика (громкость нот) вводится с помощью команды V с последующим числом, лежащим в пределах от 0 до 15, (по умолчанию - 8. Эта команда действует на все последующие ноты.

Пример:

```
PLAY"C D V15 C D"
```





### 9-1-5. Специальные эффекты

Специальные эффекты, такие как тремоло или стаккато, могут создаваться посредством передачи управления громкостью генератору огибающей. Однако, при этом не следует использовать команду V, поскольку она инициализирует параметры генератора огибающей.
(1) Период огибающей (M)

Команда M с последующим значением, лежащим в пределах от 1 до 65535, задает период огибающей. По умолчанию он равен 255. Подробнее см. генератор SOUND.

(2) Форма огибающей (S)

Форма огибающей (определяющая эволюцию звука) устанавливается командой S, за которой следует одно из перечисленных значений: 1, 4, 8, 10, 11, 12, 13 или 14. См. ниже оператор SOUND.

Пример:

```
PLAY"M1600 S8 C D E F G A B"
```

### 9-1-6. Использование переменных

Следующие команды воздействия на ноты, а также команда N, могут сопровождаться переменными. Они имеют следующий формат:

<КОМАНДА> = <переменная>; (точка с запятой обязательна)

Имеются следующие команды с переменными:

| | |
|---|---|
| O | Октава |
| N | Номер ноты |
| L | Длительность ноты |
| T | Темп |
| V | Громкость |
| M | Период огибающей |
| S | Форма огибающей |

Пример:

```
10  FOR I = 120 TO 255 STEP 10
20  PLAY"T = I; C D E F"
30  NEXT
```

### 9-1-7. Подпрограммы

Три строки символов оператора PLAY могут собраны при помощи функций или операторов, работающих с символьными переменными. Такой оператор, как

```
PLAY A$ + B$, C$, D$
```

вполне приемлем, если переменные с A$ по D$ содержат исключительно команды Музыкального макроязыка.
Команда X облегчает создание "музыкальных" подпрограмм. Она имеет следующий формат:

X<символьная переменная>; (точка с запятой обязательна)

Пример:

```
10  A$ = "D C F"
20  PLAY"V15 XA$; V6 XA$;"
```





Подпрограммы могут быть вложенными, что упрощает запись повторяющихся данных. Все повторяющиеся части звуковой партии могут быть записаны в символьных переменных и в случае необходимости вызываются командой X.

### 9-1-8. Работа с продолжительными звуковыми партиями

Когда выполняется оператор PLAY, строки, соответствующие каждому голосу (содержащие максимум 255 символов), записываются в трех т. наз. о ч е р е д я х. Затем они автоматически исполняются, в то время как основная программа может заниматься чем-либо другим. После завершения команды макроязыка удаляются из очередей. Когда все очереди пусты, музыка прекращается. Может выполняться новый оператор PLAY, но его выполнение, как правило, должно начаться сразу же после считывания из очереди последней команды предыдущего оператора PLAY, и даже в этом случае загрузка новых очередей вызовет задержку.
Однако, в MSX Бейсике имеется функция, выясняющая, имеются данные в очередях или нет. Эту функцию (PLAY) не следует путать с оператором PLAY.

```
PLAY(0)
```

возвращает -1, если данные имеются в очередях, и 0, если все данные уже прочитаны. PLAY(<номер голоса>) возвращает -1, если в указанном голосе очередь не пуста, и 0, если очередь пуста. Номера голосов распределяются следующим образом:

| | |
|---|---|
| 1 | Голос A |
| 2 | Голос B |
| 3 | Голос C |

Пример:

```
10   A$ = "CDEFGAB"
20   PLAY A$," O5 XA$;", "O6 A$;"
30   IF PLAY(0) THEN 20 ELSE 30
```

### 9-1-9. Прекращение воспроизведения звука

В программе оператор BEEP останавливает воспроизведение звука. В командном режиме достаточно нажать CTRL + G. Помните, однако, что ошибка в программе всегда вызывает BEEP. Поэтому нажатие любой символьной клавиши с последующим RETURN останавливает воспроизведение звука.

## 9-2. Оператор SOUND

Этот оператор позволяет программировать разнообразные звуковые эффекты. Использование его, однако, сложнее, чем оператора PLAY.
Звукогенератор имеет четырнадцать 8-битовых регистров и в зависимости от значений в этих регистрах воспроизводит различные звуки. Оператор SOUND загружает эти регистры. Он имеет следующий формат:

SOUND <регистр>, <значение>

Регистры пронумерованы от 0 до 13, а значения в них не должны превышать 255. Эти регистры имеют следующее назначение:

### 9-2-1. Регистры, устанавливающие частоту звука

Имеется 6 регистров, задающих частоту звука (2 на каждый канал).





| | |
|---|---|
| Канал A | регистры 0 и 1 |
| Канал B | регистры 2 и 3 |
| Канал C | регистры 4 и 5 |

Фактически, частота в каждом канале кодируется 12 битами: так, регистр 0 содержит 8 младших битов, задающих высоту звука, а регистр 1 содержит 4 старших бита (канал A). Значения, представимые на каждой паре регистров, лежат таким образом в пределах от 0 до 4095.
Значения в регистрах не задают частоту непосредственно. Она задается следующей формулой:

Частота (Гц) = F/FV
где F = 3579545/32 (Гц),
а FV = <значение из пары регистров>

Задание частоты канала A, например, требует установки пары регистров 0 и 1. FV будет получено по формуле

FV = R0 + 256*R1

Пример:
Нужно, чтобы частота канала A была 440 Гц. Что для этого должно быть введено в регистры 0 и 1?

(1) Вычислить FV:
FV = F/440 = 3579545/32/440 = 254.23
(2) R0 = FV MOD 256 = 254
(3) R1 = FV\256 = 0

Таким образом, в регистр 0 вводится 254, а в регистр 1 - 0.

Примечание:

(1) Если у Вас в обоих регистрах 0, Вы не получите неопределенной частоты: звук просто отключится.

(2) Минимальная частота определяется по следующей формуле:

3579545/32/4095 = 37.31 Гц

(3) Поскольку ширина полосы звукового канала не превышает на практике 10000 Гц, минимальное значение, которое может быть введено в пару регистров, задается следующей формулой:

FV = 3579545/32/10000 = 11.19

Таким образом, 11 будет помещено в регистр 0, а 0 - в регистр 1. Звук уже в значительной степени смягчился, но еще кое-что слышно. Фактически, предел слышимой частоты зависит как от восприятия, так и от источника звука.

### 9-2-2. Генератор шумов

Хотя имеется три звукогенератора, генератор шумов только один. Шум может рассматриваться как звук, составленный из большого количества компонент с различными частотами.
Регистр 6 устанавливает "среднюю частоту" шума. Эта частота кодируется 5 битами, следовательно, регистр 6 может загружаться значениями от 0 до 31.
Результирующая частота выводится по следующей формуле:

Частота (Гц) = F/R6
где F = 3579545/32,
а R6 = <содержание регистра 6>





### 9-2-3. Микширование каналов

Регистр 7 используется для микширования (смешивания) звука и шума на каждом канале. В каждом из каналов возможен следующий выходной сигнал:

| |
|---|
| Ничего |
| Только звук |
| Только шум |
| Звук и шум |

Первые два бита регистра 7 должны быть двоичным числом 10. Следующие три запрещают (1) или разрешают (0) выход шума. Последние три запрещают (1) или разрешают (0) выход звука. Порядок каналов обратный:

1 0 C B A C B A

Шум Звук

Пример:

SOUND 7,&B10110011
Канал A: шум
Канал B: ничего
Канал C: звук

### 9-2-4. Установка громкости

Регистры с 8 по 10 устанавливают громкость.

| |
|---|
| Регистр 8: канал A |
| Регистр 9: канал B |
| Регистр 10: канал C |

Установка лежит в пределах от 0 до 15 (15 соответствует мак-

симальной громкости). Если установка равна 16, звук контролируется генератором огибающей.

Примеры:

```
SOUND 8, &B00010000
SOUND 9, &B00001111
```

Канал A: громкость контролируется генератором огибающей
Канал B: громкость = 15

### 9-2-5. Генератор огибающей

Имеется только один генератор огибающей. Можно выбирать форму и частоту огибающей.

(1) Частота модуляции

Частота огибающей кодируется в регистрах 11 и 12. Используются все 8 битов этих регистров. Диапазон значений, таким образом -от 0 до 65535. 0 не производит никакого эффекта. Регистр 11 содержит младшие биты, а регистр 12 - старшие. Частота огибающей задается следующей формулой:

Частота (Гц) = F/FV
где F = 3579545/512
и FV = R11 + 256*R12

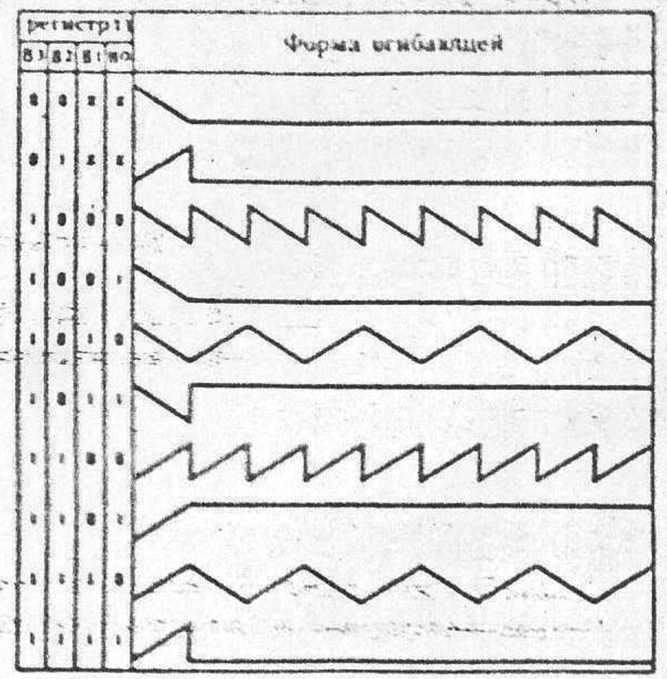
Минимальное значение частоты, следовательно, равно 0.107 Гц, что приблизительно соответствует периоду 9.4 секунды.
Максимальное значение оценивается около 6991.3 Гц и тем самым принадлежит к слышимым частотам.

(2) Форма огибающей

4 младших бита регистра 13 позволяют выбрать любую из следующих заранее заданных форм огибающей:








X = 0 или 1

### 9-2-6. Использование оператора SOUND

Этот оператор позволяет создавать специальные эффекты. Для музыки более удобен PLAY. Для прекращения звука достаточно выполнить BEEP или сбросить регистры 8-10 на 0. Для иллюстрации возможностей SOUND, рассмотрим следующие примеры:

(1) Звук бомбардировщика

BOMBER

```
10   BEEP
20   FOR I = 0 TO 13
30   READ V : SOUND I, V
40   NEXT
50   END
100  DATA 200, 14, 220, 14, 240, 14
110  DATA 0
120  DATA &B10111000
130  DATA 15, 15, 15
140  DATA 0, 0, 0
```

Формирование слегка различающихся частот в трех каналах имитирует "биения" в шуме пропеллеров, вращающихся со слегка разными скоростями.

(2) Звук моря

Замените данные в строках с 100 по 140 следующими:

SEA

```
100  DATA 0, 0, 0, 0, 0, 0
110  DATA 30
120  DATA &B10110111
130  DATA 16, 0, 0
140  DATA 0, 90, &B1110
```

В обоих примерах использовались только статические данные. Теперь попробуем изменять значение некоторых регистров во время выполнения. Послушаем звук взлетающего вертолета:

(3) Вертолет

Сначала замените данные в предыдущей программе:





COPTER

```
100  DATA 0, 0, 255, 14, 255, 14
110  DATA 10
120  DATA &B10000001
130  DATA 8, 8, 16
140  DATA 0, 20, &B1100
```

Затем добавьте следующие строки:

```
50   FOR I = 5120 TO 300 STEP - 10
60   SOUND 11, I MOD 256
70   SOUND 12, I\256
80   FOR J = 0 TO 20
90   NEXT J, I
```

(4) Звук бомбы

BOMB

```
10   FOR I = 0 TO 13
20   READ V : SOUND I, V
30   NEXT
40   FOR I = 50 TO 200
50   SOUND 0, I
60   A = 5 * 5
70   NEXT
80   SOUND 12, 100 : SOUND 13, 9 SOUND 7,
     &B10011111
90   END
100  '
110  DATA 50, 0, 0, 0, 0, 0
120  DATA 30
130  DATA &B10111110
140  DATA 10, 0, 16
150  DATA 0, 0, 0
```

Строка 70 вводит задержку на несколько миллисекунд.

# Инициализация в MSX Бейсике

Глава 10

При включении компьютера он приводится автоматически в некоторое исходное рабочее состояние (инициализируется); так, экран устанавливается в режим SCREEN 0 и окрашивается в синий цвет.

Версия 2.0 MSX Бейсика (только на компьютерах MSX-2) имеет некоторые операторы для дополнительной инициализации. Действие этих операторов в своем роде уникально: будучи раз выполнены, они воз действуют на компьютер при каждом включении (установленные ими значения запоминаются в небольшой по объему памяти, сохраняющей свое содержимое при отключенном основном питании).

**(1) SET ADJUST (X,Y)**

Регулирует положение используемой части экрана. X и Y могут изменяться от -7 до 8, а начальным значением является (0,0).

**(2) SET BEEP** <звук>,<громкость>

Переопределяет звук BEEP. Параметр "звук" изменяется от 1 до 4, так же, как и параметр "громкость".

**(3) SET TITLE"**<заголовок>", цвет

Параметр "заголовок" (строка длиной до 6 символов) позволяет переопределять заголовок, возникающий на экране при включении компьютера. Если эта строка имеет длину 6 символов, система будет ожидать нажатия любой клавиши после вывода заголовка на экран. Параметр "цвет" (может опускаться) лежит в пределах от 1 до 4.

(4) SET PASSWORD"<пароль>"

Выполнение этого оператора позволяет вводить пароль, проверяемый при включении компьютера. Его длина может быть от 0 до 255 символов.

GRAF + STOP

(5) SET PROMPT"подсказка"

Переопределяет подсказку Ok. Новая подсказка может быть до 6 символов длиной.

Примечание:

Можно сохранить результат действия только одного из трех операторов этой группы: последний "замещает" предыдущий. Если нужно отменить пароль, достаточно ввести SET PROMPT"Ok".

(6) SET SCREEN

Этот оператор позволяет сохранить параметры одного из следующих операторов, выполненного в командном режиме: SCREEN, COLOR, KEY ON/OFF или WIDTH.
SCREEN имеет следующие параметры:

SCREEN SM, SS, KE, BD, PO, DM

SM - режим экрана (0-8)
SS - размер и масштаб спрайта (0-3)
KC определяет отзвук клавиши (0-1)

| | |
|---|---|
| 0 | отзвука нет |
| 1 | отзвук |





BD - скорость обмена данными с кассетным накопителем на магнитной ленте (1-2)

| | |
|---|---|
| 1 | 1200 бод |
| 2 | 2400 бод |

PO - варианты принтера

| | |
|---|---|
| 0 | принтер MSX |
| < > 0 | другой принтер (графические символы заменяются пробелами) |

DM - режим вывода на монитор (0-3)

| | |
|---|---|
| 0 | нормальный |
| 1 | обычная развертка |
| 2 | прогрессивная развертка; попеременный (четный/нечетный) вывод |
| 3 | обычная развертка; попеременный (четный/нечетный) вывод |

Сохраняются только те параметры SCREEN, которые могут вводиться в командном режиме, и некоторые другие установки.

* SM SCREEN режим (0-1)
* WIDTH (1-40, если SM = 1)/(1-80, если SM = 0)
* Цвет текста (0-15)
* Цвет фона (0-15)
* Цвет границы (0-15)
* KC (0-1)
* KEY ON/OFF выводит на экран тексты функциональных клавиш (0-1)
* PO принтер MSX или не MSX (0-1)
* BD скорость обмена 1200 или 2400 бод (0-1)
* DM (0-3)

(7) SET DATE <дата>, A

Определяет дату в формате:

"ГГ/ММ/ДД"

ГГ = две последние цифры года
ММ = месяц
ДД = день

Пример: "86/05/12" (нули обязательны)

Если присутствует A, дата будет рассматриваться как дата подачи сигнала ("будильник с календарем").

(8) SET TIME <час>, A

Время задается в следующем формате:

"ЧЧ:ММ:СС" (нули обязательны)

Присутствие A задает время подачи сигнала ("будильник", секунды игнорируются).

Примечание:

Используя A, всегда вводите сначала время, а затем дату.

Для получения даты и времени MSX Бейсик предоставляет также еще два оператора:

(1) GET DATE <символьная переменная>, A

Этот оператор присваивает дату переменной. Дата, введенная с A, будет присвоена в таком виде переменной.





(2) GET TIME <символьная переменная>, A

Этот оператор присваивает время переменной. Если присутствует A, время в таком виде будет присвоено переменной.

Примечание:

Подача сигнала не является автоматической. Для ее осуществления необходимо вставить подпрограмму, сравнивающую текущие дату и время с заданными с параметром A, и в случае совпадения выдающую серию BEEP или сообщения.

# Краткие сведения о численных методах

Глава 11

Научно-технические расчеты на компьютере относятся скорее к численным методам решения задач, чем к "обычной" математике. дело с интегралом. В "обычной" математике используются либо стандартные процедуры, либо таблицы интегралов. Если интеграл известен, то значение интеграла получается по формуле в соответствии с пределами интеграла. Неклассический интеграл требует перевода в стандартную форму. Это занятие часто требует часов кропотливой работы и головоломных трюков.

При работе с компьютером применяется совершенно другой подход. Компьютер быстро выполняет операции с числами. Поэтому имея исходные числа, можно по определенному алгоритму получить результат, который также является числом, а не выражением.

Численные методы весьма разнообразны. Однако, теоретическое введение в численные методы в нашем руководстве было бы просто неуместно; поэтому мы приведем лишь несколько примеров в качестве иллюстрации к типичным методам.

Эти примеры даны в виде документированных подпрограмм, которые все начинаются со строки 1000. В случае необходимости, их легко можно объединить в единую программу. На практике эти примеры могут служить исходным материалом, если Вы захотите создать библиотеку вычислительных подпрограмм.

## 11-1. Численное интегрирование

Начнем с подпрограммы интегрирования, использующей метод

Симпсона. Этот метод делит интервал интегрирования на подинтервалы, на которых интегрируемая функция приближается параболами. Этот метод не подходит для интегралов, имеющих бесконечные пределы, или для функций, превращающихся в бесконечные на своих интервалах
интегрирования.

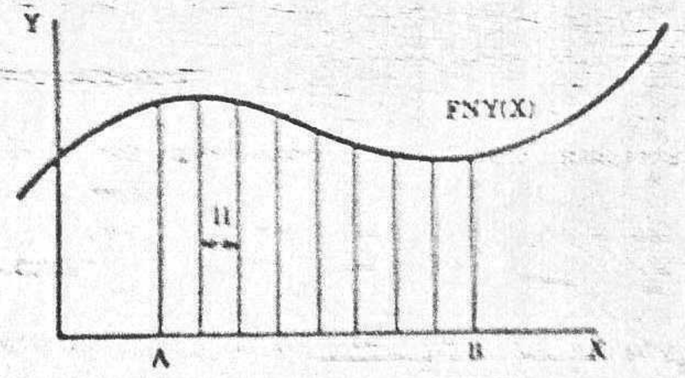
SIMPSON

```
1000  'ИНТЕГРИРОВАНИЕ ПО СИМПСОНУ
1010  'ИСХОДНЫЕ ДАННЫЕ: A верхний предел
                        нижний предел B
                        функция FNY
1020  '
1030  '
1040  'РЕЗУЛЬТАТ: значение интеграла S
1050  'ИСПОЛЬЗУЕМЫЕ ПЕРЕМЕННЫЕ: S0
1060  '                          N
1070  '                          H
1080  '                          C
1090  '                          I
1100  '
1110  N=5:S=0
1120  H = (B - A)/2/N : S0 = 0
1130  FOR I + 0 TO 2•N
1140  IF I = 0 OR I = 2•N THEN C = 1 : GOTO
      1170
1150  IF (I MOD 2) = 0 THEN C = 2 : GOTO 1170
1160  C = 4
1170  S0 = S0 + C•FNY(A + I•H)
1180  NEXT
1190  S0 = S0•H/3
1200  SWAP S, S0
1210  IF ABS(S - S0) < 1E-8 THEN 1120
```





Интервал (A,B) разделен на 2•N сегментов. Первоначально N=5. Применяется метод Симпсона; результат находится в S0. S и S0 затем меняются местами (строка 1200).
Осуществляется проверка точности и, если она удовлетворительна, происходит выход из подпрограммы. В противном случае, N получает приращение, и процесс продолжается.




Для проверки этого метода используйте следующую программу:

```
10  DEFFY(X) = 3•X^2
20  A = 0
30  B = 1
40  GOSUB 1000
50  PRINT S
60  END
```

Поскольку первообразующая 3•X^2 равна X^3, результат должен быть равен 1.

Теперь замените строку 10 следующей:

```
10   DEFFNY(X) = 4/(1 + X^2)
```

Поскольку этот интеграл равен точно 4•ATN(1), результат в значительной степени должен приближаться к числу "пи". Если Вас убедили эти два примера, Вы можете использовать эту подпрограмму для вычисления различных интегралов. Нужно только определить интегрируемую функцию пользовательской функцией FNY и задать пределы интегрирования в A и B.

Несколько важных понятий

(1) Метод, который мы используем, называется АЛГОРИТМОМ. Алгориты указывает исходные данные и метод их обработки для получения конечного результата. Скорость вычислений и точность результата в значительной степени зависят от алгоритма.

(2) Разделение интервала интегрирования называется ДИСКРЕТИЗАЦИЕЙ. В математике работа выполняется с изменяющимися действительными числами. Здесь мы должны использовать числа с ограниченной точностью, которые изменяются только дискретно.

(3) Слишком грубая дискретизация возвращает неточный результат. Следует использовать более точную дискретизацию (ИТЕРАЦИЯ) до тех пор, пока не будет достигнут приемлемый результат. Алгоритм хорош, когда он быстро ПРИБЛИЖАЕТ к решению.

## 11-2. Вычисление функций Бесселя

Этот метод рассчитан на получение значений функций Бесселя первого рода в виде степенных рядов с целыми показателями степени:

$$J_n(x) = \left(\frac{x}{2}\right)^n \sum_{k=0}^{\infty} \frac{\left(-\frac{x^2}{4}\right)^k}{k!(k+n)!}$$





BESSEL

```
1000   'ФУНКЦИИ БЕССЕЛЯ
1010   'ИСХОДНЫЕ ДАННЫЕ: порядок функции
       N
1020   '                          аргумент
                                  X
1030   'РЕЗУЛЬТАТ:                B
1040   'ИСПОЛЬЗУЕМЫЕ ПЕРЕМЕННЫЕ:  Y
1050   '                          Z
1060   '                          D
1070   '                          T
1080   '                          K
1090   '                          I
1100   IF X = 0 AND N = 0 THEN B = 1 :
       RETURN
1110   IF X = 0 AND N < > 0 THEN B = 0 :
       RETURN
1120   D = 1
1130   FOR I = 1 TO N
1140   D = D/I
1150   NEXT
1160   D = D•(X/2) ^ N
1170   Y = -(X/2) ^ 2
1180   B = 1 : K = 1 : T = 1
1190   T = T•Y/K/(K + N)
1200   B = B + T
1210   K = K + 1
1220   IF ABS(T) > 0 THEN 1190
1230   B = B•D
1240   RETURN
```

Можно проверить эту подпрограмму с помощью следующей программы:

```
10   INPUT"N"; N
20   INPUT"X"; X
30   GOSUB 1000
40   PRINT CSNG(B)
50   GOTO 10
```

Введите различные значения для N и X. Если у Вас есть таблицы, Вы можете заметить, что подпрограмма обеспечивает по меньшей мере 5 правильных разрядов для N в пределах от 0 до 11 и для X в пределах, от 0 до 20.

### 11-3. Програма ускорения сходимости

Когда алгоритм обеспечивает решение путем последовательных приближений, иногда может оказаться полезной программа ускорения сходимости. Такая программа использует последние результаты, полученные в ходе последовательных приближений, и должна получать значение, которое ближе к конечному результату, чем последнее приближение.

Программа ускорения сходимости, представленная ниже, использует три последних результата S1, S2 и S3; в ней применяется следующая формула:

$$S = (S_2 \cdot 2 - S_1 \cdot S_2)/(2 \cdot S_2 - S_1 - S_2)$$

Эта программа записывается следующим образом:





CACCEL

```
1000   'УСКОРИТЕЛЬ ВЫВЕДЕНИЯ
1010   'ВВОДЫ:      S1 S2 S3
1030   'ВЫХОД:      S
1040   '
1050   S = (S2 * 2 - S1*S3)/(2*S2 - S1 - S3)
1060   RETURN
```

Опробуем ее на следующей последовательности:

$$1 + 1/2 + 1/4 + 1/8 + 1/16 + \ldots$$

которая, как известно, стремится к 2.

```
10   SCREEN 0
20   T = 1 : S1 = 1
30   T = T/2 : S2 = S1 + T
40   T = T/2 : S3 = S2 + T
50   GOSUB 1000
60   PRINT S1
70   PRINT S2
80   PRINT S3;TAB(20);S
90   PRINT
100  T = T/2 : S1 = S3 + T
110  IF INKEY$ = "" THEN 110
120  GOTO 30
```

Эта программа печатает три первых приближения и их "сведенное" значение. Это значение при первой попытке правильно. Нажмите любую клавишу для получения следующих значений.

Обратите внимание, что после нескольких итераций подпрограмма обнаруживает деление на ноль.

Повторите попытку с другой последовательностью:

1 + 1/5 + 1/25 + 1/125 + 1/625 + ...

которая стремится к 1/(1-1/5) = 1.25 (достаточно заменить T/2 на T/5 в строках 30, 40 и 100).

Вы увидите, что выведенное значение правильно при первой попытке, но начинает отличаться от него при пятой попытке.

Это типично для всех программ ускорения сходимости: сначала они работают быстрее, чем "грубые" алгоритмы, но неожиданно начинают выдавать неверные результаты. БУДЬТЕ ВНИМАТЕЛЬНЫ!

Примечание:

Компьютер MSX может успешно применяться в научно-технических расчетах. Такие функции, как SIN, LOG и т. д. всегда вычисляются в формате с двойной точностью, что нечасто встречается даже у более дорогих компьютеров. Более того, когда число одинарной точности присваивается переменной двойной точности, 8 разрядов, добавляемых к значению, всегда равны 0 (так что значение числа не изменяется). Некоторые компьютеры при этом добавляют цепочку непредсказуемых разрядов, из-за чего следует быть очень осторожным, чтобы в вычислениях, требующих высокой точности, не возникали абсурдные результаты.



# Разработка программ

Глава 12

Создание небольших программ (до 10 строк) обычно не вызывает проблем: они непосредственно вводятся с клавиатуры и могут быть легко исправлены, если возникает такая необходимость.

Однако такая процедура может привести к настоящим катастрофам, когда она применяется к более сложным программам. Не дописав программу до конца, Вы можете забыть, что там в начале. При таком *лобовом* подходе логика программы становится беспричинно сложной и практически нечитаемой.

Общее правило: отладка программы всегда занимает больше времени, чем ввод. Следовательно, программа должна восприниматься с учетом необходимости последующего редактирования.

Читатель, переходящий к разработке сложных программ, найдет в этой главе введение в наиболее устоявшиеся методы.

## 12-1. Структурирование программы

### 12-1-1. Деление программы на подпрограммы

Для деления программы на подпрограммы имеются, очевидно, две причины:

(1) Удобно, когда программа разделяется на участки, каждый из которых выполняет хорошо определенную часть программы и умещается на одном экране.

Этот процесс разбиения на блоки должен идти до тех пор, пока каждый блок не будет настолько детализирован, что может быть прямо переведен в короткую подпрограмму на MSX Бейсике.


До сих пор использовался только метод нисходящего проектирования. Детальное рассмотрение подпрограмм нижнего уровня часто показывает, что некоторые из них действительно очень похожи; это нормально, поскольку некоторые процессы должны выполняться в программе несколько раз. Возможно ли написать только одну подпрограмму, которая будет выполнять работу во всех случаях? За исключением небольших изменений в функциях ввода/вывода, это довольно часто возможно. Если же это - подпрограмма вычисления, не исключено, что она уже написана Вами или найдена в библиотеке.





### 12-2-2. Прерывания

Если Вы используете прерывания, следует указать места, в которых они должны происходить. Нужно также подумать о соответствующих подпрограммах обработки.

### 12-2-3. Написание программы

Теперь каждая подпрограмма достаточно хорошо определена, чтобы позволить Вам непосредственно перейти к работе за клавиатурой или начать писать ее на бумаге (в конце концов, это дело вкуса).

Однако прежде чем перейти к клавиатуре, должно быть сделано следующее:

(1) Составьте список переменных

Это позволит убедиться в отсутствии конфликтов (особенно тщательно проверяйте счетчики циклов) и в случае необходимости ввести переменные в начале программы.

(2) Присвойте номера строк каждой подпрограмме

Например, присвойте 200 номеров строк каждой подпрограмме, чтобы осталось достаточно свободного места.
Теперь можно перейти к работе за клавиатурой. Программа, не принадлежащая к самому элементарному уровню, представляет собой просто серию вызовов подпрограмм, заметных в тексте по операторам GOSUB. Не бойтесь добавлять короткие комментарии, напоминающие о функции вызываемой подпрограммы.

Пример:

```
100 GOSUB 1000' Обнаруживает клавиши ESC и
    RETURN
```

На самом элементарном уровне каждой подпрограмме должен предшествовать короткий комментарий, указывающий ее функцию и требования по вводу/выводу. Использованные переменные и их значения также должны быть указаны в комментарии.

Разумеется, эти комментарии "съедают" место в памяти, однако они неоценимы при внесении изменений в текст. Если
Вы обращаетесь к давно написанной программе, Вы скоро заметите, что время, потраченное на вставку комментариев, не было потрачено зря. Более того, всегда можно хранить документированную версию, а использовать сжатую.

Законченной программе обычно будут предшествовать некоторые операторы инициализации. По большей части, в начале программы размещаются следующие операторы:

(1) **MAXFILES** определяет количество одновременно открытых файлов.

(2) **CLEAR** устанавливает размер строкового пространства и область программ в машинных кодах (см. далее); никогда не используйте **CLEAR** в подпрограммах.

(3) **DEFINT/SNG/DBL/STR** (объявляйте явно типы как можно большего числа переменных в программе).

(4) **DIM** для объявления массивов переменных.





(5)**R = RND(-TIME)**, если используются случайные числа.

## 12-3. Пример программирования

Попробуем составить программу, которая сможет выводить на экран Ваш биоритм.

В соответствии с некоторой теорией, реакция индивида на его окружение испытывает периодические флуктуации. Фактически, мы настроены на три периодических фактора:

| | |
|---|---|
| Физический: | 23-дневный цикл |
| Эмоциональный: | 28-дневный цикл |
| Интеллектуальный: | 33-дневный цикл |

Эти кривые инициализируются при рождении и затем изменяются со своей собственной частотой. Похоже, что пересечения с абсциссой составляют критические моменты. Некоторые страховые компании идут настолько далеко, что даже принимают эти кривые в расчет.

Мы займемся созданием программы и оставим открытыми любые вопросы, касающиеся интерпретации результатов...

### 12-3-1. Входы, выходы и обработка

Программе понадобится дата рождения и текущая дата. Затем она вычислит три кривые на интервале одного месяца, центрированном относительно текущей даты, и выведет их на экран.

Но это слишком легко! Что вы скажете относительно дня, в который Вы родились (Понедельник? Вторник?..), или сколько

дней Вы прожили до сегодняшнего дня? Программа должна будет показать еще и эти результаты. Кроме того, должна существовать возможность начать все заново, не выходя из программы.

### 12-3-2. Разделение на подпрограммы второго уровня

★ Подпрограмма 1 (ДАТА РОЖДЕНИЯ)
Выводит сообщение "Введите дату рождения".
Запрашивает Y (год).
Запрашивает M (месяц).
Запрашивает D (день).
Присваивает (Y,M,D) (Y0,M0,D0) и вычисляет N (количество дней, отделяющих эту дату от 1 января 1901 года)
Присваивает N переменной N0.
Начинает сначала, если переменная V (верификация) равна 1.

★ Подпрограмма 2 (ВЫБРАННАЯ ДАТА)
Выводит сообщение "Введите выбранную дату".
Запрашивает Y (год).
Запрашивает M (месяц).
Запрашивает D (день).
Присваивает (Y,M,D) (Y1,M1,D1) и вычисляет N (количество дней, отделяющих эту дату от 1 января 1901 года)
Присваивает N переменной N1.
Начинает сначала, если переменная V (верификация) равна 1.

★ Подпрограмма 3 (КОЛИЧЕСТВО ПРОЖИТЫХ ДНЕЙ)
Вычисляет N = N1 − N0.
Выполняет возврат к подпрограмме 1, если N отрицательно.





★ Подпрограмма 4 (ДЕНЬ РОЖДЕНИЯ)
Присваивает N0 переменной N2.
Вычисляет день N2 (D$).
Присваивает D$ переменной D0$.

★ Подпрограмма 5 (ВЫБРАННЫЙ ДЕНЬ)
Присваивает N1 переменной N2.
Вычисляет день N2 (D$).
Присваивает D$ переменной D1$.

★ Подпрограмма 6 (КОНЕЧНЫЙ ВЫВОД НА ЭКРАН)
Выводит на экран введенные данные с указанием дня недели и количества прожитых дней (текст).
Вычисляет интервал, центрированный вокруг выбранной даты и распространяет его до 31 дня; выводит дни недели и соответствующие им числа.
Прочерчивает рамку графика
Прочерчивает график трех кривых.

Обратите внимание, что некоторые части подпрограмм второго уровня выполняют похожую работу (это не совпадение: мы использовали славную технику нисходящего проектирования, одновременно учитывая достоинства сборочного проектирования). Эти участки естественно сделать подпрограммами третьего уровня.

### 12-3-3. Подпрограммы третьего уровня

★ Подпрограмма 7 (ГОД)
Устанавливает переменную V на 0.
Выводит сообщение "Год (4 цифры)" и передает введенное число в Y.
V = 1, если Y не находится между 1901 и 1999.

* Подпрограмма 8 (МЕСЯЦ)
Выводит сообщение "Месяц (2 цифры)" и передает введенное число в M.
V = 1, если M не находится между 1 и 12.

* Подпрограмма 9 (ДЕНЬ)
Выводит сообщение "День (2 цифры)" и передает введенное число в D.
V = 1, если D не находится между 1 и количеством дней в месяце (M); это последнее значение будет прочитано из строки DATA.

* Подпрограмма 10 (КОЛИЧЕСТВО ДНЕЙ)
Вычисляет количество дней N между (Y, M, D) и 1 января 1901; не забудьте високосные годы.

* Подпрограмма 11 (ДЕНЬ НЕДЕЛИ)
Вычисляет день недели (понедельник, вторник ...) даты, которая на N2 дней позже 1 января 1901 года, и записывает результат в D$. Это соответствует чтению "по модулю 7" списка дней недели введенных как данные (с первым днем, соответствующим 1 января 1901 года).

* Подпрограмма 12 (ВЫВОД НА ЭКРАН ТЕКСТА)
Выводит на экран сообщение "Дата рождения" с последующей полной датой.
Выводит на экран сообщение "Выбранная дата" с последующей полной датой.
Выводит сообщение "Прожито дней" с последующим N.

* Подпрограмма 13 (ИНТЕРВАЛ)
Вычисляет день недели и дату по 31-дневному интервалу с центром на выбранном дне.
Выводит эти данные на экран.



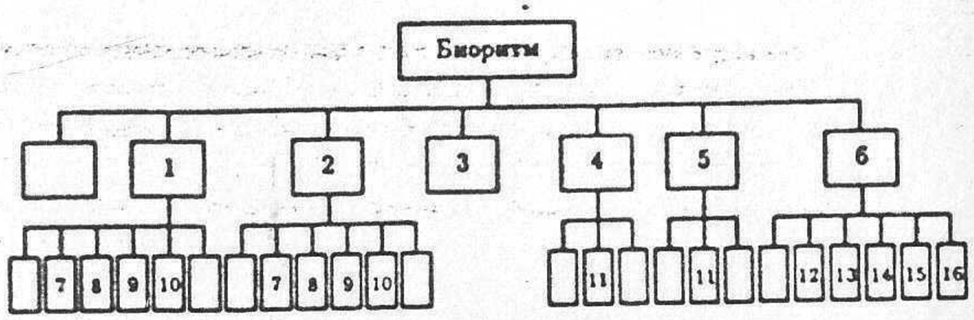


* Подпрограмма 14 (РАМКА)
Рисует границу графика.
Выводит название цвета каждой кривой.

* Подпрограмма 15 (ГРАФИК)
Рисует три кривые.

* Подпрограмма 16 (СООБЩЕНИЕ)
Выводит небольшое сообщение с просьбой нажать клавиши.

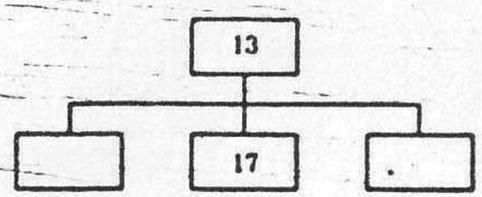
К этому моменту, наша программа может быть представлена следующим образом:




(Пустые блоки соответствуют операциям, включенным в саму программу).

### 12-3-4. Подпрограмма последнего уровня

Подпрограмма ИНТЕРВАЛ довольно сложна. Она нуждается в подпрограмме (ДЕНЬ N), возвращающей число и день недели для всех выводимых дат.
Поэтому ИНТЕРВАЛ имеет следующий формат:




### 12-3-5. Номера строк и инициализация

Строки с 10 по 190 зарезервированы для заголовка, с 200 по 390 - номер подпрограммы.
Подпрограммы начинаются со строк с номерами 400 + 200•n, где номер подпрограммы.

Ввод будет осуществляться в режиме SCREEN 0 (KEY OFF), а вывод результата - в режиме SCREEN 7.
Следовательно, должно быть активизировано устройство "GRP:". Кроме того, должны быть определены целые переменные, кроме PI, которая понадобится для вычисления синуса.

Теперь все готово к началу кодирования. Некоторые подпрограммы были описаны в весьма общих чертах, но они





будут проверены в ходе работы.

Примечание:

Эта программа будет работать только на компьютерах MSX-2, поскольку она требует высокого разрешения и работы с цветами.

BRHYTHM

```
10  '+-------------------------+
20  '|                         |
30  '|                         |
40  '|   БИОРИТМ               |
50  '|                         |
60  '|                         |
70  '+-------------------------+
80  '
200 '------ИНИЦИАЛИЗАЦИИ--------
210 '
220 SCREEN 0 : WIDTH 40
230 COLOR 15, 4
240 OPEN "GRP:" AS #1
250 KEY OFF
260 DEFINT A-Z : DEFSNG P
270 PI = 4*ATN(1)
280 '
400 '------ОСНОВНАЯ ПРОГРАММА------
410 '
420 GOSUB 600'     -1-----ДАТА РОЖДЕНИЯ
430 GOSUB 800'     -2-----ВЫБРАННАЯ ДАТА
440 GOSUB 1000'    -3-----КОЛИЧЕСТВО
                          ПРОЖИТЫХ ДНЕЙ
450 GOSUB 1200'    -4-----ДЕНЬ РОЖДЕНИЯ
460 GOSUB 1400'    -5-----ВЫБРАННЫЙ ДЕНЬ
480 GOSUB 1600'    -6-----КОНЕЧНЫЙ ВЫВОД
                          НА ЭКРАН
```

```
490 '
500 K$ = INKEY$ : IF K$ = "" THEN 500
510 IF K$ = CHR$(27) THEN END
520 IF K$ = CHR$(13) THEN RUN
530 GOTO 500
540 '
600 '-------ДАТА РОЖДЕНИЯ----------
610 '
630 PRINT"<ВВЕДИТЕ ДАТУ ВАШЕГО
    РОЖДЕНИЯ>":PRINT:PRINT:PRINT
640 GOSUB 1800'    7--------- ГОД
650 GOSUB 2000'    8--------- МЕСЯЦ
660 GOSUB 2200'    9--------- ДЕНЬ
670 Y0 = Y : M0 = M : D0 = D
680 GOSUB 2400'    10-------- К О Л И Ч Е С Т В О
                              ДНЕЙ
690
700
710
720
800 '2---------ВЫБРАННАЯ ДАТА----------
810 '
820 LOCATE0 ,11
830 PRINT"<ВВЕДИТЕ ВЫБРАННУЮ ДАТУ>";
    PRINT:PRINT:PRINT
840 GOSUB 1800'    7---------ГОД
850 GOSUB 2000'    8---------МЕСЯЦ
860 GOSUB 2200'    9---------ДЕНЬ
870 Y1 = Y : M1 = M : D1 = D
880 GOSUB 2400'    10--------КОЛИЧЕСТВО ДНЕЙ
890
900
910
```





```
920
1000 '3-------КОЛИЧЕСТВО ПРОЖИТЫХ ДНЕЙ----
1010 '
1020 N = N1-N0
1030 IF N < 0 THEN RETURN 420
1040 RETURN
1050 '
1200 '4--------ДЕНЬ РОЖДЕНИЯ----------
1210 '
1220 N2 = N0
1230 GOSUB 2600' 11---ДЕНЬ НЕДЕЛИ
1240 D0$ = D$
1250 RETURN
1260 '
1400 '5-------ВЫБРАННЫЙ ДЕНЬ----------
1410 '
1420 N2 = N1
1430 GOSUB 2600' 11---ДЕНЬ НЕДЕЛИ
1440 D1$ = D$
1450 RETURN
1460 '
1600 '6------ВЫВОД РЕЗУЛЬТАТА НА ЭКРАН------
1610 '
1620 SCREEN 7
1630 COLOR 15,1,1
1640 CLS
1650 GOSUB 2800'  12---------ВЫВОД ТЕКСТА
1660 GOSUB 3000'  13---------ИНТЕРВАЛ
1670 GOSUB 3200'  14---------РАМКА
1680 GOSUB 3400'  15---------ГРАФИК
1690 GOSUB 3600'  16---------СООБЩЕНИЕ
1700 RETURN
1710 '
```

```
1800  '7----------ГОД----------
1810  '
1820  V = 0
1830  INPUT"Год (4 цифры)"; Y
1840  IF Y < 1901 OR Y > 1999 THEN V = 1
1850  RETURN
1860  '
2000  '8----------МЕСЯЦ----------
2010  '
2020  V = 0
2030  INPUT"Месяц (2 цифры)"; M
2040  IF M < 1 OR M > 12 THEN V = 1
2050  RETURN
2060  '
2200  '9----------ДЕНЬ----------
2210  '
2220  IF V=1 THEN RETURN
2230  INPUT"День (2 цифры)"; D
2240  RESTORE 10000
2250  FOR I = 1 TO M : READ DD : NEXT
2260  IF M = 2 AND (Y MOD 4) = 0 THEN DD =
      DD + 1
2270  IF D < 1 OR D > DD THEN V = 1
2280  RETURN
2290  '
2400  '10--------КОЛИЧЕСТВО ДНЕЙ-------
2410  '
2420  Y = Y-1901 : N = D-1 : M = M-1
2430  IF M = 0 THEN 2500
2440  RESTORE 10000
2450  FOR I = 1 TO M
2460  READ DD
2470  N = N + DD
```





```
2480  NEXT
2490  IF M > 1 AND ((Y + 1) MOD 4) = 0 THEN
      N= N + 1
2500  IF D = 0 THEN RETURN
2510  FOR I = 1 TO Y
2520  N = N + 365-(((I) MOD 4) = 0)
2530  NEXT
2540  RETURN
2550  '
2600  '11--------ДЕНЬ НЕДЕЛИ----------
2610  '
2620  N2 = N2 MOD 7
2630  RESTORE 10010
2640  FOR I = 0 TO N2
2650  READ D$
2660  NEXT
2670  RETURN
2680  '
2800  '12---------ВЫВОД ТЕКСТА----------
2810  '
2820  LINE(0,156)-(511,191), 14, BF
2830  PRESET(32,160)
2840  PRINT#1,"Дата Вашего рождения
      ____";D0$;
      D0;"/";M0;"/";Y0
2850  PRESET(32,170)
2860  PRINT#1,"Выбранная дата____"
      ;D1$;D1;"/";
      M1;"/";Y1
2870  PRESET(32,180)
2880  PRINT#1,"Количество прожитых дней____";N
2890  RETURN
2900  '
```

```
3000  '13----------ИНТЕРВАЛ-----------
3010  '
3020  FOR I = 0 TO 30
3030  COLOR 15 + 9•(I = 15)
3040  GOSUB 3800'  17----------ДЕНЬ N
3050  PRESET(14 + 16•I,128) : PRINT #1, D$
3060  D$ = STR$(D) : D$ = RIGHT$("0" + RIGHT$
      (D$, LEN(D$)-1),2)
3070  COLOR 15-(I MOD 2)
3080  PRESET(10 + 16•I,136) :PRINT #1, D$
3090  NEXT
3100  RETURN
3110  '
3200  '14---------РАМКА---------
3210  '
3220  LINE(8,10)-(504,129), 15, B
3230  FOR I = 0 TO 31
3240  LINE(8 + 16•I,10)-STEP(0,120)
3250  NEXT
3260  LINE(8,65)-(504,65)
3270  LINE(8,2)-STEP(32,0),3 : PRINT#1,
      "Физический"
3280  LINE(178,2)-STEP(32,0),13 : PRINT#1,
      "Эмоциональный"
3290  LINE(364,2)-STEP(32,0),11 : PRINT#1,
      "Интеллектуальный"
3300  RETURN
3310  '
3400  '15----------ГРАФИК-----------
3410  '
3420  N = N-15
3430  PSET(8,65-50•SIN(2•N•PI/23)),2
3440  FOR I = 1 TO 62
```





```
3450  LINE-(8 + 8•I,65-50•SIN(2•(N + I/2)•PI/23)),2
3460  NEXT
3470  PSET(8,65-50•SIN(2•N•PI/28)), 13
3480  FOR I = 1 TO 62
3490  LINE-(8 + 8•I,65-50•SIN(2•(N + I/2)•PI/28)),13
3500  NEXT
3510  PSET(8,65-50•SIN(2•N•PI/33)), 11
3520  FOR I = 1 TO 62
3530  LINE-(8 + 8•I,65-50•SIN(2•(N + I/2)•PI/33)),11
3540  NEXT
3550  RETURN
3560  '
3600  '16---------СООБЩЕНИЕ---------
3610  '
3620  PRESET(8,200)
3630  PRINT#1,"<Для повторения/выхода,
      нажмите -RETURN-/-ESC->"
3640  RETURN
3650  '
3800  '17---------ДЕНЬ N-----------
3810  '
3820  D$ = MID$("TWTFSSM", 1 + ((I + N1 + 6)
      MOD 7), 1)
3830  D = D1-15 + I
3870  IF D > 0 THEN 3930
3880  M = (M1 + 10) MOD 12
3890  RESTORE 10000
3900  FOR K = 0 TOM : READ DD : NEXT
3910  IF M = 1 AND (Y1 MOD 4) = 0 THEN DD
      =DD + 1
3920  D = DD + D : RETURN
3930  M = M1-1 : RESTORE 10000
3940  FOR K = 0 TOM : READ DD : NEXT
```

```
3950  IF M = 1 AND (Y1 MOD 4) = 0 THEN DD
      =DD + 1
3960  IF D <= DD THEN RETURN
3970  D = D-DD
3980  RETURN
3990  '
10000 DATA 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
      31
10010 DATA Понедельник, Вторник, Среда,
      Четверг, Пятница, Суббота, Воскресенье
```



# Структура памяти

Глава 13

Читатель, программирующий только на MSX Бейсике, найдет большую часть необходимой ему информации в предыдущих главах.
Следующие главы содержат дополнительную информацию для тех, кто заинтересован в написании подпрограмм или даже целых программ на машинном языке. Здесь может также быть найдена важная информация о работе компьютера.

## 13-1. Постоянное запоминающее устройство (ПЗУ) и оперативное запоминающее устройство (ОЗУ)

Память разделена на группы байтов, каждый из которых имеет адрес, закодированный парой байтов. Таким образом, любые данные, записанные в памяти, могут быть найдены, если их адреса известны. Поскольку адреса требуют два байта, непосредственно адресуемая память охватывает диапазон от 0000(H) до FFFF(H) или 65535(D).
Функция PEEK используется для чтения данных по указанному адресу; оператор POKE записывает байт данных в память.
Пример:

X = PEEK(&HF676) помещает в X значение, размещенное по адресу F676(H)
POKE &HB000, 24 записывает значение 24 по адресу B000(H)

Считываемые и записываемые значения должны лежать в пределах от 0 до 255, поскольку они должны умещаться в один байт.

Адресное пространство в пределах от 0000(H) до 7FFF(H),

занято ПЗУ, ОЗУ занимает адресное пространство с 8000(H) по FFFF(H).

ПЗУ допускает только чтение (POKE не действует в ПЗУ). ОЗУ допускает как чтение (PEEK), так и запись (POKE).

### 13-1-1. ПЗУ

ПЗУ содержит те программы и данные, которые "заложены" в компьютер при изготовлении. Вот почему компьютер всегда выводит сообщения при включении и способен понимать программу на MSX Бейсике.

Когда выполняются операторы программы, компьютер проверяет синтаксис, сравнивая составляющие оператор символы с данными, записанными в ПЗУ. Если что-то не так, он выводит сообщение об ошибке, текст которого также находится в ПЗУ. Если оператор приемлем, то выполняется переход к подпрограмме на машинном языке, находящейся в ПЗУ. Такие подпрограммы фактически выполняют работу, требуемую операторами MSX Бейсика. Когда оператор выполнен, компьютер переходит к следующему оператору или возвращается в командный режим. Различные подпрограммы, выполняющие эту работу, составляют ИНТЕРПРЕТАТОР, который переводит один за другим операторы MSX Бейсика в программу на машинном языке, т.е. на единственном языке, который понимает компьютер.

В командном режиме другие подпрограммы из ПЗУ автоматически управляют клавиатурой и экраном. Они, например, выводят на экран то, что вводится с клавиатуры. Все эти подпрограммы составляют ЭКРАННЫЙ РЕДАКТОР MSX Бейсика.

При включении компьютера отмечается небольшая пауза, имеющая своей причиной различные инициализации (SCREEN

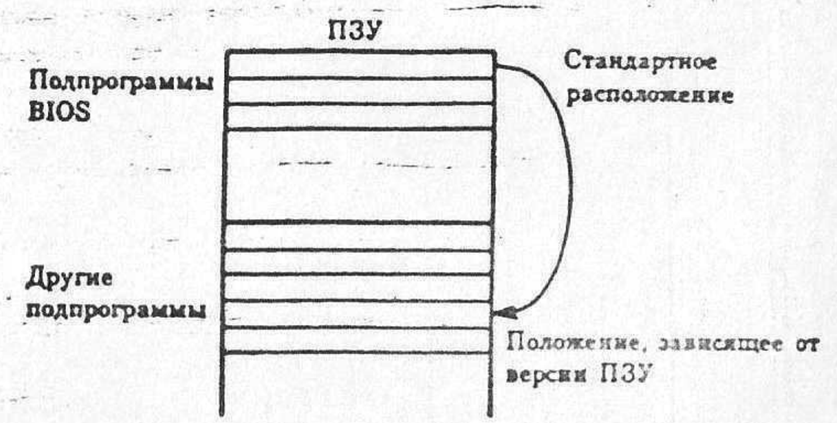




режим, WIDTH, COLOR...). Эти подпрограммы ИНИЦИАЛИЗАЦИИ посылают многочисленные данные в рабочую область ОЗУ, что будет обсуждено позже.

ПЗУ в основном разделено на две части:
(1) Подпрограммы BIOS
(2) Другие подпрограммы.

Подпрограммы BIOS осуществляют переход к другим подпрограммам. Они напоминают последовательность GOSUB, которую можно увидеть на первом уровне хорошо структурированной программы на MSX Бейсике. Именно подпрограммы BIOS составляют неотъемлемую часть стандарта MSX. Они всегда расположены по одним и тем же адресам ПЗУ независимо от версии MSX Бейсика на Вашем компьютере. Они осуществляют переход к другим подпрограммам, чье положение может быть изменено. Следующий рисунок иллюстрирует, как это все устроено.




Полный перечень подпрограмм BIOS находится в Приложении Е. Способ их использования будет объяснен в дальнейшем.

### 13-1-2. ОЗУ

В противоположность ПЗУ, ОЗУ стирается при выключении компьютера. Оно делится на несколько областей:

(1) Таблица программных команд (PIT - Program Instruction Table)
Здесь записан текст программы на MSX Бейсике. PIT начинается с 8000(H).
Как будет дальше показано, ее положение можно изменять.

(2) Таблица переменных (VT)

VT содержит все переменные, создаваемые в ходе выполнения программы.

(3) Стек

Стек содержит всю информацию, необходимую для выполнения программы. Например, именно здесь хранится адрес в PIT, соответствующий следующему выполняемому оператору.

(4) Зарезервированное строковое пространство

(5) Блоки управления файлами

Когда задается максимальное количество одновременно открытых файлов (MAXFILES =), для каждого файла автоматически резервируется 267-байтное пространство для осуществления обмена данными с файловыми устройствами.

(6) Рабочая область

Здесь записываются переменные, используемые подпрограммами ПЗУ. Эта область содержит большое количество таких данных, как позиция курсора, цвет текста, текст функциональных





клавиш и др., инициализируемых при включении компьютера. Там же находятся некоторые подпрограммы. Эти переменные перечислены в Приложении Ж. Во время выполнения программы эти переменные постоянно используются и изменяются подпрограммами ПЗУ, вызываемыми интерпретатором. Это предполагает, что переменные находятся по определенному адресу, известному подпрограммам ПЗУ. Адрес, отмечающий начало рабочей области, указан в самой этой области словом HIMEM (2 байта), расположенным по адресу FC4A(H). Следующий оператор возвращает его:

```
X = HEX$(PEEK(&HFC4A) + 256*PEEK(&HFC4B))
```

Помните, что значения, занимающие два байта, всегда записываются таким образом, что первым оказывается м л а д ш и й байт.

Адрес, отмечающий конец PIT (который оказывается также адресом, отмечающим начало VT, которая формируется в зависимости от длины программы), ЗАПИСАН в слове VARTAB по адресу F6C2(H).

Поскольку рабочая область расположена в ОЗУ, ее переменные могут изменяться операторами POKE. Но это следует делать только в том случае, если Вы знаете, что Вы делаете.

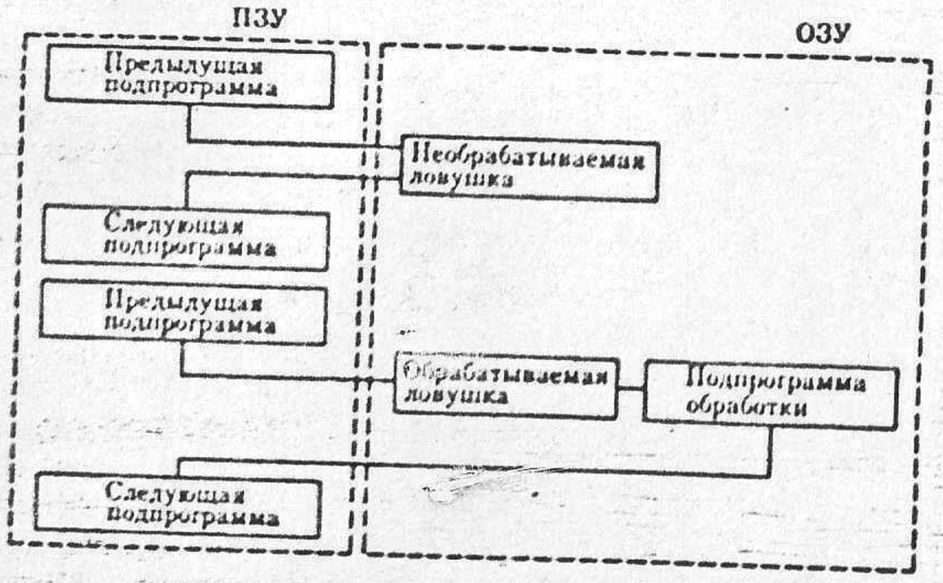
(7) Ловушки

Ловушки являются частью рабочей области и позволяют осуществлять целый набор трюков на машинном языке. Помните, что программы, делающие всю "черную" работу, находятся в ПЗУ, поэтому изменить их невозможно. Однако при входе в некоторые важные подпрограммы ПЗУ предусмотрены

"перехваты"-ловушки. Обычно подпрограмма-ловушка содержит только RETURN (это слово используется для ясности: на самом деле - это команды машинного языка). Вы можете заменить RETURN переходом к подпрограмме, которая написана на машинном языке. Тем самым Вы косвенно измените ПЗУ; о ловушке говорится, что она о б р а б а т ы в а е т с я.

Обработка ловушек позволяет в известном смысле вводить новые операторы. Например, для компьютера с MSX DISK Бейсиком предусмотрена обработка определенного количества ловушек, то есть подпрограммы, на которые они осуществляют переход, загрузятся во время инициализации рабочей области.

Следующая схема показывает, как работают ловушки.








## 13-2. Управление ОЗУ.

### 13-2-1. Оператор CLEAR

Оператор CLEAR имеет два аргумента:

Первый аргумент указывает количество байтов, зарезервированных под строковое пространство. Это количество изменяется от 0 до наличного свободного пространства, и по умолчанию равно 200.

Второй переопределяет верхний предел пространства, используемого MSX Бейсиком. Тем самым он резервирует определенное количество байтов между блоками управления файлами (FCB) и началом рабочей области. Это создает резервную память, где могут храниться, например, подпрограммы на машинном языке. Максимальным значением этого числа, конечно, может быть адрес, где начинается рабочая область. Его минимальное значение ограничивается необходимым 145-байтовым пространством между концом PIT и вершиной стека.

Второй аргумент CLEAR (записанный в слове HIMEM), должен быть больше, чем величина:

**267*(MAXFIL + 1) + VARTAB + 130 + зарезервированное строковое пространство**

Оба аргумента CLEAR могут опускаться.

Независимо от формы этого оператора, он уничтожает все переменные, короче говоря, все определенное с помощью DEF (DEFFN, DEFINT, DEFUSR и т.д.), а также уничтожает стек. Сохранится только текст программы.

Пример:

CLEAR 300, &B000 резервирует 300 байтов под строковые данные и защищает область между B000(H) и началом рабочей области

### 13-2-2. Функция FRE(0)

Функция FRE(0) возвращает количество байтов, оставшихся для PIT, VT и стека.

Верхняя часть памяти занята рабочей областью и ловушками. Нижняя часть памяти занята текстом программы (PIT) и таблицей переменных (VT).
Наконец, Вы доходите до свободного пространства, длина которого дается FRE(0). Между концом этого свободного пространства и началом рабочей области, находятся стек, строковое пространство и, наконец, блоки управления файлами.

Длина PIT зависит от длины текста программы.

Непосредственно следующая за ней VT начинается по адресу, указанному в слове VARTAB (F6C2(H)). Ее длина зависит от количества определенных переменных (скалярных и массивов) и их точности. Вершина стека указывается в слове STKTOP (F674(H)). Его позиция зависит от длины строкового пространства и блоков управления файлами, а также от второго аргумента CLEAR, если этот оператор был выполнен.

Длина пространства, зарезервированного для строк (по умолчанию 200) определяется первым аргументом CLEAR и может быть получена с помощью функции FRE("") (см. ниже).

Длина блоков управления файлами оценивается как 267 + 267*число файлов, которые могут быть открыты (файл 0 всегда открыт). Число файлов, которые могут быть открыты,





указано в байте MAXFIL (F85F(H)) и может быть изменено оператором MAXFILES.

### 13-2-3. Функция FRE("")

Эта функция возвращает количество байтов, имеющихся в пространстве, зарезервированном под строки. Если под строки зарезервирован большой объем и определено много символьных переменных, время выполнения этой функции может составить несколько минут. Строки хранятся в этом пространстве в том порядке, в каком они были определены. Если Вы определили, A$ = "ABCD" + "EF" в начале программы, а затем переопределяете эту переменную как A$ = "X" + "Y", Вы создаете пустое 4-байтовое пространство между этой переменной и следующей. FRE("") перестраивает положение переменных для удаления всего неиспользованного пространства (СБОРКА МУСОРА), и это является причиной задержки.То же самое имеет место, когда большая часть строкового пространства использована.

Во избежание этой "уборки", определяйте все переменные в начале программы и используйте следующие функции, непосредственно влияющие на указатели:

(1) MID$() =, MID$, LEFT$ и RIGHT$

(2) SWAP

Оператор SWAP A$, B$ просто включает указатели этих переменных. Он работает также с числовыми переменными.

В вышеприведенном примере A$ могла быть определена в начале программы:

```
A$ = STRING$(6,"")
```

Во время присвоения значений этой переменной, с успехом может быть проделано следующее:

```
A$ = "ABCDEF"          не изменяет длину
MID$(A$, 1, 2) = "XY
```

Для использования этой переменной в выражении четырьмя последними символами следует пренебречь:

```
LEFT$(A$, 2)
```

### 13-2-4. Оператор MAXFILES =

Определяет максимальное количество файлов, которые могут быть открыты одновременно. Файл 0 всегда открыт, и по умолчанию MAXFILES = 1.

Этот оператор резервирует в ОЗУ пространство, которое равно 267 + 267*число файлов (блоки управления файлами). Число файлов помещается в слово MAXFIL (F85C(H)).

### 13-2-5. Функция VARPTR

Эта функция имеет две формы:

(1) X=VARPTR(A) возвращает адрес первого байта, содержащего значение A.
X=VARPTR(A$) возвращает адрес байта, содержащего длину строки A$.
(2) X=VARPTR(N) возвращает адрес блока управления файлом с номером N.





### 13-2-6. Операторы BSAVE и BLOAD

BSAVE позволяет сохранить часть памяти (ОЗУ или ПЗУ). Это может быть программа на MSX Бейсике, данные, подпрограммы ПЗУ, рабочая область... Этим оператором может быть сохранено все между 0000(H) и FFFF(H). Если вывод - не на дисковод A, то следует указать имя устройства. Затем укажите имя файла, начальный адрес сохраняемых данных, конечный адрес и, возможно, адрес выполнения. Если последний пропущен, адрес, отмечающий начало данных, будет автоматически рассматриваться как адрес выполнения при загрузке этих данных (эта часть информации полезна только в том случае, если сохраняются подпрограммы на машинном языке).

BLOAD позволяет загрузить данные, сохраненные BSAVE. Должно быть указано имя устройства, если это не дисковод A, и имя файла. Может присутствовать R, если данные относятся к программе на машинном языке. В этом случае программа запустится после загрузки. Данные загружаются в то место, которое они занимали перед сохранением, если не указан адрес смещения. В этом случае данное число добавляется к начальному, конечному и адресу смещения, заданным в оператора BSAVE.

Обратите внимание, что BLOAD может использоваться в программе, не вызывая NEW или возврата в командный режим. Этот оператор может использоваться для загрузки частей программы на MSX Бейсике.

Примеры:

```
BSAVE"DATA", &HE000, &HEFFF, &HE100
```

сохраняет модуль, размещенный между E000(H) и F000(H); выполнение должно начаться по адресу E100(H).

BLOAD"DATA", R

загружает модуль DATA и начинает выполнение по адресу E100(H).

То, как работают эти два оператора, и их полезность станут очевидными после детального рассмотрения структуры PIT и при создании программ на машинном языке.

## 13-3. PIT (Program Instruction Table - Таблица программных команд).

### 13-3-1. Адрес PIT

Обычно таблица PIT начинается по адресу 8000(H). Конечно, ее можно сдвинуть, изменив слово TXTTAB в рабочей области. Например, для помещения PIT в A000(H), достаточно выполнить следующую программу:
Адрес A001(H), находящийся в слове TXTTAB (F676(H)),

```
10  POKE &HF676, 1
20  POKE &HF677, &HA0
30  POKE &HA000, 0
40  NEW
```

определяет место, где начнется текст программы. Первый байт PIT (A000(H)) должен быть нулевым. Последняя строка стирает программу.
Очевидно, что выполнение этой программы уменьшает значение FRE(0) на 2000(H) байтов.
Область, расположенная между 8000(H) и началом PIT, будет защищена от вторжения программ на MSX Бейсике, и следовательно, безопасна для размещения программ на машинном языке.





### 13-3-2. Структура PIT

Если Вы выполнили вышеуказанную программу, то теперь нажмите кнопку сброса RESET.

Все строки программы на MSX Бейсике начинаются с двухбайтного указателя. За этим указателем идут два байта, содержащие номер строки. Затем идет текст строки с последующим нулевым байтом. За последней строкой идут два дополнительных нулевых байта, адрес которых находится в адресе последней строки.

Ключевые слова и цифры записываются во внутреннем коде (один или два байта на ключевое слово). Для остального текста используется символьный код ASCII.

Введите следующую короткую программу:

```
10  R = 5
20  END
```

Теперь прочитайте, что же реально содержится в PIT, используя последовательность следующих операторов:

```
?HEX$(PEEK(A))
```

адрес A изменяется от &H8000 до &H8010.

Вы обнаружите:

```
8000    0
```

Первый байт PIT всегда нулевой.

8001 9
8002 80

Первый указатель строки, указывающий, что указатель следующей строки находится по адресу 8009(H).

8003 A
8004 0

Номер первой строки 000A(H) = 10

8005 41

код ASCII для A

8006 EF

Внутренний код знака равенства

8007 16

Внутренний код цифры 5

8008 0

Конец первой строки

8009 F
800A 80

Указатель второй строки, означающий, что указатель следующей строки находится в 800F(H)

800B 14
800C 0





Номер второй строки 0014(H) = 20

800D 81

Внутренний код оператора END

800E 0

Конец второй строки

800F 0
8010 0

Конец программы

Конечно, можно изменить программу с помощью операторов POKE. Попробуйте теперь сделать это:

```
POKE &H8005, &H42
POKE &H8007, &H17
```

Сделайте LIST, и появится новая программа:

```
10  2A = 6
20  END
```

Не пытайтесь изменять с помощью POKE длину строки или путать указатели: результат будет только катастрофическим!

Сохранение и загрузка модулей из PIT с помощью BSAVE и BLOAD (конечно, без R) - это самое лучшее использование полученных знаний.

Вы, конечно, можете представить себе и короткую программу, отменяющую действие таких операторов, как LIST, и загружающую программный модуль, который Вы хотите защитить от копирования.

Примечание:

Когда выполняется LIST или LLIST, все внутренние коды PIT переводятся в формат ASCII, чтобы сделать программу читаемой. То же самое происходит, когда программа сохраняется в формате ASCII. Она может быть прочитана как обычный файл данных оператором LINE INPUT. Внутренние коды клавиатуры перечислены в Приложении Г.

### 13-4. VT (Variables Table - Таблица переменных)

Переменные и массивы хранятся в порядке их создания. Массивы хранятся выше переменных. Это значит, что каждый раз, когда записывается скалярная переменная, все массивы сдвигаются вверх, чтобы высвободить пространство. Это может значительно замедлить выполнение программы. Во избежание этого, задавайте все массивы переменных в начале программы (DIM).

Выполните следующую программу:

```
10  DIM V(3000)
20  TIME = 0
30  A = 1
40  PRINT TIME
```





Теперь замените строку 10 следующей, и еще раз выполните программу.

```
10  DIM V(3000), A
```

Значение скалярной (числовой) переменной кодируется в 2, 4 или 8 байтах в зависимости от типа точности. Этим байтам предшествует байт VALTYPE плюс 2 байта, содержащих имя переменной. Следующие два байта указывают, на сколько байтов дальше находится следующий массив. Это число легко найти: это 3 плюс количество элементов, умноженное на VALTYPE.

Функция VARPTR возвращает текущий адрес первого байта области, содержащей либо само значение переменной (числовой), либо его длину (для символьных строк).

### 13-5. Стек

Стек используется как программой на MSX Бейсике, так и подпрограммами на машинном языке (см. ниже). Здесь компьютер хранит информацию, необходимую для того, чтобы найти следующий оператор для выполнения.

Ниже описана информация, заложенная в стек при выполнении цикла FOR/NEXT:

| | |
|---|---|
| FOR код оператора | 1 байт |
| Адрес счетчика (в VT) | 2 байта |
| Знак приращения | 1 байт |
| Значение приращения (STEP) | 4 байта |
| Предел счетчика (TO) | 4 байта |
| Номер строки FOR | 2 байта |
| Адрес FOR в PIT | 2 байта |

## 13-6. Строковое пространство

Указатели символьных переменных хранятся в VT, а не в строковом пространстве. Их длина - 6 байтов: первый - VALTYPE (один байт = 3), затем имя переменной (2 байта), и наконец байт, задающий длину строки, возвращаемую функцией VARPTR. Таким образом, обе функции

**LEN(A$) и PEEK(VARPTR(A$))**

возвращают одно и то же значение.
Следующие два байта указывают адрес первого байта строки. Но где расположена строка? Необязательно в пространстве, зарезервированном под строки. Если символьная переменная создается явным присваиванием символьной константы, указатель задает адрес этой константы в PIT. Переменная лишь потом пересылается в зарезервированное для нее пространство. Следующие примеры поясняют сказанное:

Пример 1:

```
10  A$ = "ABCD"
```

Выполните эту программу, а затем проделайте:

```
?HEX$(VARPTR(A$))
```

Вы получите 8014(H). Это адрес байта, содержащего длину.

Затем проделайте

```
?HEX$(PEEK(AD))
```

с AD, изменяющимся от 8011(H) до 8016(H). Вы получите:





| | | |
|---|---|---|
| 8014 | 3 | VALTYPE |
| 8015 | 41 | код ASCII для A |
| 8013 | 0 | второго символа нет |
| 8014 | 4 | Длина строки |
| 8015 | 9 | Адрес |
| 8016 | 80 | Адрес |

Строка помещается по адресу 8009(H). Вы легко можете убедиться, что этот адрес действительно является адресом первого символа константы "ABCD" в PIT.

Пример 2:

Измените строку 10 следующим образом, и запустите программу

```
10  A$ ="ABCD"+""
```

Повторите вышеуказанные шаги. Вы получите:

| | | |
|---|---|---|
| 8014 | 3 | VALTYPE |
| 8015 | 41 | код ASCII для A |
| 8016 | 0 | второго символа нет |
| 8017 | 4 | Длина |
| 8018 | 65 | Новый адрес |
| 8019 | F1 | Новый адрес |

Операция со строкой явилась причиной пересылки ее по адресу F165(H) (без изменения). Вы можете убедиться в этом, используя в этой области операторы PEEK.

Функция FRE("") возвратила бы 200 в первом случае, и возвращает 196 во втором случае.

## 13-7. Рабочая область

Эта область содержит короткие подпрограммы, переменные и ловушки, используемые MSX Бейсиком во время выполнения операторов программы. Все они перечислены в Приложении Ж. Поскольку она расположена в ОЗУ, можно не только читать ее (PEEK), но также вводить в нее значения (POKE). Предыдущие примеры уже показали, как используются переменные этой области (чтение слов VARTAB, STKTOP, HIMEM, MAXFIL; модификация слова TXTEND для переноса PIT).

Другие примеры:

(1) Прерывание программы

Если байт ENSTOP отличен от 0, всегда можно прервать программу, даже когда нажатие CTRL + STOP обрабатывается. Чтобы сделать это возможным, выполните

```
POKE &HFBB0, 1
```

в начале программы или в командном режиме. Для прерывания программы достаточно одновременно нажать следующие клавиши:

CTRL + GRAPH + РУС + SHIFT

F6 может иногда потребоваться для восстановления цветов и должна быть дополнена оператором SCREEN 0 или 1.

(2) Положение графического курсора

Позиция текстового курсора дается функциями POS и CSRLIN. Эти функции не возвращают положение графического курсора. Для определения этого положения, достаточно прочитать слова GRPACX и GRPACY:





```
X = PEEK(&HFCB7) + 256*PEEK(&HFCB8)
Y = PEEK(&HFCB9) + 256*PEEK(&HFCBA)
```

(3) Считывание кодов нажатых клавиш

Это может быть сделано посредством INPUT или INKEY$. К сожалению, следующие клавиши нечитаемы, поскольку нет соответствующего им кода ASCII:

CAPS  GRAPH  РУС
SHIFT  CTRL  STOP

Для чтения клавиши достаточно прочитать буфер NEWKEY (11 байтов) по адресу FBE5(H).

KBDMTRX

```
10  FOR Y = 0 TO 10
20  Z = PEEK(&HFBE5 + Y)
30  IF Z = 255 THEN 80
40  PRINT"Y ="; Y
50  Z$ = RIGHT$("00000000" + BIN$(Z), 8)
60  PRINT"X ="; 8 - INSTR(Z$, "0")
70  PRINT
80  NEXT : GOTO 10
```

Эта программа сканирует все клавиши и возвращает позицию нажатой клавиши (X,Y) в матрице клавиатуры. 11 значений, записанных в буфере NEWKEY, соответствуют 11 строкам матрицы клавиатуры. Если не нажата ни одна клавиша, все эти значения приравниваются 255 = 11111111(B). Когда клавиша нажата, считанное на этой строке значение отличается от 255. Фактически, соответствующий колонке бит просто сбрасывается на 0.

Матрица клавиатуры

| | X7 | X6 | X5 | X4 | X3 | X2 | X1 | X0 | NEWKEY |
|---|---|---|---|---|---|---|---|---|---|
| Y0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | FBE5 |
| Y1 | : | ] | [ | \ | = | - | 9 | 8 | FBE6 |
| Y2 | B | A | | / | . | , | ' | ; | FBE7 |
| Y3 | J | I | H | G | F | E | D | C | FBE8 |
| Y4 | R | Q | P | O | N | M | L | K | FBE9 |
| Y5 | Z | Y | X | W | V | U | T | S | FBEA |
| Y6 | F3 | F2 | F1 | CODE | CAPS | GRAPH | CTRL | SHIFT | FBEB |
| Y7 | RETURN | SELECT | BS | STOP | TAB | ESC | F5 | F4 | FBEC |
| Y8 | → | ↓ | ↑ | ← | DEL | INS | HOME | ESPACE | FBED |
| Y9 | 4 | 3 | 2 | 1 | 0 | / | - | + | FBEE |
| Y10 | . | , | * | 9 | 8 | 7 | 6 | 5 | FBEF |
| при нажатии | 7F | BF | DF | EF | F7 | FB | FD | FE | |

Последние две строки соответствуют цифровой (правой) зоне клавиатуры.

Ниже приведена программа, останавливаемая нажатием SHIFT:

```
10  Z = PEEK(&HFBEB)
20  IF Z < > &B11111110 THEN 10
```



# Структура видеопамяти

Глава 14

Компьютер MSX имеет память, именуемую VRAM (видеопамятью) для хранения данных отображения. Доступ к ней осуществляется функцией VPEEK и оператором VPOKE. За исключением некоторых случаев все операторы, изменяющие что-либо на экране, выполняют обмен данными с видеопамятью. Видеопроцессор (по терминологии MSX - VDP) постоянно работает с видеопамятью и интерпретирует находящиеся в ней данные в соответствии с заданным режимом SCREEN.

Видеопамять разделена на несколько специализированных таблиц, у которых имя, длина и функция зависят от режима SCREEN. Это деление памяти автоматически выполняется оператором SCREEN, который также инициализирует таблицы.

Прежде чем пытаться непосредственно модифицировать экран с помощью операторов VPOKE, следует хорошо разобраться в назначении различных таблиц.

Примечание:

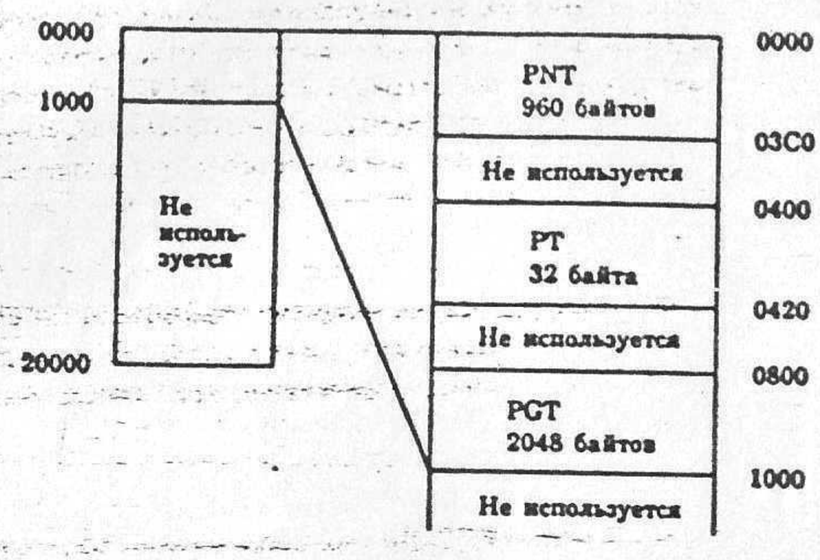
Названия таблиц сокращены следующим образом:

- PNT Таблица названий образов
- PGT Таблица генератора образов
- CT Таблица цветов
- SGT Таблица генератора спрайтов
- SAT Таблица атрибутов спрайтов
- PT Таблица палитры

## 14-1. Режим SCREEN 0

### 14-1-1. Структура видеопамяти в SCREEN 0

(1) В 40-символьном режиме видеопамять делится на три таблицы следующим образом.

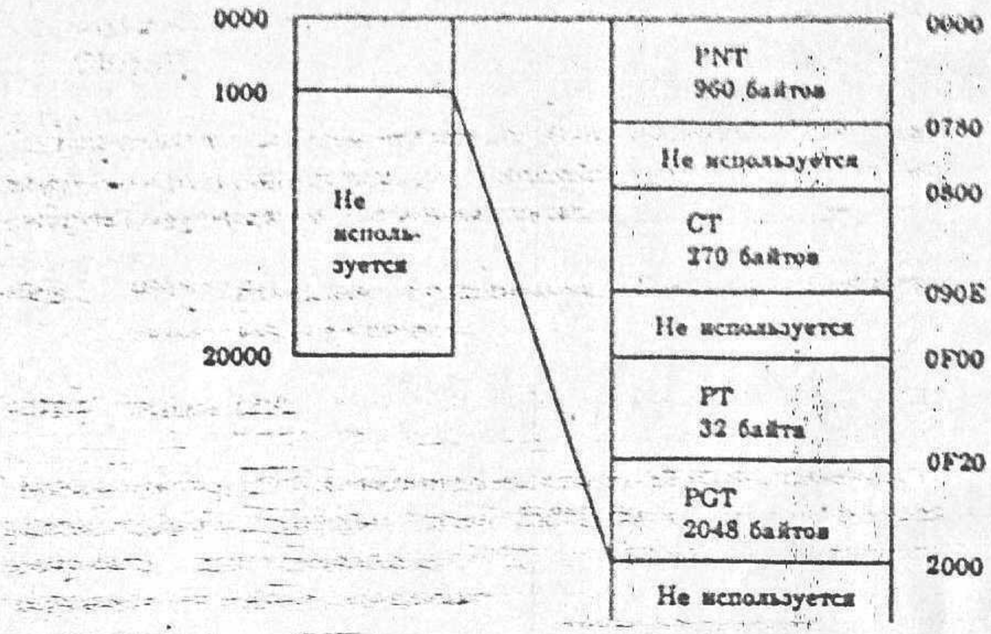



Оператор SCREEN 0 инициализирует PNT, 960 раз вставляя 32 (код пробела). Таблица форм PGT получает конфигурацию 256 имеющихся символов, каждый из которых закодирован в 8 байтах(256•8 = 2048) и записан в порядке кодовой таблицы ASCII.





(2) В 80-символьном режиме (имеющемся только на компьютерах MSX-2) видеопамять имеет следующую структуру:




PNT имеет 1920 (80*24), инициализированных с пробелом. PGT та же, что и в 40-символьном режиме. Таблица цветов (CT) не используется.

### 14-1-2. Соотношение отображения и таблиц в SCREEN 0.

Цвета текста и фона определяются оператором COLOR. Этот вид информации не содержится в видеопамяти.

Вывод символов на экран происходит следующим образом: пусть AC представляет код ASCII выводимого символа, и (L, C) - координаты места вывода на экран. AC помещается в PNT по адресу 40•L + C (80•L + C в 80-символьном режиме). Видеопроцессор ищет форму в PGT по адресу &H800 + 8•AC (&H1000 + 8•AC в 80-символьном режиме). Если PGT не модифицировалась, Вы имеете форму символа, код ASCII которого равен AC.

Пример: вывести "A" (код ASCII 65) на строке 5, колонка 10.

```
10  SCREEN 0 : WIDTH 40
20  VPOKE 40*5 + 10, 65
```

Форма, соответствующая заданному в AC коду ASCII, может изменяться. Достаточно изменить 8 байтов PGT, начиная с адреса &H800 + 8*AC (&H1000 + 8*AC в 80-символьном режиме).

Пример: заменить символ "A" (код 65) маленьким квадратом

ASQUARE

```
10  SCREEN 0 : WIDTH 40
20  FOR I = 0 TO 7 : READ A$
30  VPOKE &H800 + 8*65 + I, VAL("&B" + A$)
40  NEXT
50  '
100 DATA 11110000
110 DATA 10010000
120 DATA 10010000
130 DATA 10010000
140 DATA 10010000
150 DATA 10010000
160 DATA 11110000
170 DATA 00000000
```





Выполните эту программу, а затем попробуйте нажать "A" (заглавную) или просто распечатать программу. Кстати, новый символ может быть выведен командой ?CHR$(65), а нормальный вывод восстанавливается при выполнении SCREEN 0.

Сетчатая форма фона может быть получена изменением символа пробела (код 32).

SPSQUARE

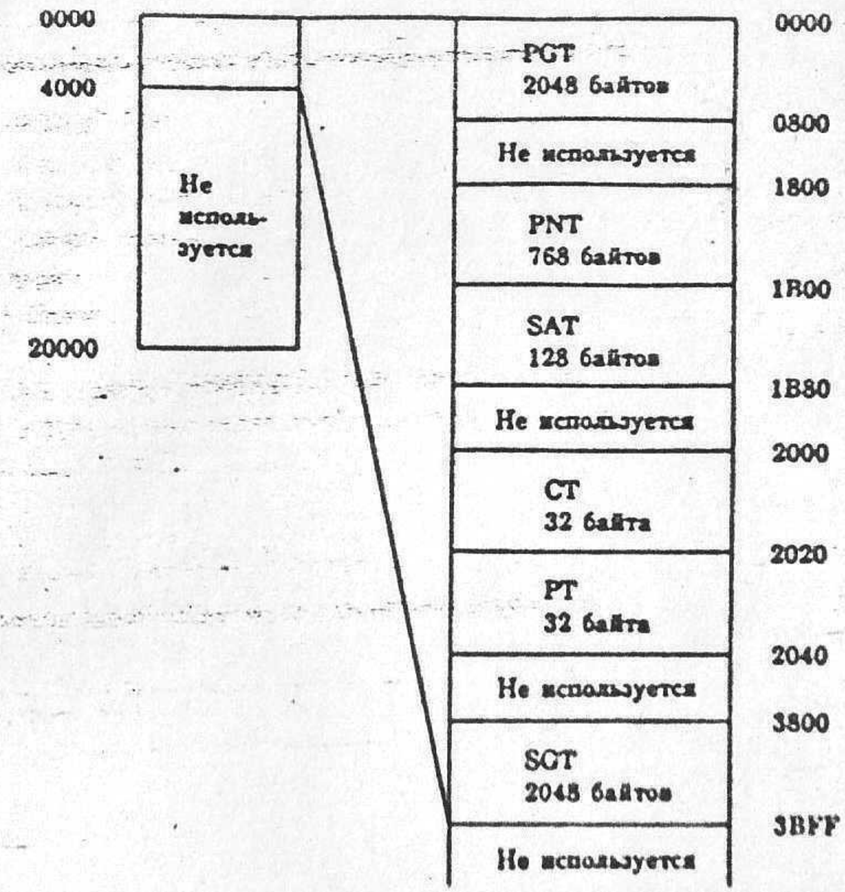
```
10  SCREEN 0 : WIDTH 40
20  VPOKE &H800 + 32*8, 255
30  FOR I = 1 TO 7
40  VPOKE &H800 + 32*8 + I, 128
50  NEXT
```

Сетка заполняет ранее пустой экран.

## 14-2. Режим SCREEN 1

### 14-2-1. Структура видеопамяти в SCREEN 1

Как показано ниже, в SCREEN 1 используется 6 таблиц. PNT и PGT инициализируются так же, как и в SCREEN 0. PNT содержит только 32*24 = 768 байтов. 32-байтная таблица цветов (CT) инициализируется с цветами текста и фона, закодированными соответственно в 4 старших и 4 младших битах каждого байта. Если выполнен COLOR 15, 4, 7, каждый из 32 байтов будет содержать значение F4(H) (F(H) = 15 = белый; 4 = голубой).




### 14-2-2. Соотношение отображения и таблиц в SCREEN 1

Инициализация PNT и PGT та же, что и в SCREEN 0. Достаточно лишь принять в расчет WIDTH (максимум 32) и адаптировать вышеприведенные примеры.





CT позволяет, вообще говоря, выводить текст нескольких цветов. Однако каждый из 32 байтов CT действует на группу из 8 символов. Пример: вывод 8 первых заглавных букв алфавита в "инвертированном" цвете.

```
10  SCREEN 1 : COLOR 15, 4, 7
20  VPOKE &H2008, &H4F
```

Адрес таблицы цветов, использованный здесь (&H2000 + 8), связан с символами, имеющими коды между 64 и 71. Это символы @ABCDEFG. Если Вы хотите напечатать в инвертированном цвете H, восьмую букву алфавита, адрес &H2009 должен быть загружен значением &H4F. Это приведет к выводу в инвертированном цвете 8 символов HIJKLMNO. Вывод в определенном цвете только жестко заданных символов - довольно причудливый трюк. Для изменения цвета курсора (код 255), загрузите адрес &H201F.

Обратите внимание, что CT в SCREEN 1 снова инициализуется оператором COLOR. Кроме того, цвет границы записывается не в видеопамяти, а непосредственно в регистр VDP.

## 14-3. Режим SCREEN 2

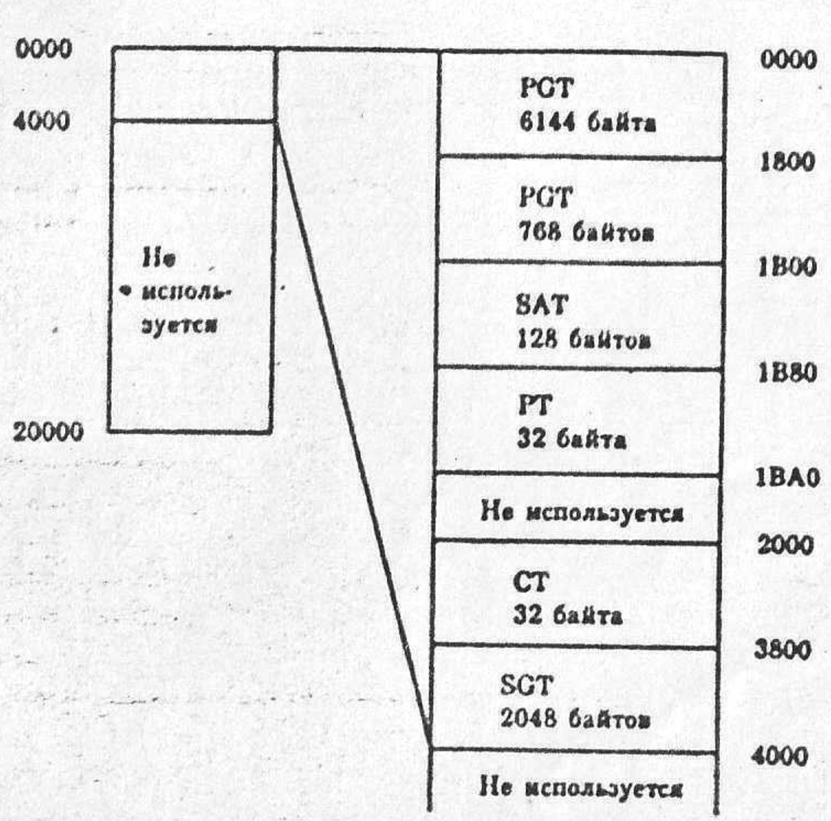
### 14-3-1. Структура видеопамяти в SCREEN 2

32*24 = 768 различных "символов" могут выводиться одновременно. Поскольку один байт может представлять только 256 различных значений, PNT делится на 3 блока по 256 байтов (256*3 =768). PGT должна содержать 768 "символов", закодированных в 8 байтах каждый; следовательно, ее длина будет равна 768*8 = 6144 байтам.

Экран разделен на 192 строки из 32 сегментов, для которых может быть определен цвет изображения и фона. **192*32 = 6144** байта - необходимый объем видеопамяти для CT.

PNT разделена на три 256-байтных блока (**3*256 = 768**). Каждый блок инициализируется со значениями от **0** до **255**, записанными в возрастающем порядке.

Все байты таблицы форм при инициализации сбрасываются на **0**.
Цвет текста закодирован в 4 старших битах байтов CT, а цвет фона в 4 младших. Цвета остаются теми же, что существовали до выполнения SCREEN 2, или же
переопределяются посредством **COLOR** с последующим **CLS**.

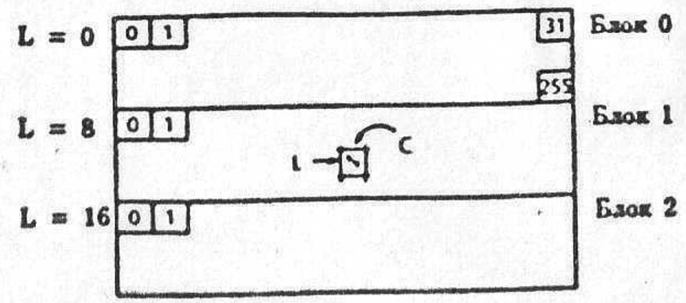







## 14-3-2. Соотношение отображения и таблиц в SCREEN 2

Экран может рассматриваться в качестве разделенного на 24 строки из 32 квадратов, как в SCREEN 1; всего 768 квадратов. Они объединены в три окна из 8 строк каждое. Квадраты в каждом из окон пронумерованы от 0 до 255. Это соответствует делению PNT на три блока по 256 байтов, пронумерованных от 0 до 255. Значения, загруженные в эти байты при инициализации, соответствуют номерам байтов, но это может быть изменено.
Как видеопроцессор (VDP) находит символы для вывода на экран (общий случай):




квадрат (C,L)

Посмотреть PNT по адресу
(L\8)*256 + (L MOD 8)*32 + C
Найти N

Посмотреть PGT по адресу
(L\8)*2048 + N*8 (L\8)*2048 + N*8 + 7
Найти "символ" для выведения в квадрате (C,L)

Ниже следует объяснение того, откуда видеопроцессор знает, что должно быть выведено в квадрате C, строка L.

Пример: L = 10, C = 5

Номер квадрата задается следующей формулой:

(L MOD 8)*32 + C (= 64 + 5 = 69)

Номер блока таблицы названий задается формулой

L\8 (= 1)

Видеопроцессор (VDP) опрашивает байт номер 69 во втором блоке (1) PNT и находит значение N (N = 69, если таблица названий не изменялась). Затем, отыскивается PGT, и в ней - 8 байтов, соответствующих выводимому образу. Таблица образов также разделена на три блока по 2048 байтов каждый. Следовательно, второй блок этой таблицы начинается по следующему адресу:

2048 = 800(H)

Первый из 8 байтов формы, на который указывает N во втором блоке, размещается в ячейке

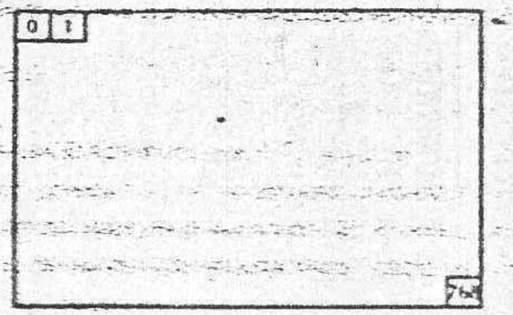
&H800 + N*8

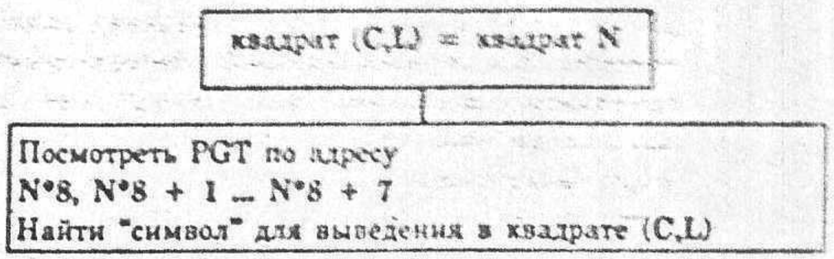
Эти 8 байтов, называемые "символами", будут выведены в квадрате (C,L). Обычно PNT не изменяется после инициализации. Следовательно, можно рассматривать структуру экрана, не принимая во внимание PNT. Можно даже забыть о делении экрана на три окна. Достаточно пронумеровать квадраты экрана от 0 до 767.

Как видеопроцессор находит "символы" для вывода (при условии, что PNT не изменялась)








То, что отображено в квадрате N, соответствует 8 байтам, начинающимся по адресу N*8 в PGT.

Цвета каждой строки этого квадрата возвращаются 8 байтами CT, начиная с:

&H2000 + N*8

Последняя деталь: в SCREEN 2 используется система координат (X,Y), в которой X изменяется от 0 до 255, а Y от 0 до 191. Следующий пример иллюстрирует, как эта система координат соотносится с делением на квадраты, представленным выше.

Пример: Включение белой точки в координате (X,Y) = (100,75) на черном фоне.

Загрузить байт PGT, соответствующий данному сегменту, нужным значением, и байт CT значением &HF1.

(1) Адрес в таблице образов

Номер квадрата, в котором будет выведена точка, равен:

N = (Y\8)*32 + (X\8) (= 9*32 + 22 = 300)

Номер соответствующего сегмента выводится посредством:

M = Y MOD 8 (= 75 MOD 8 = 3)

Адресом соответствующего байта в таблице образов будет:

AP = N*8 + M (= 300*8 + 3 = 2403 = 963(H))

(2) Адрес в CT

Достаточно прибавить &H2000 к адресу PGT.

AC = AP + &H2000 (= &H2963)

(3) Значение байта, загружаемого в PGT

X MOD 8 (100 MOD 8 = 4)

Следующее значение

VA = &B000010000 = 2^(7 - (X MOD 8))

должно быть загружено в PGT.

В таблице цветов, &HF1 введено для включения квадрата белого цвета на черном фоне.





Проверим все это:

PSET

```
10  SCREEN 2
20  X = 100 : Y = 75
30  N = (Y\8)*32 + (X\8)
40  M = Y MOD 8
50  AP = N*8 + M
60  VA = 2^7 - (X MOD 8))
70  VPOKE AP, VA
80  VPOKE AP + &H2000, &HF1
90  IF INKEY$ = "" THEN 90
100 PSET(X,Y), 6
110 GOTO 110
```

Сначала рисуется белая точка на черном фоне. Когда Вы нажимаете клавишу, эта точка перерисовывается красным цветом посредством PSET; итак, наши расчеты правильны.

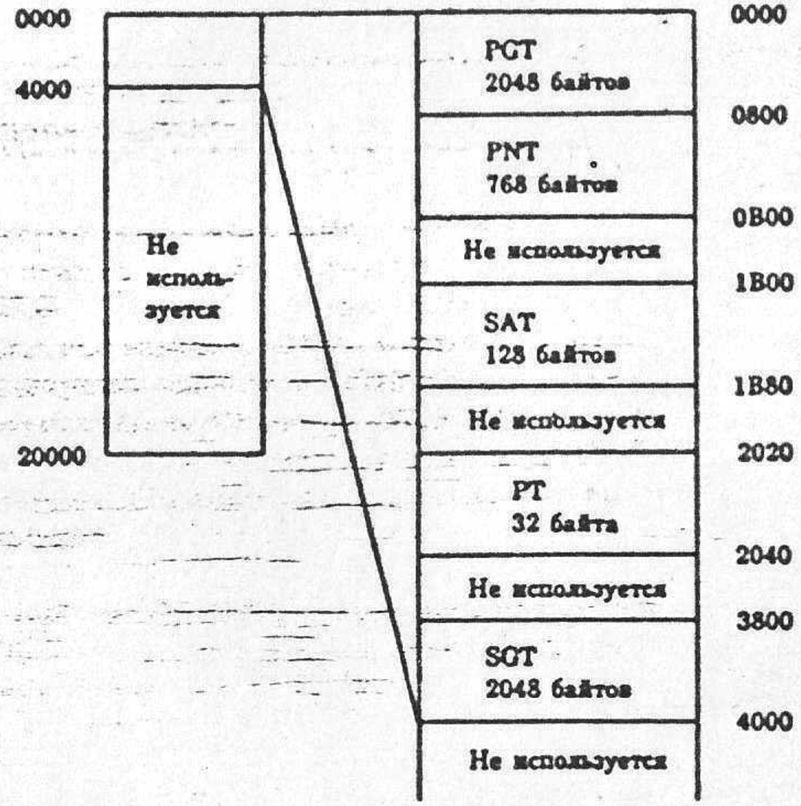
В MSX Бейсике этот способ расчетов практически бесполезен. Однако, если Вы хотите нарисовать что-либо с помощью программы на машинном языке, перечисленные шаги помогут Вам определить, что загружать в видеопамять, и как проверять операции в MSX Бейсике.

## 14-4. Режим SCREEN 3

### 14-4-1. Структура видеопамяти в SCREEN 3

Имеется 64 блока в ширину и 48 блоков в длину, каждый из которых дает 4 x 4 точки режима SCREEN 2. Каждый блок может иметь свой цвет. Поскольку один байт PGT кодирует цвета двух последовательных блоков, эта таблица требует 48*32 = 1536 байтов.

Далее, поскольку информация о цветах берется из PGT, CT в SCREEN 3 нет. PNT та же, что и в SCREEN 2 и инициализируется так же.




Все байты PGT инициализируются с цветом фона, закодированным как в четырех старших, так и в четырех младших битах. COLOR с последующим CLS вновь инициализирует эту таблицу.





### 14-4-2. Соотношение отображения и таблиц в SCREEN 3

Для простоты предположим, что таблица названий не изменялась после инициализации в SCREEN 3. Экран разделен на большие квадраты 8 x 8 (в 4 раза больше, чем в SCREEN 2). Тем самым имеется 8 квадратов по горизонтали и 6 по вертикале, всего 48 квадратов. Одна строка содержит 8 квадратов и занимает 4 байта в PGT. Потребуется 32 байта из этой таблицы для определения цвета каждого блока квадрата. Если квадраты пронумерованы, как раньше, первый из 32 байтов будет размещен в N°32. Первые 8 байтов задают вид первой колонки из двух блоков и т.п.


Пример: закрасить все блоки квадрата 3 произвольными цветами.

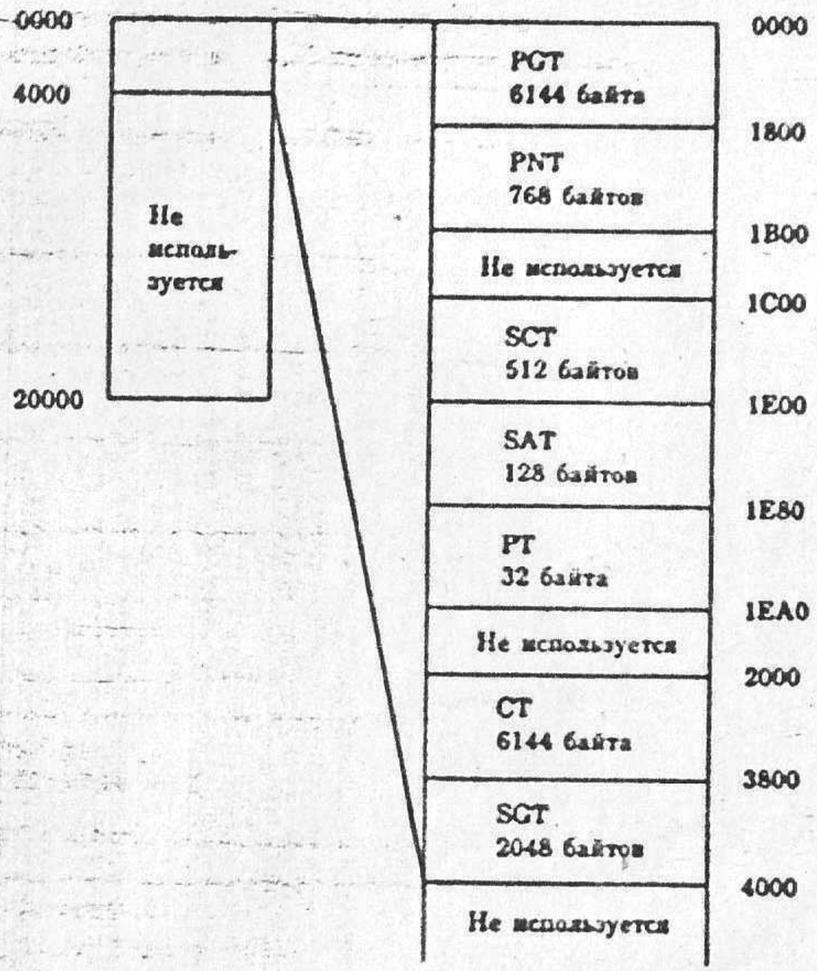
SCREEN 3

```
10  SCREEN 3 : DEFINT A-Z
20  FOR I = 0 TO 31
30  F = RND(1)*16 : B = RND(1)*16
40  VPOKE 32*3 + I, 16*F + B
50  NEXT
60  GOTO 20
```

Поскольку SCREEN 3 редко используется, на этом мы закончим его описание.

## 14-5. Режим SCREEN 4 (для компьютеров MSX-2)

Этот режим очень напоминает SCREEN 2, но с большими возможностями в том, что касается спрайтов (SPRITES). Поскольку все это будет обсуждено ниже, иллюстрации структуры памяти в этом режиме будет достаточно.



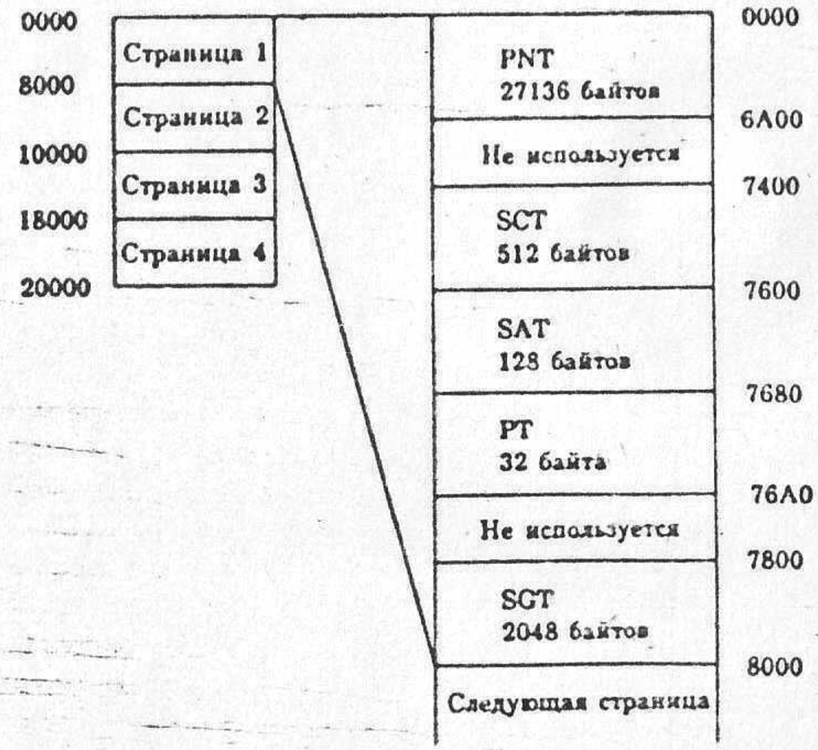





## 14-6. Режим SCREEN 5 (для компьютеров MSX-2)

### 14-6-1. Структура видеопамяти в SCREEN 5

Экран разделен на 212 строк из 256 пикселей. Каждый сегмент из двух пикселей закодирован в одном байте: первые четыре бита байта определяют цвет первого пикселя, а последние четыре - цвет второго. В результате, требуется 212*256/2 = 27136 байтов для хранения цветов пикселей экрана. PNT содержит эту часть информации. В SCREEN 5-8 нет PGT.



PNT инициализируется с цветом фона, использовавшимся до выполнения SCREEN 5, или с цветом фона, переопределенным посредством COLOR с последующим CLS.

### 14-6-2. Связь между PNT и отображением в режиме SCREEN5

Первый байт таблицы PNT связан с первым двухпиксельным сегментом первой строки растра; второй байт - со вторым сегментом и т.д. В отличие от рассмотренных ранее режимов, этот режим не имеет деления на квадраты.

Пример: включить белым цветом точку с координатами (X,Y)

SCREEN5

```
10  SCREEN 0
20  COLOR 15, 4, 7
30  INPUT"X,Y"; X, Y
40  SCREEN 5
50  AD = Y*128 + (X\2)
60  IF (X MOD 2) = 0 THEN C = &HF4 ELSE C
    = &H4F
70  VPOKE AD, C
80  GOTO 80
```

Вы легко можете убедиться, что это эквивалентно PSET (X,Y), 15.

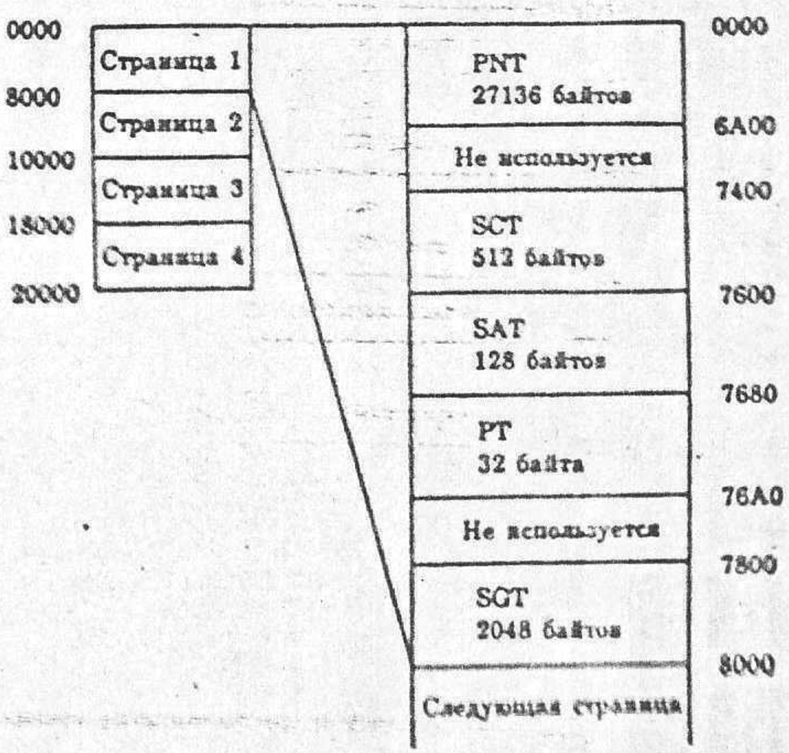




## 14-7. Режим SCREEN 6 (для компьютеров MSX-2)

### 14-7-1. Структура видеопамяти в SCREEN 6

Экран имеет 212 строк из 512 пикселей. Поскольку таблица названий не длиннее, чем в SCREEN 5, 4-пиксельный сегмент кодируется в одном байте (из PNT). Именно поэтому выбор ограничен только первыми палитрами (палитра, относящаяся к одному пикселю, кодируется в двух битах).



PNT инициализируется посредством последних двух битов цвета фона. Если, например, это цвет 5 = 101(B), таблица названий будет инициализирована байтами 01010101, т.е. 4 раза последние два бита-01 цвета фона. Помните, что одиночный оператор SCREEN 6 не инициализирует цвета: необходим COLOR с последующим CLS.

### 14-7-2. Связь между PNT и отображением в режиме SCREEN 6

Структура байтов PNT - та же, что и в SCREEN 5: первые 128 байтов кодируют первую строку и т.д. VPOKE 0, &B11100100 присваивает палитру 3 первому пикселю первой строки; второй пиксель получит палитру 2, третий палитру 1 и четвертый - палитру 0.

## 14-8. Режим SCREEN 7 (для компьютеров MSX-2 с объемом видеопамяти 128 Кбайт)

### 14-8-1. Структура видеопамяти в SCREEN 7

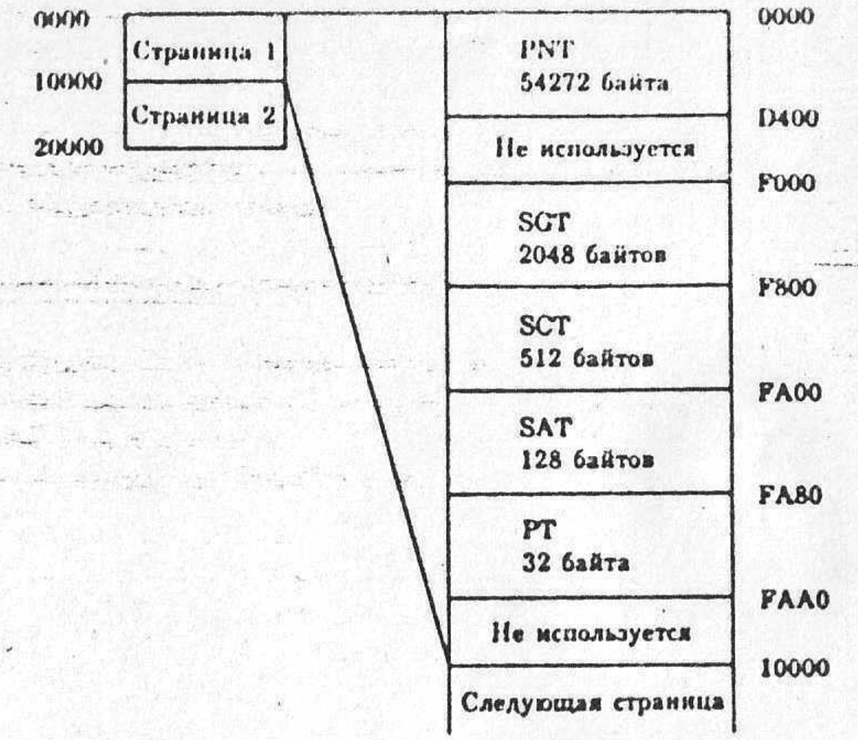
Как и в SCREEN 6, имеется 212 строк из 512 пикселей; и, как и SCREEN 5, каждому пикселю может быть присвоено 16 палитр. Поскольку для кодирования номера палитры достаточно четырех битов, таблица названий должна иметь длину 212*512/2 = 54272 байта (тем самым один байт привязывается к двухпиксельному сегменту).

### 14-8-2. Связь между PNT и отображением в режиме SCREEN 7

Все происходит так же, как в SCREEN 5, за исключением того, что здесь в строке пикселей в два раза больше.





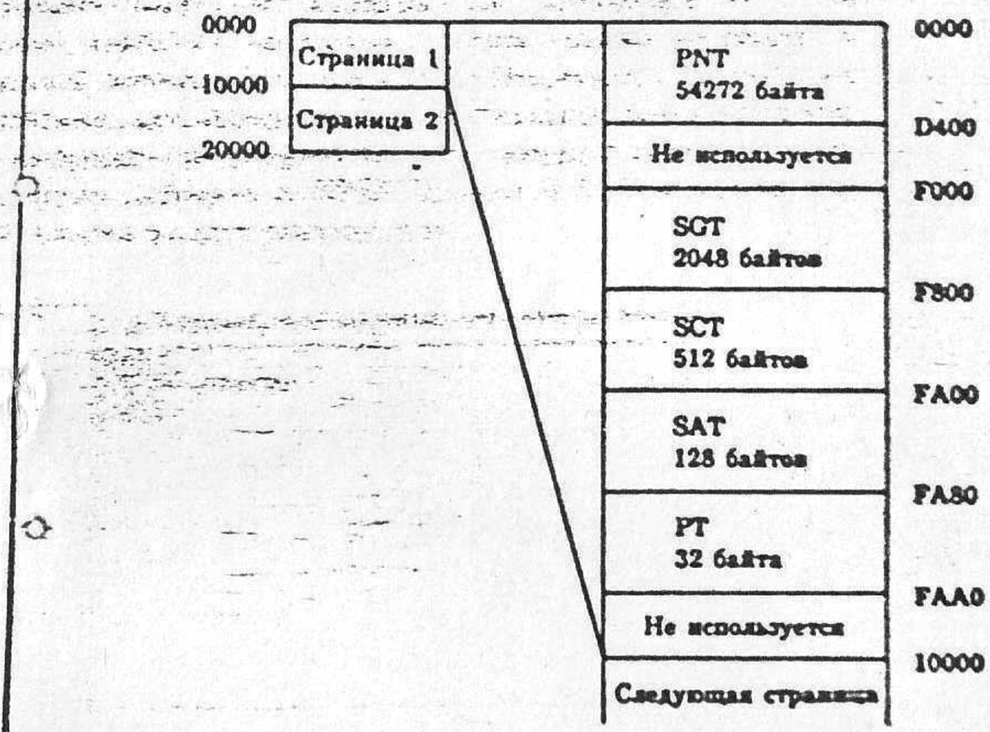



## 14-9. Режим SCREEN 8 (для компьютеров MSX-2 с объемом видеопамяти 128 Кбайт)

### 14-9-1. Структура видеопамяти в SCREEN 8

Экран состоит из 212 строк по 256 пикселей. Их цвет уже определяется не кодом палитры, а заданием непосредственной смеси основных цветов палитр. Этот способ задания цветов требует по одному байту на пиксель; поэтому PNT должна иметь длину 212*256 = 54272 байта.

При инициализации байты таблицы названий получают номер цвета фона (который изменяется в пределах от 0 до 255).




### 14-9-2. Связь между PNT и отображением в SCREEN 8

Точка с координатами (X, Y) должна находиться по адресу AD=X+256*Y. Для окрашивания этой точки достаточно загрузить значение C, лежащее между 0 и 255, по этому адресу.

Следующая формула возвращает значение C, соответствующее смеси основных цветов:





$$C = 32*G + 4*R + B$$

где G, R и B - значения в диапазоне от 0 до 7, определяющие интенсивность зеленого, красного и синего цветов соответственно.

Значение C = 187 = 10111011(B) тем самым соответствует G = 101(B) = 5, R = 110(B) =6, B = 11(B) = 3.

### 14-10. Таблица палитр (для компьютеров MSX-2)

Независимо от режима SCREEN, PT имеет в видеопамяти 32 байта. Каждая из палитр закодирована в двух байтах. Местоположение этой таблицы зависит от режима SCREEN. Например, в SCREEN 1, эта таблица начинается по адресу 2020(H). Адрес AD первого байта палитры N вычисляется по следующей формуле:

$$AD = A0 + 2*N$$

где A0 - адрес, отмечающий начало PT.

По этому адресу должен находиться байт "красный/синий"; биты 6-4 этого байта определяют пропорцию красного, а биты 2-0 - пропорцию синего цвета. Байт по адресу AD + 1 возвращает пропорцию зеленого, закодированную в битах 2-0.

Пример: чтение таблицы палитр в SCREEN 1

PALET1

```
10   SCREEN 1
20   '
30   FOR N = 0 TO 15
40   AD = &H2020 + 2*N
50   RB = VPEEK(AD) : G = VPEEK(AD + 1)
60   PRINT N; TAB(4); HEX$(RB); HEX$(6)
70   NEXT
```

Оператор SCREEN инициализирует таблицу палитр с пропорциями, показанными этой программой (определение палитр по умолчанию); оператор COLOR = (...) переопределяет байты этой таблицы, соответствующие указанной палитре.
Замените строку 20 программы следующей:

PALET2

```
20   COLOR = (7, 7, 7, 7)
```

Как видно, байты, соответствующие палитре 7, изменились, и цвет границы изображения немедленно изменился.

Разумеется, палитры могут быть также непосредственно изменены операторами VPOKE в таблице палитр.

Замените строку 20 программы следующей:

PALET3

```
20  VPOKE &H2020 + 14, &H77 : VPOKE &H2020 + 15, 7
```

Этот прием изменяет палитру 7, НО НЕ ВЛИЯЕТ НЕПОСРЕДСТВЕННО НА ИЗОБРАЖЕНИЕ.

Для изменения цветов изображения за операторами VPOKE





должен следовать оператор COLOR = RESTORE, который вызывает считывание палитры.

```
25   COLOR = RESTORE
```

Таблица палитр используется и спрайтами, и изображением (текст, фон и граница). НО ПРИ ЭТОМ:

- В SCREEN 6 для изображения используются только палитры от 0 до 3.
- В SCREEN 8 палитры не влияют на изображение.

## 14-11. Спрайты

В SCREEN 1-3 спрайты задаются посредством двух таблиц:

(1) Таблица аттрибутов спрайтов (SAT)
(2) Таблица генератора спрайтов (SGT)

В SCREEN 4-8 используется третья таблица:

(3) Таблица цветов спрайтов (SCT) (только для компьютеров MSX-2)

### 14-11-1. SGT

SGT содержит образы спрайтов, установленные оператором SPRITE$(N)= <строка символов>. Эта таблица сбрасывается на 0 оператором SCREEN, когда задан второй параметр (определяющий размер спрайта и его масштаб).
2048 байтов SGT позволяют хранить 256 форм, когда вторым параметром SCREEN является 0 или 1, и 64 формы, когда этот параметр равен 2 или 3. По следующим формулам вычисляется адрес образа, созданного с помощью SPRITE$
(N) =:

**A0 + 8*N или A0 + 32*N**

A0 - адрес начала SGT.

Байты образа записаны в SGT в том же порядке, как в строке, следующей за оператором SPRITE$(N) =.
Теперь понятно, что

**SPRITE$(0) = STRING$(8,255)**

эквивалентно

**FOR I = 0 TO 7 : VPOKE &H3800 + I, 255 : NEXT**

### 14-11-2. SAT

128 байтов SAT задают отображение спрайта. Поскольку одновременно могут отображаться 32 спрайта, атрибуты одного спрайта кодируются в 4 байтах:

| | |
|---|---|
| Первый байт | координата Y |
| Второй байт | координата X |
| Третий байт | указатель на SGT |
| Четвертый байт | номер палитры |

Если A0 - начальный адрес SAT, то следующая команда

**PUT SPRITE NS, (X,Y), C, NP**

эквивалентна

```
10  AD = AD + 8*NS
20  VPOKE AD, Y
30  VPOKE AD + 1, X
40  VPOKE AD + 2, NP
50  VPOKE AD + 3, C
```





при условии, что второй параметр SCREEN меньше 2. Если этот параметр равен 2 или 3, строку 40 следует заменить следующей:

```
40  VPOKE AD + 32*NP
```

PUT SPRITE, разумеется, гораздо легче использовать, чем последовательность операторов VPOKE. Однако, как будет показано, последовательность VPOKE гораздо проще смоделировать на машинном языке, чем оператор PUT SPRITE.

Более того, знание SAT позволяет прочитать атрибут спрайта, не помещая его в переменную.

Пример:
превращение спрайта в черный, если он был белым, и наоборот.

PRITE4

```
10  SCREEN 1, 1
20  SPRITE$(0) = STRING$(8,255)
30  PUT SPRITE 0, (20,40), 15, 0
40  IF INKEY$ = "" THEN 40
50  GOSUB 100 : GOTO 40
100 VPOKE &H1B03, VPEEK (&H1B03) XOR 14
110 RETURN
```

Оператор SCREEN вновь инициализирует SAT следующим образом:

| | |
|---|---|
| Первый байт | 209 |
| Второй байт | 50 |
| Третий байт | номер спрайта |
| Четвертый байт | цвет текста |

Координаты отображения спрайта по умолчанию - (50, 209). С учетом значения координаты Y спрайт остается невидимым. Образ спрайта имеет тот же номер, что и спрайт (по умолчанию NP = NS). Цветом спрайта по умолчанию является цвет текста.

### 14-11-3. SCT (для компьютеров MSX-2)

SCT используется только в режимах 4-8. Она соответствует оператору COLOR SPRITE. Поскольку для каждого из 32 спрайтов может быть определено 8 (формы 8 x 8) или 16 (формы 16 x 16) строк с различными цветами, эта таблица требует 32*16 = 512 байтов.

Адрес спрайта в SCT вычисляется по следующей формуле:

AD = A0 + 16*SN

где A0 - адрес начала SCT и SN - номер выведенного спрайта.

16 байтов цвета инициализированы цветом текста или цветом, заданным оператором PUT SPRITE. Они могут быть непосредственно переопределены операторами VPOKE или одним из операторов COLOR SPRITE.

Для форм 8 x 8 будут использоваться только первые 8 из 16 байтов.



# Управление видеопамятью

Глава 15

Если Вы посмотрите на карты видеопамяти в различных режимах SCREEN, Вы заметите, что некоторые области видеопамяти не используются изображением. Эти области могут служить в качестве дополнительного пространства для хранения данных, когда ОЗУ почти полностью использовано. VPOKE и VPEEK дают доступ к видеопамяти. Однако, эти операторы медленнее, чем POKE и PEEK.

Работа с псевдопеременными BASE и VDP - весьма эффективное средство управления видеопамятью. Однако при использовании этого средства нужно быть крайне осторожным: малейшая ошибка в программировании может сделать экран неуправляемым и вынудить Вас сбросить систему. Мы ограничимся несколькими безопасными примерами.

Более детальная информация о способах работы с видеопамятью содержится в книге "V9938 MSX-VIDEO Technical Data Book (Yamaha)".

### 15-1. Псевдопеременная BASE

BASE используется для чтения начальных адресов таблиц в видеопамяти, а также для переопределения организации видеопамяти.

### 15-1-1. Чтение адресов таблиц

При использовании в качестве функции BASE может иметь аргумент в диапазоне от 0 до 44 (19 в компьютерах MSX-1): BASE(N) возвращает начальный адрес таблицы T в режиме

SCREEN M. Параметры M и T связаны с аргументом N следующими формулами:

**M = N\5 и T = N MOD 5**

Номер таблицы T лежит в пределах от 0 до 4; режим экрана M может принимать любое значение между 0 и 8 (3 в компьютерах MSX-1).

Таким образом, можно иметь доступ не более, чем к 5 таблицам для каждого из режимов экрана. Ниже приведены имена таблиц, соответствующих различным значениям T:

| T | название таблицы |
|---|---|
| 0 | PNT |
| 1 | CT (кроме SCREEN 0, 3, 5, 6, 7, 8) |
| 2 | PGT (кроме SCREEN 5, 6, 7, 8) |
| 3 | SAT (кроме SCREEN 0) |
| 4 | SPT (кроме SCREEN 0) |

Пример: нахождение адреса SAT в SCREEN 7

Достаточно выполнить команду:

```
?HEX$(BASE(7*5 + 3))
```

Функция BASE не позволяет найти адреса таблицы палитр или SCT.

### 15-1-2. Установление расположения таблиц

Оператор BASE(N) = <адрес> позволяет перемещать таблицы. Этот оператор может использоваться только в режимах SCREEN 0-8, и поэтому N должен быть лежать в пределах от 0 до 19.





Пример:
запись нескольких страниц экрана в режиме SCREEN 0 и переключение страниц

Стандартный начальный адрес для PNT - 0. Передвинем эту таблицу к адресам &H2000 и &H2400. Каждый раз будем писать что-либо на этой странице для идентификации. Затем будем переключаться с одной страницы на другую посредством переопределения начала PNT.

BASE

```
10  SCREEN 0
20  FOR I = 0 TO 2
30  GOSUB 100 : GOSUB 200
40  NEXT I
50  IF INKEY$ = "" THEN 50
60  I = (I + 1) MOD 3 : GOSUB 100
70  GOTO 50
100 IF I = 0 THEN BASE(0) = 0 ELSE
    BASE(0) = &H2000 + (I - 1)*&H4000
110 IF I = 0 THEN POKE &HF923, 0 ELSE
    POKE &HF923, &H20 + (I - 1)*4
120 RETURN
200 CLS
210 FOR J = 0 TO 50
220 PRINT"PAGE"; I;
230 NEXT : RETURN
```

Примечание:

Адреса таблиц записаны в рабочей области начиная с адреса &HF922, и занимают по два байта каждый. Эти значения не изменяются оператором BASE. Вы должны сделать это сами (строка 110: изменяется только старший байт).

Помните, что в SCREEN 5-8 можно переключаться с одной страницы на другую с помощью оператора SET PAGE.

### 15-2. Псевдопеременная VDP

Следующие примеры иллюстрируют использование псевдопеременной VDP.

Пример 1: включение и выключение экрана

```
VDP(1) = VDP(1) XOR 64
```

Эта команда включает экран, если он был выключен, и наоборот.

Пример 2: изменение размера спрайта без очистки форм

```
VDP(1) = VDP(1) XOR 1
```

Эта команда переключает масштаб спрайтов с 1 на 2 или с 2 на 1 без очистки форм спрайтов.

Пример 3: изменение цветов текста и фона в SCREEN 0 или 1

```
VDP(7) =<цвет>
```

Значение цвета желательно писать в шестнадцатеричном виде. Первая цифра означает цвет текста, а вторая - цвет фона:

```
VDP(7) = &H1F
```

имеет тот же результат, что и

```
COLOR 1, 15
```



# Введение в программирование на машинном языке

Глава 16

### 16-1. Процессор Z-80

Ядром компьютера MSX является микропроцессор Z-80 (Z-80A) фирмы Зайлог (Zilog) или эквивалентный ему. Этот процессор имеет регистры общего и специального назначения, а также может работать с памятью объемом до 64 Кбайт (стандарт MSX обеспечивает расширение адресации памяти, см. Приложение Л). Z-80 является широко распространенным 8-разрядным процессором; однако его система команд содержит достаточно удобные средства для работы с 16-разрядными данными и символьными строками.

Перед выполнением оператора MSX Бейсика выполняется проверка его правильности, и в случае обнаружения ошибок выводятся сообщения на экран. Выполнимый оператор сначала преобразуется в последовательность команд, которые может "понять" процессор. Эти последовательности, или подпрограммы, запрограммированы заранее и помещены в ПЗУ на этапе изготовления компьютера.

Интерпретация операторов MSX Бейсика занимает определенное время; в некоторых случаях, однако, задержка ответа может быть сокращена, особенно для игровых программ. Возможны два решения:

(1) Основная программа пишется на MSX Бейсике, а в ситуациях, когда MSX Бейсик не может быть использован или не обеспечивает нужного быстродействия, используются подпрограммы в машинных кодах.

(2) Вся программа пишется на машинном языке. В этой главе будут рассмотрены оба этих варианта.

Детальное описание синтаксиса оператор Z-80 заняло бы слишком много места. Существует обширная литература по этому кругу проблем, и читатель, овладевший MSX Бейсиком, но желающий попробовать программировать на машинном языке, может обратиться к этой литературе. Мы же предполагаем, что у читателя имеется описание системы оператор Z-80, и он знаком с ее особенностями. Мы же рассмотрим возможности использования этих операторов на компьютере MSX.

## 16-2. Оператор DEFUSR и функция USR

Для обращения к подпрограммам в машинных кодах из программ на MSX Бейсике используются оператор DEFUSR и функция USR.

Оператор DEFUSR определяет начальный адрес машинной подпрограммы. Функция USR выполняет подпрограмму. В программе на MSX Бейсике может использоваться до 10 подпрограмм одновременно. Они различаются по номерам, которые записываются за словом USR и лежат в диапазоне от 0 до 9.

Пусть, например, одна машинная подпрограмма записана по адресу &HD000, а другая - по адресу &HD100. Эти подпрограммы определяются посредством операторов

```
DEFUSR0 = &HE000
DEFUSR1 = &HE100
```

Для выполнения этих подпрограмм в программе на MSX Бейсике задаются пользовательские функции

```
A = USR0(A)
A = USR1(A)
```





В большинстве случаев, функция USR имеет пустой аргумент. В данном случае, значение, возвращаемое функцией USR, представляет собой просто значение аргумента. Далее мы рассмотрим другие способы передачи аргументов машинным подпрограммам и их связи с переменными в языке MSX Бейсик.

## 16-3. Подпрограммы BIOS

Эти подпрограммы перечислены в Приложении E. Для них заданы стандартные входные точки, являющиеся адресами ПЗУ. Прежде, чем Вы начнете разработку собственных подпрограмм в машинных кодах, рекомендуем тщательно изучить функции подпрограмм BIOS.

### 16-3-1. Подпрограммы BIOS без параметров

Достаточно одного взгляда на описание подпрограмм BIOS, чтобы заметить, что некоторые подпрограммы для работы вовсе не требуют ввода данных.

Следующие примеры иллюстрируют использование таких подпрограмм.

Пример 1: инициализация системы.

```
DEFUSR = 0:A = USR(0)
```

Это "убийственная" инициализация - в памяти уничтожается все содержимое (так же, как при нажатии клавиши сброса RESET).

Пример 2: Инициализация значений функциональных клавиш.

Время от времени может оказаться желательным возвращение к стандартному определению текстов, присвоенных функциональным клавишам. Для этого можно поступить следующим образом:

FKEY

```
10  KEY1, "AAAA" : KEY2, "BBBB"
20  DEFUSR = &H3E
30  IF INKEY$ = "" THEN 30
40  A = USR(0) : KEY ON
```

Подпрограмма **INIFNK** по адресу **&H3E** переопределяет функциональные клавиши. Однако, следует выполнить **KEY ON**, чтобы увидеть результат этого на экране.

Пример 3: включение и выключение экрана

Если Вы хотите вывести сразу "полный" экран, можно выключить экран, выполнить операторы вывода, а затем включить экран. Подпрограмма DISSCR по адресу &H41 выключает экран, тогда как другая подпрограмма - ENASCR по адресу &H44 включает его.

CRTONOFF

```
10  DEFUSR = &H41 : DEFUSR1 = &H44
20  SCREEN 0
30  A = USR(0)
40  FOR I = 0 TO 200
50  PRINT"A";
60  NEXT
70  A = USR1(0)
```





### 16-3-2. Подпрограммы BIOS, требующие параметров

Большинство подпрограмм BIOS требуют для своей работы передачи параметров. Такие данные должны быть помещены в определенные регистры перед вызовом подпрограмм. Такие подпрограммы обычно запускаются простой программой в машинных кодах, состоящей всего из двух частей:

(1) запись данных в регистры
(2) вызов подпрограммы BIOS

Следующие примеры позволяют ознакомиться с приемами написания подпрограмм на машинном языке.

Пример 1: программа, включающая и выключающая индикатор CAPS (ввод заглавных букв).

Подпрограмма **CHGCAP** по адресу **&H132** выключает CAPS, если регистр **A** процессора содержит ненулевое значение, и включает, если этот регистр содержит 0.

Сначала опишем алгоритм программы, включающей индикатор CAPS:

(1) загрузить 0 в регистр A
(2) перейти к подпрограмме CHGCAP

На языке ассемблера (см. любое пособие по системе команд Z-80) это записывается следующим образом:

```
LD A, 0      загрузить 0 в A
JP 0132      перейти к адресу &H132
```

На машинном языке эта программа выглядит следующим образом (в шестнадцатиричных кодах):

```
3E, 00      LD A, 0
C3, 32, 01  JP 0132
```

Теперь эта программа должна быть сохранена. Первый шаг заключается в резервировании специальной области в ОЗУ. Затем данные, т.е. текст программы в кодах, могут быть записаны в эту область посредством операторов POKEs

CAPS1

```
10  CLEAR 200, &HD000
20  AD = &HD000
30  READ A$
40  IF A$ = "Z" THEN 100
50  A = VAL("&H" + A$)
60  POKE AD, A
70  AD = AD + 1 : GOTO 30
80  '
100 DEFUSR = &HD000
110 A = USR(0)
120 '
200 ' D000 CAPS ON
210 ' *** 00, 01, 02, 03, 04, 05, 06, 07
220 DATA 3E, 00, C3, 32, 01, "Z"
```





Примечание:

Для написания программ на машинном языке обычно требуется программа-монитор (т.е. программа, позволяющая подключать подпрограммы на машинном языке). Примеры программ-мониторов содержатся во многих пособиях по вычислительной технике. Если Вы не пользуетесь программой-монитором, приходится вводить данные в операторах DATA, как мы это только что проделали. Однако, для длинных подпрограмм этот путь утомителен. При малейшей ошибке система часто "зависает"; единственный выход при этом - нажать кнопку сброса RESET, а это означает, что все должно быть начато заново. Перед проверкой таких подпрограмм убедитесь, что они записаны на дискету. Обратите внимание на комментарии в операторе DATA; они содержат младший байт используемого адреса, что удобно, если внутри подпрограммы осуществляются переходы.

Приведенная выше программа включает индикатор CAPS. Выключите его вручную и введите:

```
?USR(0)
```

Заметьте, что подпрограмма CHGCAP изменяет только состояние индикатора, не затрагивая состояние самой клавиши CAPS.

Пример 2: включение и выключение индикатора CAPS в соответствии со значением аргумента USR

Для передачи однобайтного аргумента подпрограмме вызовите подпрограмму, размещенную в ПЗУ по адресу &H521F. Аргумент передается в регистре A.

Замените строку 220 следующей (имя подпрограммы CAPS2):

```
220 DATA CD, 1F, 52, C3, 32, 01, "%"
```

На языке ассемблера это пишется следующим образом:

```
CALL 521F   вызов подпрограммы в 521F
JP 0132     переход к 0132
```

Заметьте, что

```
?USR(0)     включает индикатор
?USR(1)     выключает его
```

Для передачи двухбайтного значения подпрограмме (например, адреса) используется подпрограмма, размещенная в ПЗУ по адресу &H2F8A и использующая значение в паре регистров HL.

### 16-3-3. Подпрограммы BIOS, возвращающие параметры

Многие подпрограммы BIOS возвращают один или несколько параметров (см. Приложение E: OUT). Эти результаты, разумеется, можно передать через область ОЗУ с явно заданным адресом непосредственно перед выходом из подпрограммы. В этом случае применение функции PEEK позволит Вам прочитать данные.

Однако когда подпрограмма возвращает только одно значение, проще присвоить его результат переменной в операторе: <переменная> = USR(<аргумент>)

Пример: проверка состояния джойстиков и передача значений через переменную в программе на MSX Бейсике.





Подпрограмма GTSTCK позволяет ввести данные с одного из трех джойстиков. Джойстик при этом выбирается значением в регистре A (0, 1 или 2); прочитанное значение помещается в регистр A.

Если номер джойстика задается посредством аргумента USR, подпрограмма должна быть запущена заново обращением к &H521F. Как только аргумент будет в регистре A, вызывается подпрограмма GTSTCK. Содержание регистра A (состояние джойстика) затем передается через пару регистров HL. Подпрограмма, размещенная в &H2FF99, передает содержание HL программе на MSX Бейсике. Подпрограмма пишется следующим образом:

```
D000  CD, 1F, 52    CALL 521F     A = аргумент
D003  CD, D5, 00    CALL 00D5     A = STICK(A)
D006  26, 00        LD H, 0       H = 0
D008  6F            LD L, A       L = A
D00A  C3, 99, 2F    JP 2F99       USR = HL
```

Загрузите эту подпрограмму, как показано выше, а затем проверьте ее посредством программы:

GETSTICK

```
100 DEFUSR = &HD000
110 S = USR(0)
120 IF S <> 0 THEN PRINT S
130 GOTO 110
```

Вы убедитесь, что функция USR здесь работает точно так же, как STICK.

## 16-4. Использование ловушек.

В некоторых ключевых точках ПЗУ управление передается на ловушку в ОЗУ. При инициализации большинство ловушек содержат только операторы RET(&HC9). В этом случае ничего не происходит. Однако, замена RET на JUMP в Вашей собственной подпрограмме фактически позволяет Вам модифицировать подпрограмму в ПЗУ, использующую эту ловушку.

Пример: отмена действия клавиши STOP

При выполнении программы на MSX Бейсике клавиатура постоянно опрашивается (сканируется). Нажатие клавиши порождает код, который записывается в буфер клавиатуры (с адреса &HF55E в рабочей области). До этого, код нажатой клавиши временно записывается в ячейку NEWKEY с адресом HFBE5. Регистр A загружен значением, равным 8•<номер строки матрицы клавиатуры> + <номер колонки> (см. матрицу клавиатуры). Осуществляется переход на ловушку KEYCOD по адресу &HFDCC. Обычно ловушка состоит из 5 байтов, содержащих только команду RET (&HC9). Введем вместо нее переход на подпрограмму обработки ловушки. Эта подпрограмма будет действовать только при нажатии клавиши STOP. Пусть регистр A содержит 7 • 8 + 4 = 60 = &H3C. Подпрограмма посылает значение &B11111101 = &HFD по адресу &HFBEB (буфер NEWKEY), имитируя тем самым нажатие клавиши CTRL. При этом, нажатие STOP будет прочитано как CTRL + STOP, что будет обработано в MSX Бейсике по обычным правилам. Сначала следует зарезервировать область хранения для подпрограммы обработки ловушки:





```
STOPTRAP
10  CLEAR 200,&HD000
```

Затем записать команду перехода в ловушку KEYCOD:

C3, 00, E0  JP E000

```
20  POKE &HFDCC, &HC3
30  POKE &HFDCD, 0
40  POKE &HFDCE, &HD0
```

Подпрограмма обработки ловушки пишется следующим образом:

```
D000 FF, 3C      CP, 3C       A = 3C?
D002 C0          RET NZ       нет → выход
D003 3E, FD      LD A, FD     A = FD
D005 32, EB, FB  LD FBEB, A   A → FBEB
D008 3E, 3C      LD A, 3C     A = 3C
D00A C9          RET          выход
```

Загрузка осуществляется как обычно:

```
50  AD = &HD000
60  READ A$ : IF A$= "Z" THEN END
70  POKE AD, VAL("&H" + A$)
80  AD = AD + 1 : GOTO 60
90  '
100 ' D000 перехват клавиши STOP
110 ' ••• 00, 01, 02, 03, 04, 05, 06, 07
120 DATA FE, 3C, C0, 3E, FD, 32, EB, FB
130 ' ••• 08, 09, 0A, 0B, 0C, 0D, 0E, 0F
140 DATA 3E, 3C, C9, "Z"
```

Выполните эту программу; обратите внимание, что нажатие клавиши STOP производит такой же эффект, как CTRL + STOP в командном режиме.

Удалите вышеприведенную программу и введите:

```
10  POKE &HFBB0, 1
20  ON STOP GOSUB 100 : STOP ON
30  PRINT A : A = A + 1 : GOTO 30
100 RETURN
```

Нажатие [STOP] теперь не приводит к прекращению программы с выводом на экран курсора. Поскольку [STOP] "превращен" в [CTRL] + [STOP], реакция на которые известна, клавиша [STOP] "перестает работать".
Комбинация [CTRL] + [GRAPH] + [РУС] + [SHIFT] останавливает эту программу.

## 16-5. Написание программы на машинном языке (без программы-монитора)

Программа в машинных кодах пишется следующим образом:

(1) Сначала пишется программа на MSX Бейсике, которая будет вводить машинные подпрограммы в защищенную область в ОЗУ. Предположим, что машинная программа вводится таким образом по адресам с SA по EA.

(2) Машинная программа записывается на диск в двоичном формате оператором

BSAVE "имя", SA, EA, XA

Последний параметр (адрес выполнения XA) необязателен. Он указывает адрес запуска, когда программа загружается с добавлением параметра R (автоматическое выполнение; см. ниже). Это удобно, если в начале машинной программы имеются подпрограммы.





(3) Для использования программы включите компьютер и загрузите программу с помощью оператора:

BLOAD "имя", R, 0A

Последний параметр (адрес смещения 0A) необязателен. Он указывает интервал между адресом SA, который использовался для загрузки, и адресом, по которому будет размещена программа.
ВНИМАНИЕ: если программа содержит абсолютные переходы, вызовы внутренних подпрограмм или данные, не исключено, что ее невозможно будет перемещать в памяти таким образом.
Приведенная ниже программа полностью написана на машинном языке. Она устанавливает экран в режимы SCREEN 2, 3, загружает форму спрайта и двигает спрайт по горизонтали под управлением левой и правой клавиш курсора. Эта программа размещается с адреса &HA000.
Ниже приведен текст этой программы на языке ассемблера.

(1) Размер спрайта

```
A000  3E, E3      LD A, E3     A = E3
A002  32, E0, F3  LD F3E0, A   A→F3E0 VDP(1)
```

(2) SCREEN 2

```
A005  CD, 72, 00  CALL 0072
```

(3) Загрузка SGT (таблицы генератора спрайтов)
(предполагается, что данные о формах находятся по адресу &HA050).

```
A008  21, 50, A0   LD HL, A050   HL = начало данных
A00B  11, 00, 38   LD DE, 3800   DE = начало SGT
A00E  01, 20, 00   LD BC, 0020   BC = длина данных
A011  CD, 5C, 00   CALL 005C     пересылка
                                 ОЗУ->видеопамять
A014  C9           RET           выход
```

(4) Загрузка SAT (таблицы атрибутов спрайтов)
(предполагается, что данные находятся по адресу &HA070).

```
A018  21, 70, A0   LD HL, A070   HL = начало данных
A01B  11, 00, 1B   LD DE, 1B00   DE = начало SAT
A01E  01, 04, 00   LD BC, 0000   BC = длина данных
A021  CD, 5C, 00   CALL 005C     пересылка О ЗУ →
                                 видеопамять
A024  C9           RET           выход
```

(5) Ввод кодов нажатых клавиш

```
A028  3E, 00       LD A, 00      A = 0 курсорные
                                 клавиши
A02A  CD, D5, 00   CALL 00D5     A = STICK(0)
A02D  C9           RET           выход
```

(6) Основная программа

```
A030  CD, 00, A0   CALL A000     Экран/SGT
A033  CD, 18, A0   CALL A018     SAT
A036  CD, 28, A0   CALL A028     STICK(0)
A039  21, 71, A0   LD HL, A071   HL = X координатный
                                 указатель
A03C  FE, 03       CP 03         A = 3?
```





```
A03E  20, 01       JR NZ, 01     нет → JUMP
A040  34           INC (HL)      (HL) = (HL) + 1
A041  FE, 07       CP 07         A = 7?
A043  20 01        JR NZ, 01     нет → JUMP
A045  35           DEC (HL)      (HL) = (HL) - 1
A046  01, FF, 00   LD BC, FF     счетчик = 255
A049  0B           DEC BC        BC = BC - 1
A04A  78           LD A, B       BC = 0?
A04B  B1           OR C          нет → цикл
A04E  C3, 33, A0   JP A018       инициализация
```

Исходная программа:

SPRIT

```
10  CLEAR 200, &HA000
20  A = &HA000
30  READ A$ : IF A$ = "Z" THEN 50
40  POKE A, VAL("&H" + A$) : A = A + 1 :
    GOTO 30
50  BSAVE"BSPRIT", &HA000, &HA073, &HA030
60  '
100 ' A000 ЭКРАН/SGT
105 '*** 00, 01, 02, 03, 04, 05, 06, 07
110 DATA 3E, E3, 32, E0, F3, CD, 72, 00
115 '*** 08, 09, 0A, 0B, 0C, 0D, 0E, 0F
120 DATA 21, 50, A0, 11, 00, 38, 01, 20
125 '*** 10, 11, 12, 13, 14, 15, 16, 17
130 DATA 00, CD, 5C, 00, C9, 00, 00, 00
135 '
140 ' A018 SAT
145 '*** 18, 19, 1A, 1B, 1C, 1D, 1E, 1F
150 DATA 21, 70, A0, 11, 00, 1B, 01, 04
155 '*** 20, 21, 22, 23, 24, 25, 26, 27
160 DATA 00, CD, 5C, 00, C9, 00, 00, 00
165 '
```

```
170 ' A028 STICK(0)
175 ' *** 28, 29, 2A, 2B, 2C, 2D, 2E, 2F
180 DATA 3E, 00, CD, D5, 00, C9, 00, 00
185 '
190 ' A030 ОСНОВНАЯ ПОГРАММА
195 ' *** 30, 31, 32, 33, 34, 35, 36, 37
200 DATA CD, 00, A0, CD, 18, A0, CD, 28
205 ' *** 38, 39, 3A, 3B, 3C, 3D, 3E, 3F
210 DATA A0, 21, 71, A0, FE, 03, 20, 01
215 ' *** 40, 41, 42, 43, 44, 45, 46, 47
220 DATA 34, FE, 07, 20, 01, 35, 01, FF
225 ' *** 48, 49, 4A, 4B, 4C, 4D, 4E, 4F
230 DATA 00, 0B, 78, B1, 20, FB, 18, E3
235 '
240 ' A050
250 DATA FF, 80, 80, 80, 80, 80, 80, 80
260 DATA 80, 80, 80, 80, 80, 80, 80, FF
270 DATA FF, 01, 01, 01, 01, 01, 01, 01
280 DATA 01, 01, 01, 01, 01, 01, 01, 01
285 '
290 ' A070 SAT (Y, X, цвет формы спрайта)
300 DATA 10, 10, 00, 0F, "Z"A
```

Эта программа загружает машинную программу как двоичный файл с именем BSPRIT. (Для изменения формы спрайта, достаточно изменить данные в строках с 250 по 280).
Запуск этой программы выполняется оператором:

```
BLOAD"BSPRIT",R
```

ВНИМАНИЕ: единственный способ выйти из этой программы - это выключить компьютер или нажать кнопку сброса RESET.





Для изменения скорости перемещения спрайта, измените значение счетчика в &HA047 и &HA048.

Оператор CLEAR не является единственным способом выделения защищенной области в ОЗУ. Может быть также передвинута таблица PIT; тогда все машинные подпрограммы могут быть введены в область с адреса &H8000.

Машинная программа может быть написана также в виде массивов переменных, символьных строк или программных строк REM. Более подробно см. в руководствах по программированию для Z-80.

## 16-6. Короткая программа "Монитор звуковых эффектов"

Закончим главу текстом короткой программы-монитора, которая позволяет вводить и представлять в графической форме данные для регистров 0 и 2 звукогенератора компьютера. После ввода эти данные могут быть прочитаны с различными скоростями. Как сами эти данные, так и машинная подпрограмма могут быть записаны для использования в программе на MSX Бейсике.

На экран выводится квадрат, по которому можно передвигать курсор (курсорными клавишами). Координатные оси задают значения в регистрах 0 и 2 звукогенератора. Если удерживать клавишу пробела нажатой, то траектория курсора выводится на экран, и запоминается частотная "партия" создаваемого звукового эффекта. Число шагов также выводится на экран. Для чтения этих данных, нажмите RETURN. Значение регистров 1 и 3 может быть изменено посредством клавиш F1 и F2.
Скорость чтения может быть изменена клавишами F4 и F5.
Для сброса регистров, нажмите ESC.
Для сохранения данных в двоичном виде, нажмите INS.

Программа

SEM

```
10  '+------------------------------------+
20  '|                                    |
30  '|  SOUND EFFECT MONITOR              |
40  '|                                    |
50  '+------------------------------------+
60  '
70  '-----------основная инициализация
80  CLEAR 200, &HA000 : DEFINTA-Z : DEFSNG N
90  ON KEY GOSUB 820, 830, 890, 900
100 DEFUSR = &HA018 : DEFUSR1 = &HA080
110 DEFUSR2 = &HA0B8
120 '---------- загрузка машинных
                подпрограмм
130 A = &HA000
140 READ A$ : IF A$ = "Z" THEN 180
150 POKE A, VAL("&H" + A$)
160 A = A + 1 : GOTO 140
170 '---------- инициализация экрана
180 SCREEN 2 : COLOR 15, 11, 7 : CLS
190 OPEN"GRP:"AS#1
200 '---------- форма спрайта
210 S$ = "" : FOR I = 0 TO 7 : READ A$
220 S$ = S$ + CHR$(VAL("&H" + A$)
230 NEXT: SPRITE$(0) = S$
240 '----------- фон
250 LINE(8,160)-(247,191), 15, B
260 PRESET(16,164)  : PRINT#1, "TONE"
270 PRESET(64,164)  : PRINT#1, "INIT"
280 PRESET(108,164) : PRINT#1, "SAVE"
290 PRESET(152,164) : PRINT#1, "AUTO"
300 PRESET(200,164) : PRINT#1, "DELAY"
```





```
310 PRESET(16,173)  : PRINT#1, "[F1]"
320 PRESET(16,182)  : PRINT#1, "[F2]"
330 PRESET(60,178)  : PRINT#1, "[ESC]"
340 PRESET(104,178) : PRINT#1, "[INS]"
350 PRESET(148,178) : PRINT#1, "[RET]"
360 PRESET(204,173) : PRINT#1, "[F4]"
370 PRESET(204,182) : PRINT#1, "[F5]"
380 PRESET(80,145)  : PRINT#1, "STEPS"
390 '---------- инициализация регистров
400 L = 0 : GOSUB 630
410 SOUND 8, 15 : SOUND 9, 15
420 SOUND 7, &B10111100
430 T = 20 : GOSUB 690
440 SOUND 0, 0 : SOUND 2, 0
450 POKE &HA0B0, &HA1 : POKE &HA0B1, 0
460 N = -1 : GOSUB 760
470 LINE(64,10)-STEP(127,126), 1, BF
480 PUT SPRITE0, (64,10), T, 0
490 '---------- клавиатура
500 FOR I = 1 TO 5 : KEY(I) ON : NEXT
510 A = USR(0) : IF NOT USR1(0) THEN 540
520 PSET(VPEEK(&H1B01), VPEEK(&H1B00))
530 GOSCB 760 : GOTO 510
540 K$ = INKEY$
550 IF K$ <> CHR$(27) THEN 580
560 FOR I = 1 TO 5 : KEY(I) OFF : NEXT
570 GOTO 400
580 IF K$ <> CHR$(13) THEN 600
590 A = USR2(0) : GOTO 510
600 IF K$ <> CHR$(18) THEN 510
610 GOSUB 950 : GOTO 510
```

```
620  '
630  ' ---------- грубая настройка
640  SOUND 1, L : SOUND 3, L
650  LINE(16,144)-STEP(47,8), 9, BF
660  PSET(32,145), 9
670  PRINT#1, USING"##"; L
680  RETURN
690  ' ---------- задержка
700  POKE &HA0B2, (T*10)256 / 256
710  POKE &HA0B2, (T*10)MOD256
720  LINE(192,144)-STEP(47,8), 9, BF
730  PSET(198,145), 9
740  PRINT#1, USING"#####"; T*10
750  RETURN
760  ' ---------- длина
770  LINE(128,144)-STEP(40,8), 11, BF
780  N = N + 1
790  PRESET(128,145) : PRINT#1, N
800  RETURN
810  ' ---------- изменение грубой настройки
820  L = L - 1 : GOTO 840
830  L = L + 1
840  IF L > 15 THEN L = 15
850  IF L < 0 THEN L = 0
860  GOSUB 630
870  RETURN
880  ' ---------- изменение задержки
890  T = T - 1 + 9*(T > 100) : GOTO 910
900  T = T + 1 - 9*(T > 99)
910  IF T > 1000 THEN T = 1000
920  IF T < 1 THEN T = 1
```





```
930  GOSUB 690
940  RETURN
950  ' ---------- SAVE
960  IF N = 0 THEN RETURN
970  BSAVE"A:QUEUE", &HA000, &HC000
980  RETURN
990  ' A000---------- курсор X-Y
1000 DATA 00, 01, 01, 01, 00, FF, FF, FF
1010 DATA FF, FF, 00, 01, 01, 01, 00, FF
1020 ' A010---------- задержка
1030 DATA 0B, 79, B0, 20, FB, C9, 00, 00
1040 ' A018---------- STICK
1050 DATA 3E, 00, CD, D5, 00, FE, 00, C8
1060 DATA 3D, 26, A0, 6F, 7E, F5, 7D, C6
1070 DATA 08, 6F, 7E, F5, 3E, 00, CD, 96
1080 DATA 00, C1, 80, FE, FF, 20, 02, 3E
1090 DATA 00, FE, FE, 20, 02, 3E, FD, 5F
1100 DATA 3E, 00, CD, 93, 00, 3E, 02, CD
1110 DATA 96, 00, C1, 80, FE, FF, 20, 02
1120 DATA 3E, 00, FE, FE, 20, 02, 3E, FD
1130 DATA 5F, 3E, 02, CD, 93, 00, 3E, 00
1140 DATA CD, 96, 00, CB, 3F, C6, 0A, 21
1150 DATA 00, 1B, CD, 4D, 00, 3E, 02, CD
1160 DATA 96, 00, CB, 3F, C6, 40, 21, 01
1170 DATA 1B, CD, 4D, 00, C9, 00, 00, 00
1180 ' A080---------- STRIG
1190 DATA 3E, 00, CD, D8, 00, FE, 00, C8
1200 DATA 21, B0, A0, 46, 7E, FE, C0, C8
1210 DATA 23, 4E, 03, 03, 71, 2B, 70, C5
1220 DATA 3E, 00, CD, 96, 00, C1, 02, 03
1230 DATA C5, 3E, 02, CD, 96, 00, C1, 02
```

```
1240 DATA 21, FF, FF, C3, 99, 2F, 00, 00
1250 ' A0B0 : длина       /A0B2 : задержка
1260 DATA 00, 00, 00, 00, 00, 00, 00, 00
1270 ' A0B8 ..... проигрывание
1280 DATA 21, 00, A1, 3A, B0, A0, BC, 20
1290 DATA 05, 3A, B1, A0, BD, C8, E5, 5E
1300 DATA 3E, 00, CD, 93, 00, E1, 23, E5
1310 DATA 5E, 3E, 02, CD, 93, 00, 3A, B2
1320 DATA A0, 47, 3A, B3, A0, 4F, CD, 10
1330 DATA A0, E1, 23, C3, BB, A0, 00, 00, "Z"
1340 ' ..... спрайт
1350 DATA 80, C0, E0, F0, F8, FC, 78, 78
```

Программа на MSX Бейсике делится на несколько частей, описанных ниже.

(1) Основная инициализация

Область выше &HA000 защищается.
Определяются подпрограммы обработки функциональных клавиш F1, F2, F4 и F5.
Определяются три функции USR.

(2) Загрузка машинных подпрограмм

(3) Инициализация экрана

SCREEN 2, цвет и открытие файла "GRP:".

(4) Образ спрайта

Загрузка образа спрайта, который будет служить курсором.





(5) Фон

Вывод на экран данных, которые останутся неизменными.

(6) Инициализация регистров

Инициализация переменной L (старшая половина байта частот). Подпрограмма "грубой настройки" помещает значение L в регистры 1 и 3 и выводит на экран загруженное значение.
Инициализация регистров 8 и 9 (громкость) и регистра 7 (микширование).
Инициализация переменной T, устанавливающей интервал между двумя последовательными чтениями. Подпрограмма "задержки" помещает значение T в оба байта, &HA0B2 и &HA0B3, и выводит на экран загруженное значение.
Инициализация регистров 0 и 2.

Инициализация счетчика шагов. Этот счетчик присвоен как переменной N, так и начальному адресу данных в байтах &HA0B0 и &HA0B1. Подпрограмма "длина" выводит на экран число уже использованных шагов. Если прорисовка пути курсора идет слишком медленно, вставьте в начале подпрограммы RETURN.
Прорисовка черного квадрата и высвечивание курсора в верхнем левом углу.

(7) Клавиатура

Функциональные клавиши включены (ON).
USR вызывает (STICK): если клавиша курсора нажата, то курсор движется и регистры 0 и 2 обновляются.
USR1 вызывает (STRIG): если клавиша пробела нажата, текущий адрес данных наращивается, и текущие значения в регистрах 0 и 2 помещаются по этим новым адресам. USR1 возвращает -1, когда клавиша пробела нажата, и 0 в противном

случае. Если USR1 возвращает -1, точка, в которой находится курсор, помечается, и вызывается подпрограмма "длина".
Чтение клавиатуры (переменная K$):

Если [ESC], функциональные клавиши отключаются и инициализация регистров начинается заново.
Если [RETURN], вызывается USR2 (чтение).
Если [INS], данные записываются на диск.

Ниже следует краткое описание машинных подпрограмм:

(1) STICK

Регистр A устанавливается на 0 (номер STICK) и вызывается GTSTCK. Если регистр A содержит 0, происходит выход из подпрограммы. 1 извлекается из A, и это значение используется для установки новых адресов, где записываются приращения X и Y. Они соответствуют значениям, возвращаемым GTSTCK, отличным от 0. Прочитанные значения добавляются к регистрам 0 и 2 посредством RDPSG или WRTPSG, после проверки на избежание циклических модификаций. Новые значения регистров 0 и 2 затем превращаются в координаты X и Y, и посылаются в SAT с помощью WRTVRM.

(2) STRIG

A устанавливается на 0 (клавиша пробела), и вызывается GTTRIG.
Если A содержит 0, мы выходим из подпрограммы. В противном случае начальный адрес данных, помещенный в &HA0B0 и &HA0B1, наращивается дважды (при условии, что этот адрес ниже &HC000), и значения регистров копируются в новые верхние адреса. Следующим шагом будет помещение -1 (&HFFFF) в HL переход на адрес &H2F99.





(3)Проигрывание

Верхний адрес данных считывается и сравнивается с начальным значением (&HA100), записанным в HL. Если оба значения совпадают, происходит выход из подпрограммы, в противном случае оба значения
посылаются в регистры 0 и 2. Новое значение T, хранящееся в &HA0B2 и &HA0B3, считывается, и вызывается машинная подпрограмма "задержка". HL наращивается, и вышеуказанная проверка осуществляется еще раз.

Рекомендации по использованию этой программы:

Предположим, что нужно создать звуковой эффект для игровой или какой-либо другой программы. Работа с описанным монитором будет вестись методом "проб и ошибок". При этом достаточно изменять тональность и скорость считывания.
При получении желаемого эффекта, следует нажать [INS], выполнить [CTRL] + [STOP] и ввести NEW.

Программа, в которой будет использоваться этот звуковой эффект, должна содержать следующие операции:

(1) Инициализация ОЗУ

```
CLEAR 200,&HA000
BLOAD"A:QUEUE"
```

(2) Определение функции USR

```
DEFUSR = &HA0B8
```

(3) Инициализация регистров 1, 3, 7, 8 и 9.

Разумеется, можно добавить также некоторые шумы и работу с генератором огибающей (см. Главу 9).

**(4) Чтение числовых данных**

```
A = USR(0)
```





# Программирование портов

Порты представляют собой интерфейсные схемы компьютера MSX. Процессор Z-80 работает с портами по адресам, которые не следует путать с адресами ПЗУ или ОЗУ.

Основные порты, их функции и адреса перечислены ниже.

| Адрес (16-чн) | Чтение/ Запись | Функция |
|---|---|---|
| | | **ПРИНТЕР** |
| 90 | Ч | Ввод сигнала занятости принтера BUSY |
| 91 | З | Вывод стробирующего сигнала STROBE |
| 92 | З | Вывод символа на принтер |
| | | **ВИДЕОПРОЦЕССОР (VDP)** |
| 98 | Ч/З | Обращение к регистру VDP (чтение/запись) |
| 99 | Ч/З | Регистр состояния |
| | | **ЗВУКОГЕНЕРАТОР** |
| A0 | З | Команда генератора |
| A1 | З | Запись в регистр |
| A2 | Ч | Чтение портов 14 и 15 |
| | | **ПРОГРАММИРУЕМЫЙ ПАРАЛЛЕЛЬНЫЙ ИНТЕРФЕЙС (PPI)** |
| A8 | Ч/З | Запись/чтение данных порта A |
| A9 | Ч/З | Запись/чтение данных порта B |
| AA | Ч/З | Запись/чтение данных порта C |
| AB | Ч/З | Запись/чтение данных регистра режимов PPI |

Команда **OUT** посылает данные в порт. Функция **INP** считывает значение из порта. Команда **WAIT** ожидает, пока значение, удовлетворяющее условию, не появится в порте.
Примеры иллюстрируют возможное использование **INP**, **OUT** и **WAIT**.

## 17-1. Программируемый параллельный интерфейс (PPI)

Для чтения любой клавиши клавиатуры, номер строки матрицы клавиатуры записывается в порт C PPI, а номер колонки читается в порте B.

Пример 1: обнаружение нажатия клавиши [GRAPH].
Клавиша [GRAPH] находится на строке 6 матрицы клавиатуры. Адрес порта C - &HAA. Тогда команда принимает вид

```
OUT &HAA, 6
```

Затем считывается порт B:

```
X = INP(&HA9)
```

Клавиша [GRAPH] находится в колонке 2 матрицы. Если клавиша нажата, бит 2 в X устанавливается на 0. Таким образом, эта клавиша может быть обнаружена следующей командой:

```
IF (X AND 4) = 0 THEN PRINT "GRAPH"
```

Обратите внимание, что эта программа включает индикатор CAPS. Это происходит потому, что в порт C записывается значение 6. Фактически, только четыре младших бита этого порта определяют сканируемую строку матрицы клавиатуры. Для маскирования четырех старших битов достаточно выполнить





```
OUT &HAA, 6 OR (INP(&HAA) AND &HF0)
```

вместо

```
OUT &HAA, 6
```

Биты порта C имеют следующее значение:

| | |
|---|---|
| Биты 0-3 | Строка клавиатуры |
| Бит 4 | Если 0, запускается кассетная лента |
| Бит 5 | Сигнал записи на ленту |
| Бит 6 | Если 0, включается индикатор CAPS |
| Бит 7 | Управление звуковым сигналом |

Пример 2: включение и выключение кассетной ленты

Используем операторы MSX Бейсика **MOTOR ON** и **MOTOR OFF**.
Эти операторы могут быть имитированы с помощью

```
OUT &HAA, INP(&HAA) XOR &B00010000
```

Пример 3: включение и выключение индикатора CAPS

```
OUT &HAA, INP(&HAA) XOR &B01000000
```

Порт A используется для выбора с л о т о в, осуществляющих управление расширенной памятью компьютера (см. Приложение Л).

## 17-2. Другие порты

Они функционируют точно также, как и PPI.

Пример 1: Считывание кода последнего выведенного на экран символа

```
10 K$ = INKEY$ : IF K$ = ""THEN 10
20 PRINT K$;
30 PRINT INP(&H98)
40 GOTO 10
```

Оператор WAIT используется нечасто. Он записывается следующим образом:

WAIT порт, маска, выбор

"порт" - адрес порта; "маска" и "выбор" - два числа в диапазоне от 0 до 255; "выбор" может опускаться.

Указанный порт работает в режиме чтения. Результат X комбинируется с параметром "выбор":

X = X XOR выбор (ненулевые биты быбора инвертируют биты X)

Новый результат X комбинируется с маской:

X = X AND маска (нулевые биты маски обнуляют биты X)

Оператор WAIT останавливает работу компьютера, пока результатом опроса порта оказывается нуль. Однако нажатие CTRL + STOP позволяет вернуться в командный режим.



# УПРАВЛЯЮЩИЕ КЛАВИШИ И КОДЫ

ПРИЛОЖЕНИЕ А

| 10-чн код | 16-чн код | Клавиши | Функция |
|---|---|---|---|
| 1 | 01 | CTRL + A | Заголовок графического символа |
| 2 | 02 | CTRL + B | Возвращение к началу предыдущего слова (оператора или параметра) |
| 3 | 03 | CTRL + C | Возвращение к началу следующей строки без ввода данных (*) |
| 4 | 04 | CTRL + D | Игнорируется (*) |
| 5 | 05 | CTRL + E | Очищает логическую строку начиная с позиции курсора |
| 6 | 06 | CTRL + F | Передвигает курсор на следующее слово (оператор или параметр) (*) |
| 7 | 07 | CTRL + G | ВЕЕР(звуковой сигнал); сбрасывает звуковые регистры на 0 (*) |
| 8 | 08 | CTRL + H = BS | Стирает символ слева от курсора и сдвигает последующие символы влево |
| 9 | 09 | CTRL + I = TAB | Передвигает курсор на 8 символов вправо |
| 10 | 0A | CTRL + J | Передвигает курсор вниз в той же колонке; возможно продвижение текста на экране(*) |
| 11 | 0B | CTRL + K = HOME | Передвигает курсор в начальное положение |

# СООБЩЕНИЯ ОБ ОШИБКАХ

## ПРИЛОЖЕНИЕ Б

Ниже перечислены сообщения об ошибках и соответствующие коды.

**ПРИМЕЧАНИЕ:**

**ERROR код** Осуществляет возврат в командный режим и выводит сообщение об ошибке, соответствующее данному коду; код должен находиться в пределах от 1 до 255.

**X = ERR** Присваивает переменной X код последней встреченной ошибки.

**X = ERL** Присваивает переменной X значение номера строки, где произошла последняя ошибка (или 65535, если ошибка произошла в командном режиме

**КОДСООБЩЕНИЕ**

**1 NEXT without FOR (NEXT без FOR)**
Встречен NEXT, но отсутствует переменная, определенная посредством FOR:
Переменная в операторе NEXT не соответствует никакой ранее выполненной и не имеющей пары переменной оператора FOR. Либо в программе слишком много NEXT, либо ошибка в переменной, следующей за NEXT.

**2 Syntax error (Синтаксическая ошибка)**
Встречена строка, содержащая неправильную последовательность символов (например, скобка без пары, неправильно написанное ключевое слово, неправильная расстановка знаков препинания и т.д.). Причиной ошибки является либо незнание синтаксиса, либо простая описка.

**3 RETURN without GOSUB (RETURN без GOSUB)**
Встречен оператор RETURN, для которого нет предыдущего не связанного в другой паре оператора GOSUB. Это, например, происходит, когда конец основной программы и начало подпрограммы не разделены посредством END.

**4 Out of DATA (Исчерпаны данные)**
Встречен READ, когда не осталось непрочитанных данных. В программе отсутствует оператор RESTORE или условие выполнения оператора READ.

**5 Illegal function call (Неправильный вызов функции)**
Это сообщение появляется обычно в том случае, если аргумент функции выходит за установленные пределы. Примерами этого являются:
- Отрицательный или слишком большой индекс
- Отрицательный или нулевой аргумент в LOG, или
- Отрицательный аргумент в SQR.
- Использование функции USR, для которой еще не задан начальный адрес
- Использование ERASE, SWAP, VARPTR с неиспользуемой переменной
- Неправильный аргумент в MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, ON...GOTO(GOSUB) и т.д.
- Отрицательный номер записи, использованный с GET или PUT
- Использование графических операторов в SCREEN 0 или в SCREEN 1.
- Неправильный символ в музыкальной последовательности PLAY или графической последовательности DRAW.
- Вызов сервиса в SCREEN 0.

**6 Overflow (Переполнение)**
Число превышает максимально представимое в машине. Если число меньше наименьшего представимого, то оно приравнивается нулю.





**7 Out of memory (Память исчерпана)**
Программа слишком велика или содержит слишком много переменных (слишком большие массивы), слишком много циклов FOR, переходов GOSUB (для которых требуются метки переходов в стеке). Иногда этого можно избежать, записывая массивы в файлы на дискете, исправляя распределение памяти соответствующими CLEAR или записывая под программы в двоичные файлы. В некоторых случаях можно хранить данные в неиспользуемых областях видеопамяти (VRAM).

**8 Undefined line number (Не определен номер строки)**
Введение нового номера строки без текста; переход (THEN, ELSE, GOTO, GOSUB) на номер несуществующей строки; ссылка на несуществующий номер в RESTORE и DELETE. Обратите внимание, что RENUM обнаруживает недостающие переходы.

**9 Subscript out of range (Выход за пределы массива)**
Обращение к элементу массива с индексом, превышающим размерность массива, либо с недопустимым количеством индексов.

**10 Redimensioned array (Повторное задание размерности массива)**
Задано два оператора DIM для одного массива; либо DIM задан для массива после того, как его размерность была задана по умолчанию во время предыдущего использования. Для изменения размерности массива его нужно предварительно уничтожить с помощью ERASE.

**11 Division by zero (Деление на ноль)**
Деление на ноль или ноль возводится в отрицательную степень (ноль в нулевой степени принимается равным 1). Помните, что значение будет приравнено нулю, если оно меньше минимального числа, представимого в компьютере MSX-2 ("машинного нуля").

**12 Illegal direct (Неправильная команда в командном режиме)**
Использование в командном режиме оператора, используемого только в программах (например, DEFFN).

**13 Type mismatch (Несоответствие типов данных)**
Числовой переменной присваивается символьное значение или наоборот; аргумент или параметр не совпадает с ожидаемым.

**14 Out of string space (Исчерпано место для строковых переменных)**
Пространство в памяти, отведенное под строковые переменные по умолчанию (200 байтов) или посредством CLEAR, исчерпано. Следует перезадать пространство, использовать файлы (дискету) или записать строки в неиспользуемые области видеопамяти (VRAM).

**15 String too long (Слишком длинная строка)**
Символьное выражение содержит более 255 символов. Такая строка должна быть разделена на меньшие части.

**16 String formula too complex (Строковое выражение слишком сложно)**
Строковое выражение слишком длинно или слишком сложно. Оно должно быть разбито на меньшие выражения.

**17 Can't continue (Продолжение невозможно)**
Была сделана попытка продолжить программу, которая:
(1)Прекратилась из-за ошибки, а не по клавише STOP или CTRL + STOP.
(2)Модифицировалась после останова (Break)
(3)Не существует





**18 Undefined user function (Неопределенная пользовательская функция)**
Вызывается функция USR до того, как задано ее определение.

**19 Device I/O error (Ошибка устройства ввода-вывода)**
Произошла ошибка во время операции ввода-вывода. Это может произойти, например, когда CTRL + STOP прерывает операцию чтения с кассетной ленты.

**20 Verify error (Ошибка верификации)**
Программа в памяти отличается от программы, проверяемой с помощью CLOAD.

**21 No RESUME (Отсутствует оператор RESUME)**
Осуществляется переход по ошибке ON ERROR GOTO, в то время как подпрограмма обработки прерывания по ошибке не содержит оператора RESUME.

**22 RESUME without error (Оператор RESUME при отсутствииошибки)**
Встречен оператор RESUME до перехода в подпрогамму обработки прерывания по ошибке посредством ON ERROR GOTO.

**23 Unprintable error (ошибка без вывода текста сообщения)**
Насколько известно, такая ошибка может быть вызвана только в результате оператора ERROR (имитация ошибки).

**24 Missing operand (Пропущен операнд)**
Задан оператор без обязательного параметра. Например, SCREEN.

**25 Line buffer overflow (Переполнение буфера строки)**
Попытка введения оператором INPUT строки, содержащей более 255 символов (из файла).

**26-49 Unprintable error (ошибка без вывода текста сообщения)**
См. 23.

**50 Field overflow (Переполнение поля)**
Оператор FIELD пытается разместить больше байтов, чем было определено для длины записи в файле произвольного доступа.

**51 Internal error (Внутренняя ошибка)**
Неисправность. Произведите сброс компьютера посредством кнопки сброса RESET либо выключением.

**52 Bad file number (Ошибочный номер файла)**
Номер файла соответствует файлу, который не открыт, либо этот номер превышает число, заданное в MAXFILES.

**53 File not found (Файл не найден)**
Требуемый файл не существует на диске.

**54 File already open (Файл уже открыт)**
Оператор режима последовательного вывода OPEN задан для файла, который уже открыт таким оператором; либо оператор KILL относится к открытому файлу.

**55 Input past end (Ввод после конца файла)**
Оператор INPUT пытается прочитать больше данных, чем есть в файле. Во избежание этой ошибки используйте EOF для обнаружения конца файла.

**56 Bad file name (Ошибочное имя файла)**
Неправильное имя файла (например, слишком длинное) в LOAD, KILL или OPEN.





**57 Direct statement in file (Командный оператор в файле)**
Оператор командного режима встречен при загрузке файла в формате ASCII оператором LOAD. Действие LOAD прекращается. Выдается также при попытке загрузить командой типа LOAD двоичный файл.

**58 Sequential I/O only (Только последовательный ввод-вывод)**
Оператор GET или PUT ошибочно использован по отношению к файлу, открытому в качестве последовательного посредством OPEN.

**59 File not OPEN (Файл не открыт командой OPEN)**
Оператор ввода-вывода используется по отношению к файлу, который не был открыт оператором OPEN.

**60 Bad FAT (Неверная информация в FAT - таблице распределения файлов)**
Таблица распределения файлов (FAT) не в порядке. Возможно, дискета не инициализирована командой FORMAT.

**61 Bad file mode (Неверный режим обращения к файлу)**
Сделана попытка использовать PUT, GET или LOF по отношению к последовательному файлу, загрузить командой LOAD файл произвольного доступа, или выполнить оператор OPEN в неподходящем файловом режиме.

**62 Bad drive name (Неправильное имя дисковода)**
Использовано неправильное имя дисковода.

**63 Bad sector number (Неправильный номер сектора)**
В последней версии MSX DISK Бейсика этой ошибки не должно быть.

**64 File still open (Файл еще открыт)**
Файл не был закрыт оператором CLOSE.

**65 File already exists (Файл уже существует)**
Имя файла, заданное в операторе NAME, идентично имени файла, уже использованному на диске.

**66 Disk full (Дискета заполнена)**
Все пространство дискеты использовано.

**67 Too many files (Слишком много файлов)**
Сделана попытка создать новый файл (используя SAVE или OPEN), в то время как все 255 возможных элементов каталога заполнены.

**68 Disk write protected (Дискета защищена от записи)**
Дискета защищена от попыток записи.

**69 Disk I/O error (Ошибка при выводе на дискету или вводе с нее)**
Ошибка во время операции ввода-вывода по отношению к дискете.

**70 Disk offline (Дисковод отключен)**
Дисковод не подключен к компьютеру.

**71 Rename across disks (ошибка переименования файла)**
Сделана попытка переименования файла с новым идентификатором дисковода. Это не допускается.

**72-255 Unprintable error (ошибка без вывода текста сообщения)**
См.23.



# ВЫРАЖЕНИЯ ДЛЯ НЕКОТОРЫХ МАТЕМАТИЧЕСКИХ ФУНКЦИЙ



## ВЫРАЖЕНИЯ ДЛЯ НЕКОТОРЫХ МАТЕМАТИЧЕСКИХ ФУНКЦИЙ

| | |
|---|---|
| СЕКАНС | SEC(X) = 1/COS(X) |
| КОСЕКАНС | CSC(X) = 1/SIN(X) |
| КОТАНГЕНС | COT(X) = 1/TAN(X) |
| АРКСИНУС | ARCSIN(X) = ATN(X/SQR(-( X*X + 1)) |
| АРККОСИНУС | ARCCOS(X) = - ATN(X/SQR(- X*X + 1)) + PI |
| АРКСЕКАНС | ARCSEC(X) = = ATN(X/SQR(X*X - 1)) + SGN(SGN(X) - 1)*PI |
| АРККОСЕКАНС | ARCCSC(X) = = ATN(X/SQR(X*X - 1)) + (SGN(X) - 1)*PI |
| АРККОТАНГЕНС | ARCCOT(X) = ATN(X) + 1.5708 |
| ГИПЕРБОЛИЧЕСКИЙ СИНУС | SINH(X) = (EXP(X) - EXP(-X))/2 |
| ГИПЕРБОЛИЧЕСКИЙ КОСИНУС | COSH(X) = (EXP(X) + EXP(-X))/2 |
| ГИПЕРБОЛИЧЕСКИЙ ТАНГЕНС | TANH(X) = = (EXP(X) - EXP(-X))/(EXP(X) + EXP(-X)) |
| ГИПЕРБОЛИЧЕСКИЙ СЕКАНС | SECH(X) = 2/(EXP(X) + EXP(-X)) |
| ГИПЕРБОЛИЧЕСКИЙ КОСЕКАНС | CSCH(X) = 2/(EXP(X) - EXP(-X)) |
| ГИПЕРБОЛИЧЕСКИЙ КОТАНГЕНС | COTH(X) = (EXP(X) + EXP(-X))/(EXP(X) - EXP(-X)) |
| ГИПЕРБОЛИЧЕСКИЙ АРКСИНУС | ARCSINH(X) = LOG(X + SQR(X*X + 1)) |

| | |
|---|---|
| ГИПЕРБОЛИЧЕСКИЙ АРККОСИНУС | ARCCOSH(X) = LOG(X + SQR(X*X - 1)) |
| ГИПЕРБОЛИЧЕСКИЙ АРКТАНГЕНС | ARCTANH(X) = LOG((1 + X)/(1 - X))/2 |
| ГИПЕРБОЛИЧЕСКИЙ АРКСЕКАНС | ARCSECH(X) = LOG((SQR(-X*X + 1) + 1)/X) |
| ГИПЕРБОЛИЧЕСКИЙ АРККОСЕКАНС | ARCCSCH(X) = LOG((SNG(X)*SQR(X*X + 1) + 1)/X) |
| ГИПЕРБОЛИЧЕСКИЙ АРККОТАНГЕНС | ARCCOTH(X) = LOG((X + 1)/(X - 1))/2 |
| PI = 4*ATN(1)<br>1 радиан = 180/PI градусов<br>1 градус = PI/180 градусов | |



# СПИСОК КЛЮЧЕВЫХ СЛОВ С ВНУТРЕННИМИ КОДАМИ



(1) Однобайтные слова

| Внутр. код (шест) | Ключ.слово | Внутр. код (шест) | Ключ.слово | Внутр. код (шест) | Ключ.слово |
|---|---|---|---|---|---|
| 81 | END | 82 | FOR | 83 | NEXT |
| 84 | DATA | 85 | INPUT | 86 | DIM |
| 87 | READ | 88 | LET | 89 | GOTO |
| 8A | RUN | 8B | IF | 8C | RESTORE |
| 8D | GOSUB | 8E | RETURN | 8F | REM |
| 90 | STOP | 91 | PRINT | 92 | CLEAR |
| 93 | LIST | 94 | NEW | 95 | ON |
| 96 | WAIT | 97 | DEF | 98 | POKE |
| 99 | CONT | 9A | CSAVE | 9B | CLOAD |
| 9C | OUT | 9D | LPRINT | 9E | LLIST |
| 9F | CLS | A0 | WIDTH | A1 | ELSE |
| A2 | TRON | A3 | TROFF | A4 | SWAP |
| A5 | ERASE | A6 | ERROR | A7 | RESUME |
| A8 | DELETE | A9 | AUTO | AA | RENUM |
| AB | DEFSTR | AC | DEFINT | AD | DEFSNG |
| AE | DEFDBL | AF | LINE | B0 | OPEN |
| B1 | FIELD | B2 | GET | B3 | PUT |
| B4 | CLOSE | B5 | LOAD | B6 | MERGE |
| B7 | FILES | B8 | LSET | B9 | RSET |
| BA | SAVE | BB | LFILES | BC | CIRCLE |
| BD | COLOR | BE | DRAW | BF | PAINT |
| C0 | BEEP | C1 | PLAY | C2 | PSET |
| C3 | PRESET | C4 | SOUND | C5 | SCREEN |
| C6 | VPOKE | C7 | SPRITE | C8 | VDP |

| Внутр. код (шест) | Ключ.слово | Внутр. код (шест) | Ключ.слово | Внутр. код (шест) | Ключ.слово |
|---|---|---|---|---|---|
| C9 | BASE | CA | CALL | CB | TIME |
| CC | KEY | CD | MAX | CE | MOTOR |
| CF | BLOAD | D0 | BSAVE | D1 | DSKO$ |
| D2 | SET | D3 | NAME | D4 | KILL |
| D5 | IPL | D6 | COPY | D7 | CMD |
| D8 | LOCATE | D9 | TO | DA | THEN |
| DB | TAB( | DC | STEP | DD | USR |
| DE | FN | DF | SPC( | E0 | NOT |
| E1 | ERL | E2 | ERR | E3 | STRING$ |
| E4 | USING | E5 | INSTR | E6 | |
| E7 | VARPTR | E8 | CSRLIN | E9 | ATTR$ |
| EA | DSKI$ | EB | OFF | EC | INKEY$ |
| ED | POINT | EE | > | EF | = |
| F0 | < | F1 | + | F2 | - |
| F3 | * | F4 | / | F5 | ^ |
| F6 | AND | F7 | OR | F8 | XOR |
| F9 | EQV | FA | IMP | FB | MOD |
| FC | | | | | |





(2) Двухбайтные слова (младший байт всегда 255)

| Внутр. код (шест) | Ключ.слово | Внутр. код (шест) | Ключ.слово | Внутр. код (шест) | Ключ.слово |
|---|---|---|---|---|---|
| 81 | LEFT$ | 82 | RIGHT$ | 83 | MID$ |
| 84 | SGN | 85 | INT | 86 | ABS |
| 87 | SQR | 88 | RND | 89 | SIN |
| 8A | LOG | 8B | EXP | 8C | COS |
| 8D | TAN | 8E | ATN | 8F | FRE |
| 90 | INP | 91 | POS | 92 | LEN |
| 93 | STR$ | 94 | VAL | 95 | ASC |
| 96 | CHR$ | 97 | PEEK | 98 | VPEEK |
| 99 | SPACE$ | 9A | OCT$ | 9B | HEX$ |
| 9C | LPOS | 9D | BIN$ | 9E | CINT |
| 9F | CSNG | A0 | CDBL | A1 | FIX |
| A2 | STICK | A3 | STRIG | A4 | PDL |
| A5 | PAD | A6 | DSKF | A7 | FPOS |
| A8 | CVI | A9 | CVS | AA | CVD |
| AB | EOF | AC | LOC | AD | LOF |
| AE | MKI$ | AF | MKS$ | B0 | MKD$ |

# СИСТЕМЫ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ

## ПРИЛОЖЕНИЕ Д

Вся информация, которую компьютер может обработать, представлена числами в соответствии со строгими правилами. Компьютер состоит из большого числа микросхем, характеризующимися двумя уровнями (значениями) напряжения (вкл/выкл). Эти два различных состояния могут представлять две цифры — 1 и 0. Следовательно, система представления, реализуемая в компьютере, должна быть двоичной, использующей только цифры 1 и 0 для представления любого числа, а не привычной десятичной системой. Программисты, с другой стороны, привыкли к написанию некоторых данных, особенно команд в машинных языках, в шестнадцатеричной системе, которая компактней десятичной. Иногда используется также восьмеричная и так называемая двоично-десятичная.

### Д-1. Десятичная система

Овладение десятичной системой поможет пониманию того, как выглядят другие системы.
Число всегда может рассматриваться как полином с основанием 10.

$$356 = 6 * 10^0 + 5 * 10^1 + 3 * 10^2$$

10 называется основанием: оно представляет собою число, которое возводится в нужную степень в зависимости от своей позиции в вышеприведенном выражении.
Основание определяет также количество различных цифр, используемых в системе: десятичная система использует 10 цифр (0–9).

## Д-2. Двоичная, восьмеричная и шестнадцатеричная системы

Независимо от системы счисления, число всегда может рассматриваться в качестве полинома "по B", где B – основание. Более того, основание определяет общее количество цифр, используемых для написания чисел.
Следовательно, число abcd с основанием B может быть интерпретировано следующим образом:

$$abcd = d * B^0 + c * B^1 + b * B^2 + a * B^3$$

Значение цифр a, b, c и d находится между 0 и B–1.

В следующей таблице описаны четыре системы счисления, рассматриваемые в этом Приложении, и используемые в них цифры:

| Система | Используемые цифры | | | | | | | | | | | | | | | | Основание |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Двоичная | 0 | 1 | | | | | | | | | | | | | | | 2 |
| Восьмеричная | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | | | | | | | | 8 |
| Десятичная | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | | | | | 10 |
| Шестнадцатеричная | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 16 |
| Десятичное значение цифр | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |

В шестнадцатеричной системе не хватает 6 символов, необходимых для представления чисел со значением от 10 до 15, поэтому используются буквы от A до F.





Примечание:

Во всех системах последовательные степени основания записываются следующим образом: 1, 10, 100, 1000, 10000,...

## Д-3. Преобразования

(1) Преобразование из любой системы в десятичную
Преобразуемое число первоначально переводится в полином:

```
0011(B)  = 1*2^0 + 1*2^1 + 0*2^2 + 0*2^3 + 1*24
         = 1     + 2     + 0     + 0     + 16
         = 19
1AF(H)   = F*16^0 + A*16^1 + 1*16^2
         = F      + A*16    + 1*256
         = 15     + 160     + 256
         = 431
```

- (B) и (H) означают соответственно двоичную и шестнадцатеричную системы.

(2) Преобразование из десятичной в другую систему
Пусть B является основанием, к которому должно быть приведено число N1, записанное в десятичном виде.

Преобразование осуществляется следующим образом:

- Вычисляется R1 = N1 MOD B
  N2 = N1\B

- Если N2 = 0, преобразование заканчивается.
  Если N2 > 0, таким же образом вычисляются R2 и N3.
  Полученная последовательность R1, R2, R3 ... соответствует цифрам числа с основанием B.

Пример 1: записать 19 в двоичном виде

```
R1   = 19 MOD 2 = 1
N2   = 19\2 = 9
R2   = 9 MOD 2 = 1
N3   = 9\2 = 4
R3   = 4 MOD 2 = 0
N4   = 4\2 = 2
R4   = 2 MOD 2 = 0
N5   = 2\2 = 1
R5   = 1 MOD 2 = 1
N6   = 1\2 = 0
19(D) = 10011(B)
```

Пример 2: записать 234 в шестнадцатиричном виде

```
R1     = 234 MOD 16 = 10 → A
N2     = 234\16 = 14
R2     = 14 MOD 16 = 14 → E
N3     = 14\16 = 0
234(D) = EA(H)
```

(3)Преобразование "двоичная/шестнадцатиричная"

Преобразование между любыми двумя системами может осуществляться через посредство десятичной. Сначала преобразование идет в соответствии с п.(1), а затем — п.(2).
Преобразования между двоичной и шестнадцатиричной системами могут осуществляться быстрее:

Пример 1: записать 100011111 (B) в шестнадцатиричном виде

- Двоичное число делится на группы из 4 цифр каждая, начиная справа. Затем отдельные группы прямо преобразуются в шестнадцатиричные цифры.





```
1 | 0001 | 1111
1     1      F
10011111 (B) = 11F(H)
```

Пример 2: записать 11F (H) в двоичном виде

- Каждая шестнадцатиричная цифра переводится в двоичную с добавлением необходимых нулей для получения групп из 4 двоичных цифр.

```
   1     1     F
0001  0001  1111
11F(H) = 000100011111 = 100011111(B)
```

## Д-4. Представление в памяти целых чисел

Итак, все числа должны быть записаны в электронной памяти компьютера в двоичном виде. Однако, существуют различия между "математическим" двоичным представ-лением, которое только что было объяснено, и компьютерным:

- Математические целые числа могут быть любого размера, в то время как размер чисел, хранимых в реальном компьютере, из практических соображений должен быть ограничен

- Для представления отрицательных чисел в математике перед числом просто ставится знак минус. В памяти компьютера элементы могут представлять только 0 и 1. Поэтому для представления знака числа необходим специальный прием.

Элемент, способный представлять 0 и 1 называется битом.
Компьютерная память разделена на группы по восемь битов, называемые байтами. Компьютер может обратиться к содержимому любого байта в памяти по определенному "адресу". Из практических соображений целое число хранится в двух байтах, что дает 65536 различных конфигураций:

```
00000000 00000000 = 0
00000000 00000001 = 1
00000000 00000010 = 2
00000000 00000011 = 3
.....
11111111 11111111 = 65535
```

На первый взгляд кажется, что два байта позволяют представить целые числа от 0 до 65535. Но в компьютере первый бит зарезервирован для з н а к а числа: если первый бит — 0, число положительное; если же оно отрицательное, то первый бит — 1.

Следующие конфигурации битов представляют положительные целые числа:

```
00000000 00000000 = 0
00000000 00000001 = 1
.....
01111111 11111111 = 32767
```

Кажется вполне естественным, что для образования отрицательных чисел в памяти компьютера достаточно заменить первый бит вышеуказанных конфигураций на 1 (обратите внимание, что в этом случае результатом будут два представления нуля). Однако, реально это все осуществляется несколько сложнее. Компьютер работает с представлением отрицательного целого числа следующим образом:

- Он начинает с конфигурации битов в "положительной форме".
- Затем он инвертирует каждый бит (1 становится 0, а 0 становится 1).
- К младшему биту добавляется 1; игнорируется возможность переноса в старший бит.





Пример 1: представление —1

```
00000000 00000001 = +1
11111111 11111110   все биты инвертированы
11111111 11111111   к младшему биту добавлена 1
                    (переноса нет).
```

В памяти компьютера —1 имеет то же двоичное представление, что и 65535 в математике. Поэтому —1 считается двоичным дополнением 65535.

Пример 2: представление —32767

```
01111111 11111111 = 32767
10000000 00000000  инвертированные биты
10000000 00000001  добавлена 1
```

Таким образом, в двух байтах возможно представление следующих чисел:

```
00000000 00000000 = 0
00000000 00000001 = 1
.....
01111111 11111111 = 32767
11111111 11111111 = −1
11111111 11111110 = −2
.....
10000000 00000001 = −32767
10000000 00000000 = −32768
```

Итак, мы получили представление чисел от —32768 до +32767, т.е. всего 65535 различных значений.

Зачем нужно такое сложное представление?

На практике такое представление чисел с использованием двоичного дополнительного кода предоставляет два преимущества:

(1) Одно представление нуля (+0 и −0 записываются в виде 16-ти нулевых битов)
(2) Операция, состоящая в получении двоичного дополнительного кода обратима.

Для понимания того, что представляет 11111111 11111111, заметьте, что в старшем бите 1 (следовательно, представленное число отрицательно). Затем вычислите дополнительный код, который будет равен 00000000 00000001. Представлена −1.

(3) Не требуется специальных схем для вычитания:
Для вычитания используются схема получения дополнительного кода и схема сложения.

Пример 1: как компьютер получает результат операции 256 − 8

```
256 = 00000001 00000000
  8 = 00000000 00001000
```

Двоичный дополнительный код этой конфигурации:

```
−8 = 11111111 11111000
```

Компьютер получает сумму

```
  00000001 00000000
+ 11111111 11111000
-------------------
00000000 11111000 = 248
```





Пример 2: вычисление 8 − 256

```
   8 = 00000000 00001000
 256 = 00000001 00000000
-256 = 11111111 00000000
       00000000 00001000
     + 11111111 00000000
     -------------------
       11111111 00001000
```

Для получения значения этого отрицательного числа вычисляется его дополнительный код, который равен

```
00000000 11111000 = 248
```

В результате вычитания действительно получается −248.

## Д-5. Хранение целых чисел

Как упоминалось ранее, целое число кодируется в двух байтах. Перый байт называется старшим; второй - младшим. Процессор MSX (Z80) фактически записывает младший байт перед старшим. Когда целочисленная переменная получает значение, процессор записывает в памяти следующие 5 байтов данных:

1. Число 2, задающее параметр VALTYPE, которое означает, что переменная является целочисленной (его значение укладывается в два байта).
2. Код ASCII первого символа имени переменной.
3. Код ASCII второго символа имени (0, если имя состоит из одного символа).
4. Младший байт значения.
5. Старший байт значения.

Адрес четвертого байта (младшего байта значения) представляется в MSX Бейсике переменной VARPTR. Кроме того, содержание байта с известным адресом может быть прочитано функцией PEEK.
Все вышесказанное может быть подтверждено с помощью этих двух функций.
Введите следующие команды:

```
NEW
A% = 356
?HEX$(VARPTR(A%))
```

Компьютер должен вывести 8006. Это соответствует шестнадцатиричному адресу младшего байта.

Введите

```
?PEEK(&H8003)
```

Этот адрес соответствует байту VALTYPE. Поэтому вы должны получить 2.

Введите

```
?PEEK(&H8004)
```

Этот адрес соответствует первому символу имени переменной.

Вы должны получить 65, что является кодом ASCII символа "A". Следующий адрес, 8005(H), должен дать 0, поскольку имя переменной состоит лишь из одного символа.





Введите

```
? PEEK(&H8006)
```

Этот адрес, данный функцией VARPTR, соответствует младшему байту значения.
Должно быть получено 100.

Введите

```
? PEEK(&H8007)
```

Этот адрес сответствует старшему байту значения. Должно быть получено 1.
Перепишем два последних значения в двоичном виде по 8 битов каждое.

```
100  = 01100100
  1  = 00000001
```

Принимая во внимание инверсию порядка байтов, результатом будет

```
00000001 01100100 = 356
```

Можно повторить все эти процедуры с AB% = −256.
Если вводится NEW, адреса вновь будут в диапазоне от 8003(H) до 8007(H). Если NEW не вводится, переменная AB% будет занимать следующие адреса, от 8008(H) до 800C(H), и адресом, возвращенным VARPTR(AB%), будет 800B(H).

### Д-6. Двоично-десятичная система счисления

Двоично-десятичная система очень удобна для представления чисел одинарной или двойной точности. В версиях Бейсика, не имеющих этого типа представления, обычно трудно переходить от одного типа точности к другой.
Для кодирования целого числа в этой системе, двоичная конфигурация каждого десятичного разряда записывается в 4 битах.

Пример: запись 5973 в двоично-десятичной системе

```
5    9    7    3
0101 1001 0111 0011
01011001 01110011(BCD) = 5973
```

* BCD (Binary Coded Decimal) - двоично-десятичное представление

Это представление имеет одно необычное свойство:
Если конфигурацию битов двоично-десятичного числа рассматривать в качестве обычного двоичного представления, полученное значение будет, конечно, отличаться. В вышеприведенном случае мы получим

01011001 01110011(B) = 22899(D)

Если же число 22899 перевести в шестнадцатиричное, результатом будет

22899(D) = 5973(H)

Как видно, этот результат соответствует первоначальной десятичной записи этого числа.





### Д-7. Хранение чисел одинарной и двойной точности.

Введите

```
NEW
A! = 22.5E + 4
? HEX$(VARPTR(A!))
```

Возвращенным значением должно быть 8006(H).
Если прочитать адреса с 8003(H) по 8009(H), обнаружится:

```
8003(H)4  VALTYPE переменной одинарной
          точности (ее значение занимает 4 байта)
8004(H)65 код ASCII для "A"
8005(H)0  второго символа нет
8006(H)70 = 01000110(B)
8007(H)34 = 00100010(B) = 22(BCD)
8008(H)80 = 01010000(B) = 50(BCD)
8009(H)0  = 00000000(B) = 00(BCD)
```

В трех последних байтах узнается число 225.
Поскольку введено 22.5E+6, то значение может быть также записано как 225000 или .2250E+6 (различные комбинации мантиссы и порядка).
Три последних адреса содержат мантиссу, записанную в двоичнодесятичном виде (.2250). Десятичная точка (в русскоязычной записи запятая) обычно предшествует мантиссе, поэтому нет необходимости хранить ее представление в памяти.
Адрес 8006(H) содержит экспоненту (порядок), которая кодируется следующим образом:

```
01 | 000000 = 64  экспонента = 0
01 | 000001 = 65  экспонента = 1
-----------------------------------
01 | 111111 = 127 экспонента = 63
00 | 111111 = 63  экспонента = -1
00 | 111110 = 62  экспонента = -2
-----------------------------------
00 | 000000 = 0   экспонента = 63
```

Отсюда можно сделать вывод, что экспонента задается следующей формулой:

Экспонента = 64 - двоичное значение 7 последних битов

Тогда какова функция первого бита байта экспоненты? Совершенно верно! Он содержит знак мантиссы.

```
0 ~ +
1 ~ -
```

Мантисса числа одинарной точности занимает 3 байта, которые позволяют закодировать 6 разрядов в двоично-десятичном виде. Числа двойной точности хранятся таким же образом. VALTYPE равен 8, и 14-разрядная мантисса кодируется в 7 байтах. В следующей таблице суммировано все вышесказанное о хранении числовых переменных.





| точности | VALTYPE | число | байта | Значение/система счисления |
|---|---|---|---|---|
| Целое | 2 | 5 | -3 | VALTYPE = 2/двоичная |
| | | | -2 | Первый символ имени/ASCII |
| | | | -1 | второй символ имени/ASCII |
| | | | 0 - 1 | Значение/двоичная; двоичный дополнительный код для отрица-тельных чисел; инвертированный порядок байтов |
| Одинарной | 4 | 7 | -3 | VALTYPE = 4/двоичнаяточности |
| | | | -2 | Первый символ имени/ASCII |
| | | | -1 | второй символ имени/ASCII |
| | | | 0 | Экспонента и знак/знак в первом бите, фактическая экспонента + 64 в последних 7 битах |
| | | | 1 - 3 | Мантисса/двоично-десятичная |
| Двойной | 8 | 11 | -3 | VALTYPE = 8/двоичнаяточности |
| | | | -2 | Первый символ имени/ASCII |
| | | | -1 | второй символ имени/ASCII |
| | | | 0 | Экспонента и знак/знак в первом бите; фактическая экспонента + 64 в последних 7 битах |
| | | | 1 - 7 | Мантисса/двоично-десятичная |

(*) Номера байтов учитываются при вычислении адресов с помощью VARPTR

# ПОДПРОГРАММЫ BIOS

## ПРИЛОЖЕНИЕ К

Подпрограммы BIOS служат в основном для вызова подпрограмм из постоянной памяти (ПЗУ).

Программа, написанная на машинном языке, не должна содержать прямых обращений к ячейкам ПЗУ, поскольку распределение подпрогграмм в ПЗУ может меняться. Стандартом MSX, однако, гарантируется неизменное положение и определение всех подпрограмм, включенных в BIOS.
Ниже приведен полный список этих подпрограмм.

Формат:

Адрес точки входа (HEX) Имя подпрограммы
признак *n (см. ниже)

Определение:

| | |
|---|---|
| **IN** | Регистр(ы) и параметр(ы) (вход) |
| **OUT** | Регистр(ы) и параметр(ы) (выход) |
| **MOD** | Список измененных регистров |

Содержание или адрес точки входа в ПЗУ некоторых подпрограмм в стандарте MSX-2 модифицированы. Это следует учитывать авторам программных средств для КУВТ "Ямаха" поставки 1985 года.

Признак *n может принимать следующие значения:

- *1 Неизмененная подпрограмма
- *2 Использует расширение ПЗУ в SCREEN 5–8
- *3 Использует расширение ПЗУ
- *4 Подпрограмма модифицирована в SCREEN 4–8

## E-1. Интерпретатор, управление слотами, аппаратные прерывания

ВХОД — ФУНКЦИЯ

**000 — CHK RAM *1**

Проверяет ОЗУ (RAM) и устанавливает слот для командной области. Для дальнейшей инициализации должен быть сделан переход на INIT.

| | |
|---|---|
| IN | Ничего |
| OUT | Ничего |
| MOD | Все регистры |

**0008 — SYNCHR *1**

Сравнивает символ, на который указывает [HL] с символом, следующим за RST; если они равны, осуществляет переход на CHRGTR; в противном случае выдает сообщение о синтаксической ошибке (Syntax error).

| | | |
|---|---|---|
| IN | [HL] | адрес проверяемого символа |
| OUT | [HL] | адрес следующего символа |
| | [A] | символ |
| Флаг | C = 1 | если символ является цифрой |
| Флаг | Z = 1 | если конец оператора |
| MOD | [AF] | |
| | [HL] | |

**000C — RDSLT *1**

Выбирает слот в соответствии со входными параметрами и считывает содержимое слота. Прерывания отменены и заново не активируются.





| | | |
|---|---|---|
| IN | [A] | FxxxSSPP |
| | | F = 1 если указан вторичный слот |
| | | SS = номер вторичного слота (0 - 3) |
| | | PP = номер первичного слота (0 - 3) |
| | [HL] | Адрес памяти-приемника |
| OUT | [A] | содержание памяти |
| | [AF] | |
| | [BC] | |
| | [DE] | |

**0010 — CHRGTR *1**

Выбирает следующий символ из текста программы на Бейсике.

| | | |
|---|---|---|
| IN | [HL] | |
| OUT | [HL] | адрес следующего символа |
| | [A] | символ |
| Флаг | C = 1 | если символ является цифрой |
| Флаг | Z = 1 | если конец команды |
| MOD | [HL] | |
| | [AF] | |

**0014 — WRSLT *1**

Выбирает слот в соответствии со входными параметрами и записывает в него новое содержание. Прерывания отменены и заново не активируются.

| | | |
|---|---|---|
| IN | [A] | так же, как и для RDSLT |
| | [HL] | адрес памяти-приемника |
| | [E] | записываемые данные |
| OUT | Ничего | |
| | [D] | |
| MOD | [AF] | |
| | [BC] | |

0018 OUTDO *2

Выводит данные на активированное устройство (экран или принтер)

IN [A] символ
PTRFIL
PTRFLG

OUT Ничего

MOD Ничего

001C CALSLT *1

Межслотовый вызов по указанному адресу. Прерывания отменены и заново не активируются. Передача аргументов через индексные регистры IX, [IX] или [IY], невозможна.

IN [IYH] FxxxssPP (см.RDSLT)
[IX] адрес вызова

OUT ?

MOD ?

0020 DKCOMPR *1

Сравнивает [HL] с [DE].

IN [HL]
[DE]

OUT Флаги

MOD [AF]

0024 ENASLT *1

Выбирает слот в соответствии со входными параметрами и активирует слот. Прерывания отменены и не активируются заново.

IN [A] FxxxssPP (см.RDSLT)
[HL] адрес памяти-приемника

OUT Ничего

MOD Все регистры





0028 GETYPR *1

Возвращает тип FAC.

IN [FAC]

OUT Флаги

MOD [AF]

030 CALLF *1

Межслотовый вызов.

IN Ничего

OUT?

MOD?

0038 KEYINT *1

Выполняет процедуры прерывания.

IN Ничего

OUT Ничего

MOD Ничего

## E-2. Инициализация ввода-вывода

003B INITIO *1

Инициализация устройств.

IN Ничего

OUT Ничего

MOD Все регистры

003E INIFNK *1

Инициализация текста функциональных клавиш.

IN Ничего

OUT Ничего

MOD Все регистры

## Е-3. Доступ к видеопроцессору (VDP) (режим T19918)

**0041** DISSCR *1

Отключение изображения экрана (без его очистки).

IN Ничего
OUT Ничего
MOD [AF]
[BC]

**0044** ENASCR *1

Активирует изображение экрана.

IN Ничего
OUT Ничего
MOD [AF]
[BC]

**0047** WRTVDP *2

Записывает данные в регистр VDP. Вызывает расширенное ОЗУ, если бит EV регистра 0 изменен или если у Вас регистр 8-46.

IN [C] номер регистра (0-23 или 32-46)
[B] записываемое значение
OUT Ничего
MOD [AF]
[BC]

**004A** RDVRM *1

Считывает видеопамять (один байт)

IN [HL] адрес
OUT [A] считываемое значение
MOD [AF]

**004D** WRTVRM *1

Записывает один байт в видеопамять.





IN [HL] адрес
[A] записываемое значение
OUT Ничего
MOD [AF]

**0050** SETRD *1

Устанавливает регистр VDP для чтения.

IN [HL] адрес
OUT Ничего
MOD [AF]

**0053** SETWRT

Устанавливает VDP для записи.

IN [HL] адрес
OUT Ничего
MOD [AF]

**0056** FILVRM *4

Записывает один байт определенное число раз в видео память.

IN [HL] адрес
[BC] длина
[A] записываемое значение
OUT Ничего
MOD [AF]
[BC]

**0059** LDIRMV *4

Переносит блок данных из видеопамяти в ОЗУ.

IN [HL] адрес видеопамяти
[DE] адрес ОЗУ
[BC] длина
OUT Ничего
MOD Все регистры

**005C** LDIRVM *4

Переносит блок данных из ОЗУ в видеопамять.

IN [HL] адрес ОЗУ
[DE] адрес видеопамяти
[BC] длина
OUT Ничего
MOD Все регистры

**005F** CHGMOD *3

Устанавливает режим видеопроцессора в соответствии с режимом SCREEN. Для его инициализации используйте CHGMDP в расширенном ПЗУ.

IN [A] режим экрана (0-8)
OUT Ничего
MOD Все регистры

**0062** CHGCLR *1

Изменяет цвет экрана.

IN [A] режим экрана
FORCLR цвет изображения
BAKCLR цвет фона
BRDCLR цвет границы
OUT Ничего
MOD Все регистры

**0066** NMI *1

Выполняет процедуры немаскируемого прерывания.

IN Ничего
OUT Ничего
MOD Ничего





**0069** CLRSPR *3

Инициализирует все спрайты. Образы очищаются, номера спрайтов устанавливаются по значениям плоскостей спрайтов, цвет устанавливается на значение цвета фона. Вертикальная позиция устанавливается на 209 (или 217 в SCREEN 4-8).

IN SCRMOD
OUT Ничего
MOD Все регистры

**006C** INITXT *3

Инициализирует экран для SCREEN 0 (40*24). Определяет VDP. Эта подпрограмма не инициализирует палитру. Для ее инициализации используйте INITPLT в расширенном ПЗУ.

IN TXTNAM
TXTCGP
OUT Ничего
MOD Все регистры

**006F** INIT32 *3

Инициализирует экран для SCREEN 1 (32*24). Определяет VDP. Эта подпрограмма не инициализирует палитру.

IN T32NAM
T32CGP
T32COL
T32ATR
T32PAT
OUT Ничего
MOD Все регистры

**0072** INIGRP *3

Инициализирует экран для SCREEN 2. Определяет VDP. Эта подпрограмма не инициализирует палитру.

IN GRPNAM
GRPCGP
GRPCOL
GRPATR
GRPPAT
OUT Ничего
MOD Все регистры

**0075** INIMLT *3

Инициализирует экран для SCREEN 3. Определяет VDP. Эта подпрограмма не инициализирует палитру.

IN MLTNAM
MLTCGP
MLTCOL
MLTATR
MLTPAT
OUT Ничего
MOD Все регистры

**0078** SETTXT *3

Устанавливает VDP для SCREEN 0 (40*24).

IN TXTNAM
TXTCGP
OUT Ничего
MOD Все регистры

**007B** SETT32 *3

Устанавливает VDP для SCREEN 1 (32*24).





IN T32NAM
T32CGP
T32COL
T32ATR
T32PAT
OUT Ничего
MOD Все регистры

**007E** SETGRP *3

Устанавливает VDP для SCREEN 2.

IN GRPNAM
GRPCGP
GRPCOL
GRPATR
GRPPAT
OUT Ничего
MOD Все регистры

**0081** SETMLT *3

Устанавливает VDP для SCREEN 3.

IN MLTNAM
MLTCGP
MLTCOL
MLTATR
MLTPAT
OUT Ничего
MOD Все регистры

**0084** CALPAT *1

Возвращает адрес образа спрайта.

IN [A] номер спрайта
OUT [HL] адрес используемого спрайтом образа
MOD [AF]
[DE]
[HL]

0087 **CALATR *1**

Возвращает адрес атрибутов спрайта.

IN [A] номер спрайта
OUT [HL] адрес используемых спрайтом атрибутов
MOD [AF]
[DE]
[HL]

008A **GSPSIZ *1**

Возвращает размер спрайта.

IN Ничего
OUT [A] число байтов
Флаг C = 1 если 16*16
MOD [AF]

008D **GRPPTR *2**

Выводит символ на графическом экране.

IN [A] код выводимого символа или код логиче ской операции (SCREEN 5-8)
OUT Ничего
MOD Ничего

## К-4. Доступ к PSG (звукогенератору)

0090 **GICINI *1**

Инициализирует PSG и статические данные для оператора PLAY.

IN Ничего
OUT Ничего
MOD Все регистры





0093 **WRTPSG *1**

Записывает данные в регистр PSG.

IN [A] номер регистра
[E] записываемое значение
OUT Ничего
MOD Ничего

0096 **RDPSG *1**

Считывает регистр PSG.

IN [A] номер регистра
OUT [A] считываемые данные
MOD Ничего

0099 **STRTMS *1**

Проверяет и запускает фоновую музыку оператора PLAY.

IN Ничего
OUT Ничего
MOD Все регистры

## К-5. Доступ к консоли (клавиатуре и монитору)

009C **CHSNS *1**

Проверяет статус буфера клавиатуры.

IN Ничего
OUT Флаг Z = 0 если в буфере есть любой символ

009F **CHGET *1**

Ожидает нажатия любой клавиши и дает ее код.

IN Ничего
OUT [A] код символа
MOD [AF]

00A2 **CHPUT *1**

Выводит символ на монитор.

IN [A] код символа
OUT Ничего
MOD Ничего

00A5 **LPTOUT *1**

Выводит символ на принтер.

IN [A] код символа
OUT Флаг C = 1 в случае невыполнения
MOD [F]

00A8 **LPTSTT *1**

Проверяет статус принтера.

IN Ничего
OUT [A] = 255 (или 0)
Флаг Z = 0 (или 1) Принтер готов (или нет)
MOD [AF]

00AB **CNVCHR *1**

Проверяет графический заголовок и преобразует код.

IN [A] код символа
OUT Флаг C = 0графический заголовок
Флаг C = 1 и флаг Z = 1: преобразование в графический символ
Флаг C = 1 и флаг Z = 0: преобразования нет
MOD [AF]

00AE **PINLIN *1**

Принимает строку с клавиатуры до ввода RETURN или STOP и записывает ее.

IN Ничего
OUT [HL] адрес начала буфера - 1
Флаг C = 1 если STOP
MOD Все регистры





00B1 **INLIN *1**

То же, что и PINLIN, за исключением того, если AUTFLG не является нулевым.

IN Ничего
OUT [HL] адрес начала буфера - 1
Флаг C = 1 если STOP
MOD Все регистры

00B4 **QINLIN *1**

Выводит на экран ? и переключает на INLIN.

IN Ничего
OUT [HC] адрес начала буфера - 1
Флаг C = 1 если STOP
MOD Все регистры

00B7 **BREAKX**

Проверяет, не нажата ли комбинация CTRL + STOP. Работает, когда прерывания отменены.

IN Ничего
OUT Флаг C = 1 если нажата CTRL + STOP
MOD [AF]

00BA **ISCNTC *1**

Проверяет, не нажата ли комбинация CTRL + STOP.

IN Ничего
OUT Ничего
MOD Ничего

00BD **CKCNTC *1**

То же, что и ISCNTC, но используется MSX Бейсиком.

IN Ничего
OUT Ничего
MOD Ничего

**00C0** BEEP *3

Формирует звуковой сигнал.

| | |
|---|---|
| IN | Ничего |
| OUT | Ничего |
| MOD | Все регистры |

**00C3** CLS *3

Очищает экран.

| | |
|---|---|
| IN | Флаг Z = 1 |
| OUT | Ничего |
| MOD | [AF]<br>[BC]<br>[DE] |

**00C6** POSIT *1

Локализует курсор.

| | |
|---|---|
| IN | [H] колонка<br>[L] строка |
| OUT | Ничего |
| MOD | [AF] |

**00C9** FNKSB *1

Проверяет, активирован ли вывод текста функциональных клавиш на экран.

| | |
|---|---|
| IN | FNKFLG (вывод, если < > 0) |
| OUT | Ничего |
| MOD | Все регистры |

**00CC** ERAFNK *1

Отменяет вывод текста функциональных клавиш.

| | |
|---|---|
| IN | Ничего |
| OUT | Ничего |
| MOD | Все регистры |





**00CF** DSPFNK *2

Активирует вывод текста функциональных клавиш.

| | |
|---|---|
| IN | Ничего |
| OUT | Ничего |
| MOD | Все регистры |

**00D2** TOTEXT *1

Переводит экран в режим 0 или 1 в конце программы или после ошибки. Эта подпрограмма не изменялась, но поскольку она вызывает CHGMDP, используется расширенное ПЗУ.

| | |
|---|---|
| IN | Ничего |
| OUT | Ничего |
| MOD | Все регистры |

**00D5** GTSTCK *1

Возвращает статус джойстика.

| | | |
|---|---|---|
| IN | [A] | Номер джойстика (0-2) |
| OUT | [A] | Направление (0-8) |
| MOD | Все регистры | |

**00D8** GTTRIG *1

Возвращает статус кнопки триггера.

| | | |
|---|---|---|
| IN | [A] | Номер кнопки (0-4) |
| OUT | [A] | 255 (или 0) нажата (или не нажата) |
| MOD | [AF] | |

**00DB** GTPAD *1

Возвращает статус клавиши на графическом планшете PAD.

| | | |
|---|---|---|
| IN | [A] | Номер |
| OUT | [A] | Значение |
| MOD | Все регистры | |

00DE GTPDL *2
Возвращает статус палитры MSX.
IN [A] номер
OUT [A] статус
MOD Все регистры

### Е-6. Доступ к кассетной ленте

00E1 TAPION *1
Включает мотор и считывает заголовок с ленты
IN Ничего
OUT Флаг C = 1 в случае невыполнения
MOD Все регистры

00E4 TAPIN *1
Читает ленту (один байт)
IN Ничего
OUT [A] данные
Флаг C = 1 если работа прервана
MOD Все регистры

00E7 TAPIOF *1
Останавливает чтение с ленты (останавливается мотор)
IN Ничего
OUT Ничего
MOD Ничего

00EA TAPOON *1
Включает мотор и записывает заголовок.
IN [A] < > 0 (0) для записи длинного заголовка (короткого)
OUT Флаг C = 1 если работа прервана
MOD Все регистры





00ED TAPOUT *1
Записывает на ленту (один байт).
IN [A] записываемые данные
OUT Флаг C = 1 если работа прервана
MOD Ничего

00F0 TAPOOF *1
Останавливает запись на ленту (останавливает мотор)
IN Ничего
OUT Флаг C = 1 если прервано
MOD Ничего

00F3 STMOTR *1
Управляет мотором.
IN [A] 0 (1) для остановки (включения); если [A] содержит 255, перевернуть кассету.
OUT Ничего
MOD [AF]

### Е-7. Обработка очередей (для оператора PLAY)

00F6 LFTQ *1
Возвращает количество байтов, оставшихся в очереди.
IN
OUT
MOD

00F9 PUTQ *1
Помещает байт в очередь.
IN
OUT
MOD

## Е-8. Подпрограммы, используемые модулями GENGRP и ADVGRP

Примечание:

Подпрограммы от RIGHTC до TDOWNC вызывают расширенное ПЗУ, если режим экрана - не высокого разрешения

**00FF** LEFTC *2
Сдвигает на один пиксель влево (GRACX уменьшается).
IN
OUT
MOD

**0102** UPC *2
Сдвигает на один пиксель вверх (GRACY уменьшается).
IN
OUT
MOD

**0105** TUPC *2
Сдвигает на один пиксель вверх (GRACY уменьшается).
IN
OUT
MOD

**0108** DOWNC *2
Сдвигает на один пиксель вниз (GRACY наращивается).
IN
OUT
MOD





**010B** TDOWNC *2
Сдвигает на один пиксель вниз (GRACY наращивается).
IN
OUT
MOD

**010E** SCALXY *2
Масштабирует координаты XY.

| | | |
|---|---|---|
| IN | [BC] | X |
| | [DE] | Y |
| OUT | [BC] | модуль ширины X |
| | [DE] | модуль высоты Y |

**0111** MAPXY
Соотносит координаты с физическим адресом.
Вызывает расширенное ПЗУ, если Вы в режиме SCREEN 3.

| | | |
|---|---|---|
| IN | [BC] | X |
| | [DE] | Y |
| OUT | [HL] | адрес видеопамяти |
| | [A] | маска |

**0114** FETCHC *1
Выдает текущий физический адрес и конфигурацию маски.

| | | |
|---|---|---|
| IN | Ничего | |
| OUT | [HL] | адрес |
| | [A] | маска |
| MOD | [A] | |
| | [HL] | |

**0117** STOREC *1
Записывает физический адрес и маску.

| | | |
|---|---|---|
| IN | [HL] | адрес |
| | [A] | маска |
| OUT | Ничего | |

**011A** SETATR *4
Устанавливает байт атрибута.
IN
OUT
MOD

**011D** READC *2
Считывает атрибут текущего пикселя. Вызывает расширенное ПЗУ, если SCREEN > = 3.
IN
OUT
MOD

**0120** SETC *2
Устанавливает текущий пиксель в соответствии со значе нием указанного атрибута (ATTRBYT).
Вызывает расширен ное ОЗУ, если Вы в SCREEN 3.
IN
OUT
MOD

**0126** GTASPC *1
Возвращает коэффициент сжатия (CIRCLE).
IN Ничего
OUT [DE]
[HL]
MOD [DE]
[HL]

**0129** PNTINI *1
Инициализирует оператор PAINT.
IN
OUT
MOD





**012C** SCANR *2
Сканирует пиксели слева направо (PAINT).
Вызывает расширенное ПЗУ, если SCREEN > = 3.
IN
OUT
MOD

**012F** SCANL *
Сканирует пиксели справа налево (PAINT).
Вызывает расширенное ПЗУ, если SCREEN > = 3.
IN
OUT
MOD

## Е-9. Дополнительные подпрограммы

**0132** CHGCAP *1
Включает и выключает индикатор CAPS.
IN [A] 0 (или < > 0) для выключения (включения)
OUT Ничего
MOD [AF]

**0135** CHGSND *1
Изменяет бит 7 порта C PSG.
IN [A] 0 (или < > 0) для выключения (включения)
OUT Ничего
MOD [AF]

**0138** RSLREG *1
Считывает текущие данные, выводимые в первичный слот (порт A)
IN Ничего
OUT [A] считанное значение
MOD [A]

013B WSLREG *1

Записывает в регистр первичного слота (порт A)

IN [A]
OUT Ничего
MOD Ничего

013E RDVDP *1

Считывает регистр статуса VDP.

IN Ничего
OUT [A] Считанное значение
MOD [A]

0141 SNSMAT *1

Возвращает статус указанной строки матрицы клавиатуры (записывает в порт C и считывает порт D).

IN [A] номер строки
OUT [A] считываемое значение (соответствующий нажатой клавише бит сбрасывается на 0).
MOD [AF]
[C]

0144 PHYDIO *1

Выполняет операции на устройстве внешней памяти (дисковод). В минимальной конфигурации обеспечивается только одна ловушка.

IN
OUT
MOD

0147 FORMAT *1

Выполняет инициализацию на устройстве внешней памяти. В минимальной конфигурации обеспечивается только одна ловушка.





IN
OUT
MOD

014A ISFLIO *1

Проверяет, выполняет ли устройство операции ввода-вывода.

IN Ничего
OUT [A] 0 (или < > 0) если нет операции (есть)
MOD [AF]

014D OUTDLP *1

Вывод на LPT. Эта подпрограмма отличается от LPTOUT следующим: TAB преобразуется в пробел; при прерывании осуществляется переход на сообщение Device I/O error (Ошибка устройства ввода-вывода).

IN [A] код
OUT Ничего
MOD [AF]

0150 GETVCP *1

Проигрывает музыку в режиме фоновой задачи.

IN [A] номер канала (0, 1, 2)
OUT
MOD

0153 GETVC2 *1

Проигрывает музыку в режиме фоновой задачи.

IN
OUT
MOD

**0156** **KILBUF *1**

Очищает буфер клавиатуры.

IN
OUT
MOD [HL]

**0159** **CALBAS *1**

Выполняет межслотовый вызов интерпретатора MSX Бейсика.

IN [IX] адрес
OUT?
MOD?

**015C** **SUBROM**

Выполняет межслотовый вызов SUBROM.

IN [IX] записанный в стеке адрес
OUT ?
MOD индексные регистры [IX] и [IY] зарезервированы

**015F** **EXTROM**

Выполняет межслотовый вызов SUBROM.

IN [IX]
OUT ?
MOD индексные регистры [IX] и [IY] зарезервированы

**0162** **CHKSLZ**

Сканирует слот из SUBROM.

IN Ничего
OUT Ничего
MOD Все





**0165** **CHEKNEW**

Проверяет режим экрана.

IN Ничего
OUT Флаг переноса сбрасывается в SCREEN 5-8
MOD [AF]

**0168** **EOL**

Стирает до конца строки.

IN Номер колонки в [H], номер строки в [L]. Курсор должен остаться неизмененным.
OUT Ничего
MOD Все

**016B** **BIGFIL**

То же, что и FILVRM, за исключением следующего: FILVRM проверяет, является ли текущим режимом экрана 0, 1, 2 или 3. Если это так, он ведет себя так же, как VDP, имеющий видеопамять только 16 К. Это необходимо для сохранения совместимости с MSX1. BIGFIL, однако, не проверяет режим экрана и заполняет видеопамять так, как определено параметрами.

IN То же, что и FILVRM
OUT То же, что и FILVRM
MOD То же, что и FILVRM

**016E** **NSETRD**

Устанавливает VDP для чтения.

IN Адрес в [HL]: действительны все биты.
OUT Ничего
MOD [AF]

**0171** **NSTWRT**

Устанавливает VDP для записи.

IN Адрес в [HL]: действительны все биты.
OUT Ничего
MOD [AF]

0174 NRDVRM
Считывает видеопамять, адресованную посредством

[HL]:действительны все биты.
IN Адрес в [HL]: действительны все биты.
OUT Данные в [A]
MOD [F]

0177 NWRVRM
Записывает [A] в видеопамять, адресованную посредством [HL]: действительны все биты
IN Адрес в [HL], данные в [A]
OUT Ничего
MOD [AF]

## Е-10. РАСШИРЕННОЕ ПЗУ

Как вызвать расширенное ПЗУ.

```
        LD      IX, INIPLT
        CALL    EXTROM
                ;Возврат сюда или
INIPAL:
        PUSH    IX
        LD      IX, INIPLT
        JP      SUBROM  ;Возвращает вызывающая
                        INIPAL программа или
        LD      IY, (EXBRASA-1);получает адрес слота
                расширенного ПЗУ
        LD      IX, INIPLT
        CALL    CALSLT
```





### Е-10-1. Работа с графическими средствами Бейсика

0069 PAINT*5
Выполняет закраску. Для режимов экрана 5, 6, 7 или 8.
IN [HL] содержит текстовый указатель на внутреннее представление оператора (токен) Бейсика
OUT [HL] содержит обновленный текстовый указатель
MOD Все

006D PSET*5
Устанавливает точку. Для режимов экрана 5, 6, 7 или 8.
IN [HL] как в предыдущем случае
OUT [HL] как в предыдущем случае
MOD Все

0071 ATRSCN*5
Сканирует атрибут цвета. Для режимов экрана 5, 6, 7 или 8.
IN [HL] как в предыдущем случае
OUT [HL] как в предыдущем случае
MOD Все

0075 GLINE*5
Рисует линию. Для режимов экрана 5, 6, 7 или 8.
IN [HL] как в предыдущем случае
OUT [HL] как в предыдущем случае
MOD Все

0079 DOBOXF*5
Рисует закрашенный квадрат. Для режимов экрана 5, 6, 7 или 8.

IN [HL] как в предыдущем случае
Начальная координата ([BC],[DE]).
Конечная координата (GXPOS,GYPOS).
OUT [HL] как в предыдущем случае
MOD Все

007D DOLINE*5
Рисует линию. Для режимов экрана 5, 6, 7 или 8.
IN [HL] как в предыдущем случае
Начальная координата ([BC],[DE]).
Конечная координата (GXPOS,GYPOS).
OUT [HL] как в предыдущем случае
MOD Все

0081 BOXLIN*5
Рисует квадрат. Для режимов экрана 5, 6, 7 или 8.
IN [HL] как в предыдущем случае
Начальная координата ([BC],[DE]).
Конечная координата (GXPOS,GYPOS).
OUT [HL] как в предыдущем случае
MOD Все

F-10-2. Графика нижнего уровня

0085 DOGRPH
Рисует линию. Для режимов экрана 5, 6, 7 или 8.
IN Начальная координата ([BC],[DE]).
Конечная координата (GXPOS,GYPOS).Атрибут в (ATRBYT)
Код логической операции (LOGOPR)
OUT Ничего
MOD [AF]

0089 GRPPTR
Выводит символ на графическом экране. Для режимов экрана 5, 6, 7 или 8.





IN Код для выхода в
[A] Атрибут в (ATRBYT) Код логической операции (LOGOPR)
OUT Ничего
MOD Ничего

008D SCALXY
Масштабирует координаты X Y.
IN Горизонтальная позиция [BC],
вертикальная позиция
[DE]
[DE]
OUT Усеченная горизонтальная позиция [BC],
усеченная вертикальная позиция [DE].
MOD [AF]

0091 MAPXYC
Соотносит координату с физическим адресом. Для режимов экрана 3, 5, 6, 7 или 8.
IN Координата ([BC],[DE]).
OUT Режим экрана 3. Адрес видеопамяти [HL] и (CLOC). Маска битов [A] и (CMASK). Режим экрана 5, 6, 7 или 8. Горизонтальная позиция [HL] и (CLOC). Вертикальная позиция [A] и (CMASK).
MOD [F]

0095 READC
Считывает атрибут текущего пикселя. Для режимов экрана 3, 5, 6, 7 или 8.
IN Координата (CLOC) и (CMASK).
OUT Атрибут [A]
MOD [AF]

0099 **BYTATR**

Устанавливает байт атрибута.

IN Атрибут [A]

OUT Устанавливается флаг переноса, если атрибут недействителен.

MOD [F]

009D **BYTC**

Устанавливает текущий пиксель на указанный атрибут. Для режимов экрана 3, 5, 6, 7 или 8.

IN Координата (CLOC) и (CMASK).
Атрибут (ATRBYT).

OUT Ничего

MOD [AF]

00A1 **TRIGHT**

Сдвигает на один пиксель вправо. Только для режима экрана 3..

IN Координата (CLOC) и (CMASK)

OUT Измененная координата (CLOC) и (CMASK). Устанавливается бит переноса, если указанная координата находится на границе экрана.

MOD [AF]

00A5 **RIGHTC**

Сдвигает на один пиксель вправо. Только для режима экрана 3.

IN Координата (CLOC) и (CMASK).

OUT Измененная координата (CLOC) и (CMASK).

MOD [AF]

00A9 **TLEFTC**

Сдвигает на один пиксель влево. Для режимов экрана 3, 5, 6, 7, или 8.





IN Координата (CLOC) и (CMASK).

OUT Измененная координата (CLOC) и (CMASK). Устанавливается флаг переноса, если координата нахо дится на границе экрана.

MOD [AF]

00AD **LEFTC**

Сдвигает на один пиксель влево. Только для режима экрана 3.

IN Координата (CLOC) и (CMASK).

OUT Измененная координата (CLOC) и (CMASK).

MOD [AF]

00B1 **TDOWNC**

Сдвигает на один пиксель вниз. Для режимов экрана 3, 5, 6, 7, или 8.

IN Координата (CLOC) и (CMASK).

OUT Измененная координата (CLOC) и (CMASK). Устанавливается бит переноса, если указанная координата находится на границе экрана.

MOD [AF]

00B5 **DOWNC**

Сдвигает на один пиксель вниз. Только для режима экрана 3.

IN Координата (CLOC) и (CMASK).

OUT Измененная координата (CLOC) и (CMASK). Устанавливается бит переноса, если указанная координата находится на границе экрана.

MOD [AF]

00B9 **TUOC**

Сдвигает на один пиксель вверх. Для режимов экрана 3, 5, 6, 7, или 8.

IN Координата (CLOC) и (CMASK).
OUT Измененная координата (CLOC) и (CMASK). Устанавливается бит переноса, если указанная координата находится на границе экрана.
MOD [AF]

00BD **UPC**

Сдвигает на один пиксель вверх. Только для режима экрана 3.

IN Координата (CLOC) и (CMASK).
OUT Измененная координата (CLOC) и (CMASK).
MOD [AF]

00C1 **SCANR**

Сканирует пиксели слева направо. Для режимов экрана 3, 5, 6, 7, или 8.

IN Флаг задержки в [B], счетчик границы в [DE].
OUT Счет границы в [DE], флаг изменения пикселя в [C].
MOD Все

00C5 **SCANL**

Сканирует пиксели справа налево. для режимов экрана 5-8 и в многоцветном режиме.

IN Счетчик границы в [DE].
OUT Счетчик границы в [DE], флаг изменения пикселя в [C].
MOD Все





00C9 **NVBXLN**

Рисует квадрат. Для режимов экрана 5, 6, 7 или 8.

IN Начальная координата ([BC] , [DE]). Конечная координата (GXPOS , GYPOS). Атрибут в (ATRBYT) Код логической операции (LOGOPR)
OUT Ничего
MOD Все

00CD **NVBXFL**

Рисует закрашенный квадрат. Для режимов экрана 5, 6, 7 или 8.

IN Начальная координата ([BC],[DE]). Конечная координата (GXPOS,GYPOS). Атрибут в (ATRBYT) Код логической операции (LOGOPR).
OUT Ничего
MOD Все

## E-10-3. Доступ к VDP (видеопроцессору 9938)

00D1 **CHGMOD**

Устанавливает режим VDP в соответствии с SCRMOD.

IN [A] Режим экрана (0-8)
OUT Ничего
MOD Все

00D5 **INITXT**

Инициализирует экран для текстового режима (40*24), устанавливает VDP.

IN TXTNAM, TXTCGP.
OUT Ничего
MOD Все

**00D9** **INIT32**
Инициализирует экран для текстового режима (32*24), устанавливает VDP.
IN T32NAM, T32CGP, T32COL, T32ATR, T32PAT
OUT Ничего
MOD Все

**00DD** **INIGRP**
Инициализирует экран для режима высокого разрешения, устанавливает VDP.
IN GRPNAM, GRPCGP, GRPCOL, GRPATR, GRPPAT
OUT Ничего
MOD Все

**00E1** **INIMLT**
Инициализирует экран для многоцветного режима. Устанавливает VDP.
IN MLTNAM, MLTCGP, MLTCOL, MLTATR, MLTPAT
OUT Ничего
MOD Все

**00E5** **SETTXT**
Устанавливает VDP для текстового (40*24) режима.
IN TXTNAM, TXTCGP.
OUT Ничего
MOD Все

**00E9** **SETT32**
Устанавливает VDP для текстового (32*42) режима.
IN T32NAM, T32CGP, T32COL, T32ATR, T32PAT
OUT Ничего
MOD Все





**00ED** **SETGRP**
Устанавливает VDP для режима высокого разрешения.
IN GRPNAM, GRPCGP, GRPCOL, GRPATR, GRPPAT
OUT Ничего
MOD Все

**00F1** **SETMLT**
Устанавливает VDP для многоцветного режима.
IN MLTNAM, MLTCGP, MLTCOL, MLTATR, MLTPAT
OUT Ничего
MOD Все

**00F5** **CLRSPR**
Инициализирует все спрайты. Образы заполняются нулями, имена спрайтов устанавливаются по номерам плоскостей спрайтов, цвета спрайтов устанавливаются на цвет фона, вертикальные позиции устанавливаются на 217.
IN (SCRMOD)
OUT Ничего
MOD Все

**00F9** **CALPAT**
Возвращает адрес таблицы образов спрайтов.
IN Идентификатор спрайта в [A].
OUT Адрес в [HL].
MOD [AF], [DE], [HL].

**00FD** **CALATR**
Возвращает адрес таблицы атрибутов спрайтов. Эта под программа эквивалентна подпрограмме в BIOS MSX-1.

IN Идентификатор спрайта в [A].
OUT Адрес в [HL].
MOD [AF], [DE], [HL].

**0101 GSPSIZ**

Возвращает текущий размер спрайта. Эта подпрограмма
эквивалентна подпрограмме вBIOS MSX-1.
IN Ничего
OUT Размер спрайта (число байтов) в [A].
Устанавливается перенос, если используется спрайт 16*16, в противном случае сброс.
MOD [AF]

**0105 GETPAT**

Возвращает образ символа. Эта подпрограмма эквивалентна подпрограмме в BIOS MSX-1.
IN Код ASCII символа в [A].
OUT Образ символа в (PATWRK)
MOD Все

**0109 WRTVRM**

Записывает в видеопамять, адресованную через [HL]. Поддерживает адреса 0-0FFFFH.
IN Адрес в [HL], данные в [A].
OUT Ничего
MOD [AF]

**010D RDVRM**

Считывает видеопамять, адресованную через [HL]. Поддерживает адреса 0-0FFFFH.
IN Адрес в [HL].
OUT Данные в [A].
MOD [AF]





**0111 CHGCLR**

Изменяет цвет экрана
IN Режим в [A].
Цвет изображения в FORCLR.
Цвет фона в BAKCLR.
Цвет границы в BRDCLR.
OUT Ничего
MOD Все

**0115 CLS**

Очищает экран.
IN Ничего
OUT Ничего
MOD Все

**0119 CLRTXT**

Очищает текстовый экран.
IN Ничего
OUT Ничего
MOD Все

**011D DSPFNK**

Отображает функциональные клавиши.
IN Ничего
OUT Ничего
MOD Все

**0121 DELLN0**

Удаляет строку в текстовом режиме.
IN Номер строки в [L].
OUT Ничего
MOD Все

0125 **INSLNO**
Вставляет строку в текстовом режиме.
IN Номер строки в [L].
OUT Ничего
MOD Все

0129 **PUTVRM**
Вставляет символ в текстовом экране.
IN Номер колонки в [H], номер строки в [L].
OUT Ничего
MOD [AF]

012D **WRTVDP**
Записывает в регистр VDP.
IN Регистр в [C], данные в [B].
OUT Ничего
MOD [AF], [BC]

0131 **VDPSTA**
Считывает статус VDP.
IN Регистр статуса в [A], (0-9)
OUT Данные в [A]
MOD [F]

013D **SETPAG**
Устанавливает регистры VDP на смену страниц.
IN (ACPAGE), (DPPAGE)
OUT Ничего
MOD [AF]





### Е-10-4. Доступ к палитре

Палитра видеопроцессора имеет 3 цвета (красный, зеленый, синий). Каждый цвет имеет 3 бита для отображения интенсивности цвета. Текущая палитра записана в видеопамяти, поскольку мы не имеем возможности считывать палитру из видеопроцессора (VDP).

0141 **INIPLT**
Инициализирует палитру и видеопамять для области сохранения палитры.
IN Ничего
OUT Ничего
MOD [AF],[BC],[DE]

0145 **RSTPLT**
Восстанавливает палитру из видеопамяти.
IN Ничего
OUT Ничего
MOD [AF],[BC],[DE]

0149 **GETPLT**
Получает коды цветов из палитры.
IN Палитра в [A] (0-15).
OUT Код RED (красный) в старших 4 битах [B].
Код BLUE (синий) в младших 4 битах [B]. Код GREEN (зеленый) в младших 4 битах [C].
MOD [AF],[DE]

014D **SETPLT**
Задает коды цветов палитре.

IN Палитра в [D] (0-15).
Код RED (красный) в старших 4 битах [A].
Код BLUE (синий) в младших 4 битах [A].
Код GREEN (зеленый) в младших 4 битах [E].
OUT Ничего
MOD [AF]

### E-10-5. Операторы расширенного MSX Бейсика

**0151 PUTSPR**
Помещает спрайты на экран.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

**0155 COLOR**
Изменяет цвет экрана, цвет спрайта, палитру.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

**0159 SCREEN**
Изменяет режим экрана.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

**015D WIDTHS**
Изменяет ширину текстового экрана.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все





**0161 VDP**
Устанавливает регистр VDP.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

**0165 VDPF**
Считывает текущий регистр VDP.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

**0169 BASE**
Устанавливает регистры базы VDP.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

**016D BASEF**
Считывает регистры базы VDP.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

**0171 VPOKE**
Записывает один байт в видеопамять.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

**0175 VPEEK**
Считывает один байт из видеопамяти.
IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

0179 SETS

Устанавливает звук сигнала, настройку экрана, время и дату.

IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

### E-10-6. Разное

017D BEEP

Формирует короткий звуковой сигнал (зуммер).

IN Ничего
OUT Ничего
MOD Все

0181 PROMPT

Выводит подсказку.

IN Ничего
OUT Ничего
MOD Все

### E-10-7. Восстановление экрана

0185 SDFSCR

Восстанавливает относящиеся к экрану параметры из ОЗУ на микросхеме таймера, являющейся энергонезависимой. Не выводит текст функциональных клавиш, если перенос сброшен при обращении из MSX DOS.

IN Перенос сброшен при обращении из MSX DOS.
OUT Ничего
MOD Все





0189 SETSCR

Восстанавливает экран и выводит открывающее сообщение.

IN Ничего
OUT Ничего
MOD Все

### E-10-8. Функции передачи данных для видеопамяти

018D SCOPY

Копирует видеопамять, массивы и файлы на дискетах.

IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

0191 BLTVV

Копирует видеопамять в видеопамять.

IN [HL] = 0F562H
OUT Ничего
MOD Все

0195 BLTVM

Копирует массив в видеопамять.

IN [HL] = 0F562H
OUT Ничего
MOD Все

0199 BLTMV

Копирует видеопамять в массив.

IN [HL] = 0F562H
OUT Ничего
MOD Все

**0191** BLTVD

Копирует файл с дискеты в видеопамять.

IN [HL] = 0F562H
OUT Ничего
MOD Все

**01A1** BLTDV

Копирует видеопамять в файл на дискете.

IN [HL] = 0F562H
OUT Ничего
MOD Все

**01A5** BLTMD

Загружает данные массива из файла на дискете.

IN [HL] = 0F562H

**01A9** BLTDM

Записывает данные массива в файл на дискете.

IN [HL] = 0F562H
OUT Ничего
MOD Все

### E-10-9. Работа с манипуляторами

**01AD** NEWPAD

Считывает данные с аналогового игрового манипулятора, а также с манипуляторов типа МЫШЬ и "трекбол".

IN [A]

Коды подпрограмм работы с манипуляторами. Передаются через параметр GTPAD в заголовке BIOS (00DBH).

8 Опрос светового пера (возвращает 255, если опрос происходит нормально)
9 Возвращает координату X
10 Возвращает координату Y





11 Возвращает статус кнопки пера (255, если кнопка нажата)
12 Опрос манипулятора на разъеме 1 (всегда 255)
13 Возвращает смещение X
14 Возвращает смещение Y
15 Ничего (всегда 0)
16 Опрос манипулятора на разъеме 2 (всегда 255)
17 Возвращает смещение X
18 Возвращает смещение Y
19 Ничего (всегда 0)

OUT Значение в [A]
MOD Все

### E-10-10. Разное

**01B1** GETPUT

Обращение к подпрограммам GET TIME, GET DATE и PUT KANJI

IN указатель текста в [HL].
OUT измененный указатель текста в [HL].
MOD Все

**01B5** CHGMDP

Устанавливает режим VDP в соответствии с SCRMOD. Палитра инициализирована.

IN режим экрана в [A] (0-8).
OUT Ничего
MOD Все

**01B9** RESVL

Не используется. Зарезервирована.

IN
OUT
MOD

**Е-10-11. Доступ к микросхеме таймера**

**01F5 RDCLK**
Считывает данные таймера (астрономическое время)
**IN** [C] = адрес ОЗУ таймера
бит - 7 6 5 4 3 2 1 0
[C] = X X M1 M0 A3 A2 A1 A0
**OUT** [A] = Считываемые данные (действительны младшие 4 бита)
**MOD** [F]

**01F9 WRTCLK**
Записывает данные в таймер
**IN** [C] = адрес ОЗУ таймера
[A] = записываемые данные
бит - 7 6 5 4 3 2 1 0
[C] = X X M1 M0 A3 A2 A1 A0
**OUT** Ничего
**MOD** [F]



# РАБОЧАЯ ОБЛАСТЬ

## ПРИЛОЖЕНИЕ Ж

Рабочая область находится в ОЗУ и инициализируется при включении компьютера. Она делится на две части:

(1) Область переменных MSX Бейсика
(2) Область ловушек

**Ж-1. Область переменных MSX Бейсика**

Здесь будут перечислены только основные переменные.

| АДРЕС | ДЛИНА | СОДЕРЖАНИЕ |
|---|---|---|
| F3B3 | 20 | TXTNAM - MLTPAT<br>Адреса таблиц в SCREEN 0–3: последовательность PNT, CT, PGT, SAT, SGT; 2 байта на адрес |
| F3DB | 1 | CLICKSW<br>Отзвук клавиши (0 - нет отзвука) |
| F3DC | 1 | CSRY<br>Колонка текстового курсора |
| F3DD | 1 | CSRX<br>Строка текстового курсора |
| F3DE | 1 | CNSDFG<br>Отображение текста функциональных клавиш (0 - нет отображения) |
| F3DF | 8 | RG0SAV - RG7SAV<br>Регистры 0-7 VDP |
| F3E9 | 1 | FORCLR<br>Цвет текста |
| F3E9 | 1 | BAKCLR<br>Цвет фона |
| F3EB | 1 | BRDCLR<br>Цвет границы |

| | | |
|---|---|---|
| F3F2 | 1 | ATRBYT<br>Байт атрибута |
| F414 | 1 | ERRFLG<br>Номер последней ошибки |
| F415 | 1 | LPTPOS<br>Позиция головки принтера |
| F416 | 1 | PRTFLG<br>1 принтер MSX; 0 экран |
| F417 | 1 | NTMSXP<br>1 принтер MSX; 0 принтер, не соответствующий стандарту MSX |
| F55E | 258 | BUF<br>Буфер клавиатуры (логическая строка) |
| F663 | 1 | VALTYP<br>Указывает тип переменной в DAC |
| F672 | 2 | MEMSIZ<br>Верхний адрес памяти, доступной для MSX Бейсика |
| F674 | 2 | STKTOP<br>Верхний адрес стека |
| F676 | 2 | TXTTAB<br>Начальный адрес программы на MSX Бейсике |
| F69B | 2 | FRETOP<br>Верхний адрес строкового пространства |
| F6B3 | 2 | ERRLIN<br>Номер строки, где произошла последняя ошибка |
| F6C2 | 2 | VARTAB<br>Начальный адрес таблицы скалярных переменных |
| F6C4 | 2 | ARYTAB<br>Начальный адрес таблицы массивов переменных |





| | | |
|---|---|---|
| F6C6 | 2 | STREND<br>Верхний адрес пространства, используемого в текущий момент |
| F7F6 | 8 | DAC<br>Арифметический аккумулятор |
| F847 | 8 | ARG<br>Вторичный арифметический аккумулятор |
| F857 | 8 | RNDX<br>Последнее сгенерированное случайное число |
| F85F | 1 | MAXFIL<br>Максимальное число файлов |
| F860 | 2 | FILTAB<br>Адрес данных о файлах |
| F862 | 2 | NULBUF<br>Адрес файла 0 |
| F87D | 2 | SAVEND<br>Конец двоичных данных (BSAVE) |
| F87F | 160 | FNKSTR<br>Подсказка для функциональных клавиш |
| F91F | 3 | CGPNT<br>Адрес ОЗУ образа символа |
| F922 | 2 | NAMBS<br>Адрес PNT |
| F924 | 2 | CGPBAS<br>Адрес PGT |
| F926 | 2 | PATBAS<br>Адрес SGT |
| F928 | 2 | ATRBAS<br>Адрес SAT |
| FBB0 | 1 | ENSTOP<br>Если значение ненулевое, возможна остановка программы посредством нажатия комбинации CTRL + STOP + SHIFT + CODE |

| | | |
|---|---|---|
| FBDA | 11 | OLDKEY<br>Старый статус клавиш |
| FBE5 | 11 | NEWKEY<br>Новый статус клавиш |
| FBF0 | 40 | KEYBUF<br>Код клавиши |
| FC48 | 2 | BOTTOM<br>Начальный адрес ОЗУ |
| FC4A | 2 | HIMEM<br>Верхний адрес доступной памяти |
| FC9C | 1 | PADY |
| FC9D | 1 | PADX |
| FCA9 | 1 | CSRSW<br>Курсор включен/выключен |
| FCAA | 1 | CSTYLE<br>Код символа, используемого курсором |
| FCAB | 1 | CAPST<br>Верхний/нижний регистр |
| FCAF | 1 | SCRMOD<br>Режим экрана |
| FCB0 | 1 | OLDSCR<br>Старый режим экрана |
| FCB | 2 | GXPOS<br>Позиция X графического курсора |
| FCB5 | 2 | GYPOS<br>Позиция Y графического курсора |
| FCB7 | 2 | GRPACX<br>Значение X графического аккумулятора |
| FCB9 | 2 | GRPACY<br>Значение Y графического аккумулятора |





**Ж-2. Ловушки**

Ниже перечислены только основные ловушки. Когда возможно, указывается ключевое слово, переводящее на ловушку. Если за ним следует звездочка (*), это означает, что ловушка обрабатывается всякий раз, когда при инициализации опрашивается дисковод.

Первая строка определения указывает, когда ловушка вызывается; вторая строка указывает, как эта ловушка может использоваться.

| АДРЕС | ОПРЕДЕЛЕНИЕ/ИСПОЛЬЗОВАНИЕ |
|---|---|
| FDC7 | Подпрограмма, инициализирующая образы символов Определение альтернативного набора символов |
| FDCC | Подпрограмма кодирования клавиш Изменяет действие клавиши |
| FDD1 | Подпрограмма, присваивающая клавишам функцию Изменяет действие клавиши |
| FDEF | DSKO$* |
| FDF4 | SET* |
| FDF9 | NAME* |
| FDFE | KILL* |
| FE03 | IPL* |
| FE08 | COPY* |
| FE0D | CMD* |
| FE12 | DSKF* |
| FE17 | DSKI$* |
| FE1C | ATTR$* |
| FE21 | LSET* |
| FE26 | RSET* |
| FE2B | FIELD* |
| FE30 | MKI$* |
| FE35 | MKS$* |
| FE3A | MKD$* |
| FE3F | CVI* |
| FE44 | CVS* |

| | |
|---|---|
| FE49 | CVD• |
| FE58 | OPEN• |
| FE5D | KILL,LOAD...• |
| FE62 | CLOSE•• |
| FE67 | MERGE |
| FE6C | SAVE• |
| FE71 | SAVE• |
| FE76 | SAVE• |
| FE7B | FILES• |
| FE80 | GET,PUT• |
| FE8F | INPUT$ |
| FE99 | LOC$ |
| FE9E | LOF• |
| FEA3 | EOF• |
| FEA8 | FPOS• |
| FEB2 | Анализирует имя устройства |
| FEB7 | Имя не является подтвержденным именем устройства Определение новых имен |
| FEC1 | Имя устройства подтверждено Определение новых имен |
| FEC6 | Устройство не является дисководом |
| FECB | NEW, RUN |
| FED0 | Очищает таблицу переменных (CLEAR) |
| FEE4 | Вывод символов |
| FF4D | RETURN |
| FF52 | PRINT |
| FF7F | MID$ |
| FF84 | WIDTH |
| FF89 | LIST and LLIST |
| FF93 | POKE |
| FFC0 | SCREEN |
| FFC5 | PLAY |



# ОБРАЗЫ И КОДЫ СИМВОЛОВ



Код "A" - 41(H) = 65 (D), поэтому можно печатать "A", используя ?CHR$(65). В режимах SCREEN 0 и 1, форма "A" хранится в видеопамяти по адресу

AD + 8*ASC("A") или AD + 8*65

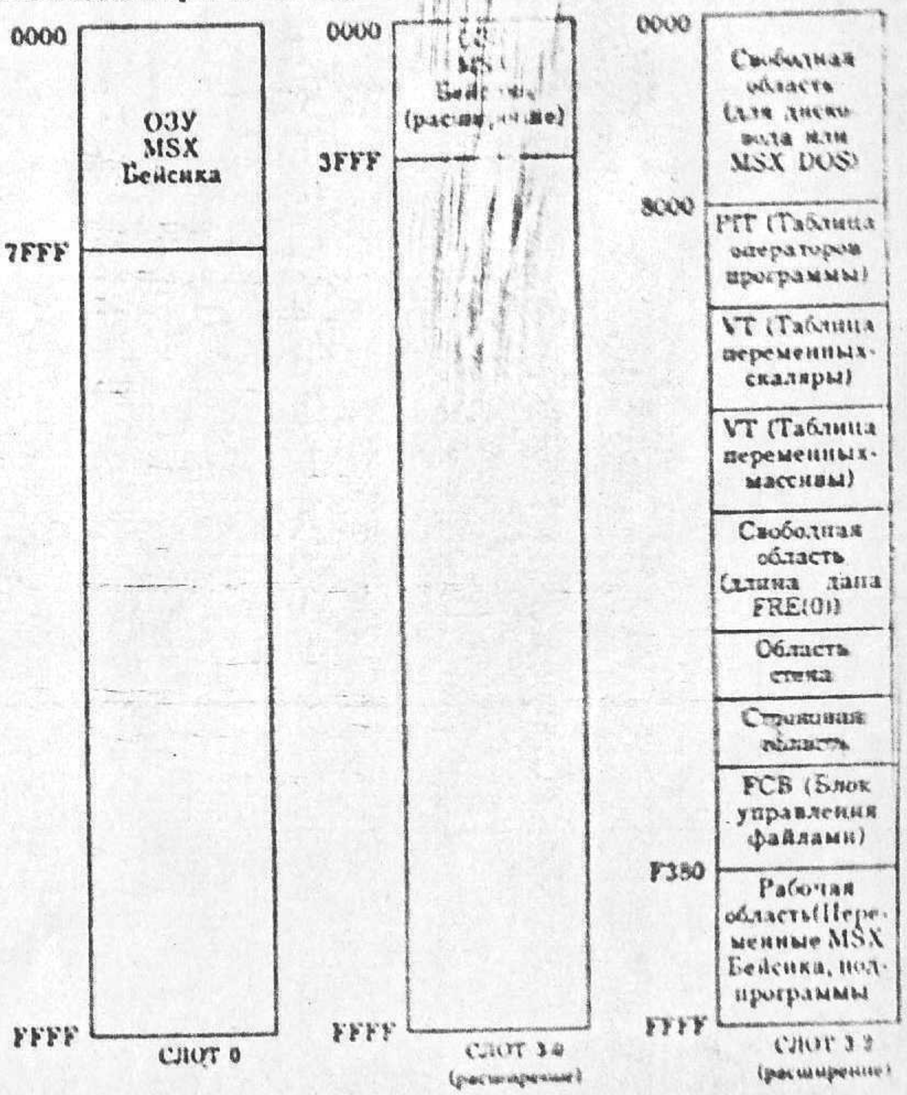
где AD является начальным адресом PGT. Можно использовать 8 операторов VPOPE для изменения формы символа.

# КАРТА ПАМЯТИ

## ПРИЛОЖЕНИЕ Н

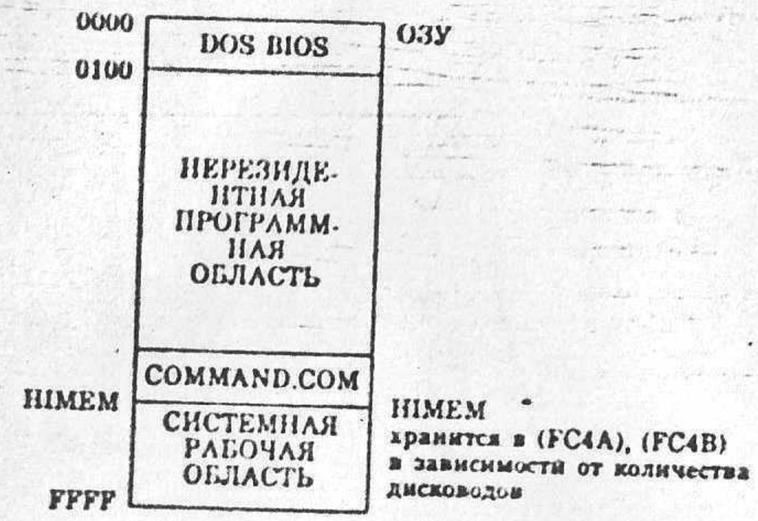
### 1. Компьютеры [MSX]-2

**Шестнадцатиричный адрес**

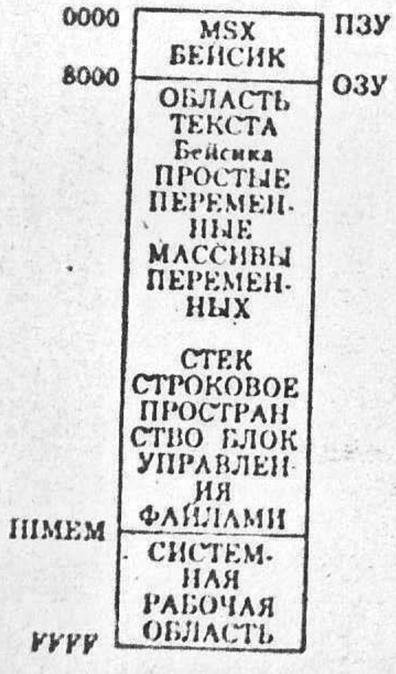


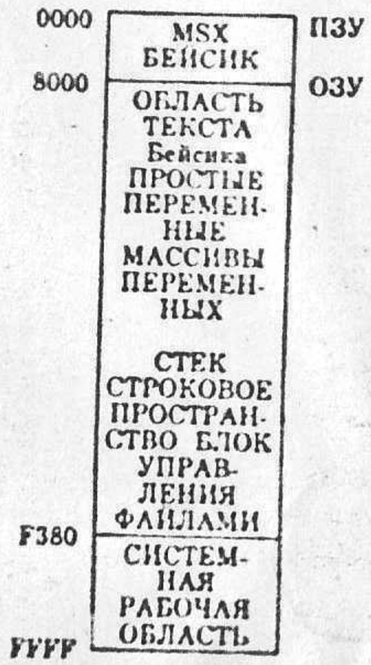
### 2. Компьютеры MSX-1

**РЕЖИМ MSX DOS (ОЗУ 64 К)**




**РЕЖИМ MSX DISK Бейсик (ОЗУ 32 К)**






# ИСПОЛЬЗОВАНИЕ ДИСКА "ОЗУ"(ВИРТУАЛЬНОГО ДИСКА) (ТОЛЬКО ДЛЯ MSX-2)



Свободное пространство памяти, определяемое функциями **FRE(0)** и **FRE("")**, автоматически управляется интерпретатором MSX Бейсика. Когда программа слишком длинна или использует слишком много переменных, следует использовать внешнее устройство для временного хранения данных(см. Главу 5). Кроме того, компьютеры MSX-2 имеют большой общий объем ОЗУ - обычно 128, 256 или 512 Кбайт. Эти дополнительные области не управляются автоматически - т.е. если программа, например, использует слишком много переменных, это не значит, что эти переменные будут автоматически перенесены в дополнительные области ОЗУ. Доступ к этим областям такой же, как и к дисководам.

Примечание:

Имеется две версии MSX2 БЕЙСИКа: MSX версия 2.0 и MSX версия 2.1. Прежде чем продолжить ознакомление с этим разделом, проверьте сообщения, появляющиеся при включении Вашей машины.

Системы расширения, используемые для доступа к дополнительным областям ОЗУ, будут упоминаться как Диск ОЗУ 1 и Диск ОЗУ 2 для MSX 2.0 и MSX 2.1 соответственно. Имеются некоторые различия между Диском ОЗУ 1 и Диском ОЗУ 2:

| | Диск ОЗУ 1 | Диск ОЗУ 2 |
|---|---|---|
| Объем памяти | 32 Кбайта | Общий объем ОЗУ 32 Кбайта |
| Максимальное число файлов | 32 | 64 |
| Скорость | | Быстрее |

### К-1. Оператор CALL MEMINI(размер)

Оператор **CALL MEMINI** инициализирует Диск ОЗУ и сообщает его имеющийся текущий объем. Этот оператор должен быть выполнен до попыток обращения к Диску ОЗУ. Кроме того, помните, что обращение это уничтожает все файлы, записанные до этого на Диске ОЗУ.

Оператор **CALL MEMINI** имеет аргумент, указывающий размер минус 1 (в байтах), который Вы хотите зарезервировать для временного хранения данных. "Размер" лежит в пределах от 1023 до 32767 у Диска ОЗУ 1, и от 6654 до максимального объема ОЗУ минус 32769 у Диска ОЗУ 2. Если "размер" меньше вышеуказанного минимального значения, Диск ОЗУ отключается; если "размер" больше вышеуказанного максимального значения, появляется сообщение неправильного вызова функции (Illegal function call). Когда "размер" пропущен, он по умолчанию устанавливается равным максимальному.

Примечание:

> Аргумент "размер" необходим только если Вы хотите использовать дополнительные области ОЗУ как для временного хранения файлов, так и для других целей. Если же это не так, то можно всегда использовать **CALL MEMINI** без аргумента "размер".

### К-2. Спецификации файлов

Спецификации файлов включают имя устройства "MEM:", имя файла, состоящее не более чем из 8 символов, и расширение имени файла, состоящее из точки и не более чем 3 символов.

Пример:

**"MEM: PROGRAM I.BAS"**





Расширение имени файла необязательно; если оно пропущено, точка также может быть опущена. Имя устройства не требуется с операторами расширения, перечисленными в разделе 13-8-3. Символы маскирования не могут использоваться в имени файла или в расширении.

### К-3. Операторы расширения, используемые с Диском ОЗУ

Операторы расширения для Диска ОЗУ очень похожи на операторы **FILES, KILL** и **NAME** MSX DISK Бейсика.

**(1) CALL MFILES**

Выводит на экран список файлов, записанных на Диске ОЗУ, и оставшееся свободное пространство.Этот оператор в отличие от **FILES** не принимает аргумента.

**(2) CALL MKILL**(спецификация файла)

Стирает указанный файл из Диска ОЗУ. Обратите внимание на использование круглых скобок.

**(3) CALL MNAME**(спецификация старого файла **AS** спецификация нового файла)

Изменяет имя файла. "спецификация старого файла" должна существовать; "спецификация нового файла" может не существовать.

### К-4. Другие операторы и функции

Следующие операторы и функции можно использовать, как в MSX DISK Бейсике. Помните, однако, что имя устройства MEM: обязательно всякий раз, когда требуются спецификации файлов.

| | | | |
|---|---|---|---|
| SAVE | LOAD | MERGE | RUN |
| OPEN | CLOSE | | |
| PRINT# | PRINT#USING | | |
| INPUT# | INPUT$ | LINE INPUT# | |
| EOF | LOC | LOF | FPOS |

- **SAVE** всегда записывает программу в формате ASCII, поэтому она всегда готова для операторов **MERGE** и **RUN** (слияние и выполнение).

- **LOC**(номер файла) возвращает позицию в файле, начиная с начала файла. Например, **LOC** возвращает 0 сразу после того, как файл открыт для ввода или вывода (**OPEN FOR INPUT/OUTPUT**) и возвращает длину файла сразу после того, как файл открыт для добавления (**OPEN FOR APPEND**).

- **LOF**(номер файла) возвращает длину файла

- **FPOS**(номер файла) возвращает оставшийся объем Диска ОЗУ.
  Указанный файл не должен быть открытым.

Примечание:

С Диском ОЗУ 2 значения, возвращенное функциями **LOC**, **LOF** и **FPOS** могут быть больше 32767, тем самым вызывая переполнение, если они присвоены целочисленным переменным.



# ПРИМЕЧАНИЯ ДЛЯ ПОЛЬЗОВАТЕЛЯ СХЕМЫ УПРАВЛЕНИЯ ПАМЯТЬЮ (М А П П Е Р А)



Программы, использующие маппер, должны избегать конфликта с другими программами, также использующими маппер . Это означает, что если две программы используют маппер одновременно, не зная о существовании друг друга, они легко могут вызвать сбой системы. Это очень важно, поскольку MSX DOS версия 2 (в дальнейшем упоминаемая как ДОС 2) широко использует маппер. Ее дисковые буферы и даже программы ее ядра резидентно находятся в распределенной памяти.

Во избежание такой ситуации ДОС2 предлагает подпрограммы поддержки маппера. Доступ к ним из прикладной программы может быть осуществлен через вектор перехода, размещенный в системной рабочей области ДОС2, и программы могут найти адрес этого вектора, используя "ВЫЗОВ РАСШИРЕННОЙ BIOS" ("EXTENDED BIOS CALL").

Распределенная память логически разделена на ряд сегментов ОЗУ по 16К. Используя подпрограммы поддержки маппера, программы легко могут выделять, освобождать, читать, записывать, вызывать и активизировать эти сегменты. За исключением первых четырех сегментов, составляющих основные 64К ОЗУ, сегменты должны быть явно распределены перед их использованием. .

Имеется два типа сегментов, пользовательские и системные.
Для облегчения переноса существующих прикладных программ, пользовательские сегменты распределяются снизу вверх (т.е. первым распределяется сегмент с наименьшим номером). Прикладная программа может найти сегмент с максимальным номером путем повторного вызова подпрограммы "распределения сегмента" ("allocate segment") до тех пор, пока она не возвратит ошибку, или возвращаемый номер сегмента перестанет равномерно наращиваться. В большинстве случаев,

номер распределенного сегмента будет равномерно наращиваться до тех пор, пока не закончатся все свободные сегменты. Это происходит потому, что системные сегменты распределены - сверху - вниз, и пользовательские сегменты автоматически освобождаются по окончании программы. Однако, свободные сегменты могут стать фрагментарными, если пользовательские и системные сегменты часто распределяются и освобождаются.

Прежде чем программа приступит к использованию подпрограмм маппера, она должна убедиться, что работает на машине с подпрограммами поддержки маппера. Это также может быть выяснено с помощью "ВЫЗОВ РАСШИРЕННОЙ BIOS". Если подпрограммы маппера не существуют, программа должна управлять маппером самостоятельно, хотя вся распределяемая память свободна для использования.

Программа, использующая маппер без поддержки ДОС2 и взаимодействующая с другими программами, должна обеспечить подпрограммы поддержки маппера так же, как и ДОС2.

### Л-1. Описание подпрограмм поддержки маппера

#### Л-1-1. Вызовы расширенной BIOS

Вектор перехода подпрограмм поддержки маппера может быть получен вызовом ловушки Расширенной BIOS с номером устройства 4. Это может быть проделано сначала проверкой байта флага по адресу FB20H, который показывает, действительна ли ловушка вызова расширенного BIOS или нет. Если наименьший значащий бит не установлен, тогда ловушка недействительна, и, следовательно, поддержки маппера не существует. В противном случае, может и должен быть вызван адрес FFCAH, который является ловушкой всех Расширенных BIOS. Если ответа на этот вызов нет, то нет и поддержки маппера.





Должны быть приняты меры предосторожности, поскольку этот вызов проходит по всем Расширенным BIOS. Например, указатель стека должен находиться в PAGE-3 (адрес выше, чем C000h) для вызова этой ловушки.

Построить таблицу имен устройств (Функция передачи #0)
Входы:

[B] = слотовый адрес таблицы
[HL] = указатель на элемент таблицы
[D] = 0 (передача)
[E] = 0 (номер функции)

Выходы:

[B:HL]
+ 0 = номер устройства (4 для поддержки маппера)
+ 1 = зарезервирован

[HL] = передвинут вперед на 2
AF, BC, DE сохранены

Вызывающий должен зарезервировать 64 байта области ОЗУ (2 байта помноженные на 32 элемента) и передать ее слотовый и начальный адрес. По возвращении, эта область заполнена номерами устройств всех существующих в системе Расширенных BIOS, и HL передвигается вперед на 2, помноженное на число возвращенных элементов.

Построить таблицу входа в вектор перехода (Функция маппера #0)
Входы:

[B] = слотовый адрес таблицы
[HL] = указатель на вход таблицы
[D] = 4 (число устройств для поддержки маппера)
[E] = 0 (номер функции)

Выходы:

[B:HL]
+ 0 = слотовый адрес маппера ОЗУ
+ 1 = нижний адрес вектора перехода
+ 2 = верхний адрес вектора перехода
+ 3 = число свободных сегментов по 16К
+ 4 = общее число сегментов по 16К
от + 5 до + 7
= зарезервированы, заполнены 0.
[HL] = переводится вперед на 8
AF, BC, DE сохранены

Вызывающий должен зарезервировать 8 байтов области ОЗУ (один вход, поскольку две подпрограммы поддержки маппера не могут сосуществовать) и передать ее слотовый и начальный адрес. По возвращении, эта область заполнена описанной выше информацией. Если поддержки маппера нет, HL будет сохранен. Эта функция имеет интерфейс, похожий на интерфейсы других устройств Расширенной BIOS.

Вернуть вектор перехода (Функция маппера #2)

Входы:

[A] = 0
[D] = 4 (номер устройства поддержки маппера)
[E] = 2 (номер функции)

Выходы:

[A] = некоторое ненулевое значение, детали еще не определены
[B] = слотовый адрес распределенного ОЗУ
[C] = число свободных сегментов по 16К
[HL] = адрес вектора перехода
DE сохранен

Если нет поддержки маппера, AF, BC, DE и HL сохраняются. Следовательно, регистр A должен быть инициализирован перед вызовом.





### Л-1-2. Вектор перехода

Программа может использовать маппер , вызывая различные подпрограммы через вектор перехода, возвращаемый из вызова Расширенной BIOS. Вектор перехода расположен в странице-3 ОЗУ, поскольку страница-3 может быть вызвана непосредственно (не через межслотовые вызовы) для повышения эффективности работы маппера.

Входы этого вектора следующие:

| | | |
|---|---|---|
| + 0 | ALL SEG | Распределяет сегмент 16К |
| + 3 | FRE SEG | Освобождает сегмент 16К |
| + 6 | RD SEG | Межслотовый вызов. Адрес в IYh:IX |
| + 9 | WR SEG | Записывает байт из E в адрес A:HL. |
| + 12 | CAL SEG | Межсегментный вызов. Адрес в IYh:IX. |
| + 15 | CALLS | Межсегментный вызов. Адрес в строкепосле оператора вызова. |
| + 18 | PUT PH | Помещает сегмент в страницу (HL). |
| + 21 | GET PH | Извлекает текущий сегмент для страницы (HL) |
| + 24 | PUT P0 | Помещает сегмент в страницу 0 |
| + 27 | GET P0 | Извлекает текущий сегмент для страницы 0. |
| + 30 | PUT P1 | Помещает сегмент в страницу 1. |
| + 33 | GET P1 | Извлекает текущий сегмент для страницы 1. |
| + 36 | PUT P2 | Помещает сегмент в страницу 2. |
| + 39 | GET P2 | Извлекает текущий сегмент для страницы 2. |

Ошибка при распределении сегмента обычно указывает, что уже нет свободных сегментов, хотя она также может означать, что в регистр A передан недействительный параметр. Ошибка при освобождении сегмента указывает, что сегмент с указанным номером не существует или уже свободен.

**ALL SEG:** Входы: [A] = 0 распределить пользовательский сегмент
[A] = 1 распределить системный сегмент
Выходы: Установка переноса → Ошибка
Сброс переноса → Распределен, номер сегмента в [A]

**FRE SEG:** Входы: [A] = номер освобождаемого сегмента
Выходы: Установка переноса → Ошибка
Сброс переноса → Сегмент освобожден

### Л-2-2. Межсегментные чтение и запись

Следующие две подпрограммы могут быть вызваны для чтения или записи одного байта из любого распределенного сегмента. Вызывающая последовательность очень похожа на подпрограммы межслотового чтения и записи. Все регистры за исключением AF сохраняются, и проверка действительности номера сегмента не производится. Два старших бита адреса игнорируются, поскольку номер сегмента определяет сегмент 16К, который может оказаться в любой из четырех страниц. Чтение или запись данных будут произведены из правильного сегмента независимо от текущего выбора слотов или деления на страницы, которые не будут затронуты.





**RD SEG:** Входы: [A] = номер сегмента для чтения
[HL] = адрес внутри этого сегмента
Выходы: [A] = значение байта по этому адресу Все остальные регистры сохраняются

**WR SEG:** Входы: [A] = номер сегмента для записи
[HL] = адрес внутри этого сегмента
[E] = значение для записи
Выходы: [A] = испорчен
Все остальные регистры сохраняются

### Л-2-3. Межсегментные вызовы

Для межсегментных вызовов имеется две подпрограммы. Они очень точно моделируют подпрограммы межслотовых вызовов, и специфика их использования очень близка.

Проверка действительного существования вызываемого сегмента не производится, поэтому ответственность за это ложится на пользователя. Вызываемый сегмент будет помещен в правильную страницу микросхемы маппера, однако обеспечение активизации слота маппера на этой странице также ложится на пользователя, поскольку ни одна из этих программ не изменит выбора слотов.

Подпрограмма не может использоваться для осуществления межсегментного вызова в страницу-3. Если такая попытка будет сделана, тогда указанный адрес в странице-3 будет просто вызван безо всякого распределения страниц, поскольку страница-3 ни при каких условиях не должна изменяться. Вызов в страницу-0 должен осуществляться с некоторой осторожностью из-за прерывания и другой входной точки.

Эти подпрограммы всегда отменяют прерывания перед передачей управления вызываемой подпрограмме, но никогда не разрешают их при выходе.

Следовательно, ответственность за повторное разрешение прерываний, как можно скорее, ложится на вызывающую подпрограмму. Кроме того, мы советуем помещать команду "EI" после межсегментного вызова на тот случай, если вызывающая подпрограмма не разрешила прерывания.

Параметры не могут быть переданы в регистры IX, IY, AF, BC, DE или HL, поскольку они используются внутри подпрограммы. Эти регистры будут испорчены межсегментным вызовом, а также могут быть испорчены вызывающей подпрограммой. Следует позаботиться о том, чтобы стек не был выведен за пределы страниц межсегментным вызовом.

**CAL SEG:** Входы: [IYh] = номер вызываемого сегмента
[IX] = вызываемый адрес
Выходы: AF, BC, DE и HL, возвращенные из вызванной подпрограммы

**CALLS:** Входы: Никаких
Вызывающая последовательность:

```
CALL    CALLS
DB      SEGMENT
DW      ADVRESS
```

Выходы: AF, BC, DE и HL, возвращенные из вызванной подпрограммы

### Л-2-4. Подпрограммы прямого распределения страниц

Следующие подпрограммы предназначены для того, чтобы программы могли непосредственно манипулировать текущим состоянием распределения страниц, не прибегая к доступу к аппаратуре.





Использование этих подпрограмм обеспечивает совместимость с любыми изменениями в деталях аппаратного обеспечения. Подпрограммы очень быстрые, поэтому их использование не помешает программам.Имеются подпрограммы для прямого чтения или записи в любой регистр четырех страниц. Эти подпрограммы эквивалентны простым командам "IN" и "OUT" Z80 для регистров страниц. Проверка номера сегмента не производится, поэтому ответственность за него принадлежит пользователю.

Хотя подпрограмма "PUT P3" имеется, она фактически является пустой подпрограммой и никогда не изменяет регистр страницы-3. Это имеет место потому, что содержание страницы-3 не может изменяться. Подпрограмма "GET P3" не ведет себя ожидаемым образом: дать пользователю возможность определить, какой сегмент находится в странице-3.

Другая пара подпрограмм ("GET PH" и "PUT PH") идентична по функции, за исключением того, что страница определяется двумя старшими битами регистра H. Это удобно, поскольку регистр HL содержит адрес, и эти подпрограммы не портят регистр HL. "PUT PH" никогда не изменяет регистра страницы-3.

**PUT Pn** Входы: n = 0, 1, 2 или 3 для выбора страницы
[A] = номер сегмента
Выходы: Никаких
Все регистры сохраняются

**GET Pn** Входы: n = 0, 1, 2 или 3 для выбора страницы
Выходы: [A] = номер сегмента
Все регистры сохраняются

| | | |
|---|---|---|
| **PUT PH** | Входы: | [H] = старший байт адреса |
| | | [A] = номер сегмента |
| | Выходы: | Никаких |
| | | Все регистры сохраняются |
| **GET PH** | Входы: | [H] = старший байт адреса |
| | Выходы: | [A] = номер сегмента |
| | | Все регистры сохраняются |

Перед использованием этих подпрограмм прямого распределения страниц для изменения его состояния, программа должна сначала использовать подпрограммы "GET Pn" для определения начальных четырех сегментов для момента, когда понадобится восстановить их. Программа не должна предполагать фиксированные значения для этих начальных сегментов, поскольку они вполне могут измениться в будущих версиях системы.

### Л-3. Примеры использования маппера

Следующая иллюстративная программа распределяет непрерывный блок сегментов, пронумерованных начиная от 0 до числа, на 1 меньшего размера блока. Она также возвращает размер блока в регистр B. Обратите внимание, что свободные сегменты, несмежные с первым блоком, не могут использоваться.

Когда известно, что в системе нет программ поддержки маппера, следует управлять аппаратурой маппера самим. Для поддержки этих обоих случаев, программа вводит локальный вектор перехода, который первоначально направлен на наши собственные подпрограммы управления маппером. Должны поддерживаться только используемые входы. Остальные входы могут быть пустыми. В любом случае следует знать, сколько физической памяти имеется в машине, что является самой первой и самой трудной задачей, когда нет поддержки маппера.





```
HOKVLD  EQU   0FB20H    ;флаг действия ловушки
EXTBIO  EQU   0FFCAH    ;вход вызова EXTBIO
;
START:
        LD    A,(HOKVLD) ;имеются Расширенные
                         BIOS?
        RRA              ;(см.наим.значащий бит)
        JR    NC,MYOWN   ;нет - ничего,использую
                         свою собствен
                         ;н о б с т в е н н у ю
                         подпрограмму
        XOR   A
        LD    D8,0402H   ;установка устройства 4,
                         функция 2
        CALL  EXTBIO     ;
        OR    A          ;есть подпрограммы
                         маппера?
        JR    Z,MYOWN    ;нет, использую свою
                         собственную;подпрограмм
                         у
        LD    DE,MAPVEC  ;переписать свой вектор
                         перехода
        LD    BC,VECSIZ  ;
        LDIR
        LD    B, 4       ;ожидается номер
                         первого сегмента
ALLOCATE MORE:
        XOR   A
        CALL  ALL SEG    ;запрос пользовательского
                         сегмента
        RE    C          ;распределение не
                         удалось,больше
                         ;не будет
```

```
                CP      B               ;что я ожидаю?
                JR      NZ, FRE SEG     ;несмежный, освободить последний
                                        ;сегмент и вернуться
                INC     B               ;еще одна попытка распределения
                JR      ALLOCATE MORE
MAPVEC
;
;       Мои собственные векторы перехода, некоторые могут иметь
;       пустые входы
ALL SEG: JP     MY ALL SEG      ;распределить сегмент
FRE SEG: JP     MY FRE SEG      ;освободить сегмент
RD SEG: JP      MY RD  SEG      ;межсегментное чтение
WR SEG: JP      MY WR  SEG      ;межсегментная запись
CAL SEG: JP     MY CAL SEG      ;межсегментный вызов
CALLS: JP       MY CALLS        ;межсегментный вызов
PUT PH: JP      MY PUT PH       ;вставить сегмент страницы (HL)
GET PH: JP      MY GET PH       ;взять сегмент страницы (HL)
PUT P0: JP      MY PUT P0       ;вставить сегмент страницы 0
GET P0: JP      MY GET P0       ;взять сегмент страницы 0
PUT P1: JP      MY PUT P1       ;вставить сегмент страницы 1
GET P1: JP      MY GET P1       ;взять сегментстраницы 1
PUT P2: JP      MY PUT P2       ;вставить сегмент
                                 страницы 2
GET P2: JP      MY GET P2       ;взять сегмент страницы 2
PUT P3: JP      MY PUT P3       ;вставить сегмент страницы 3
```





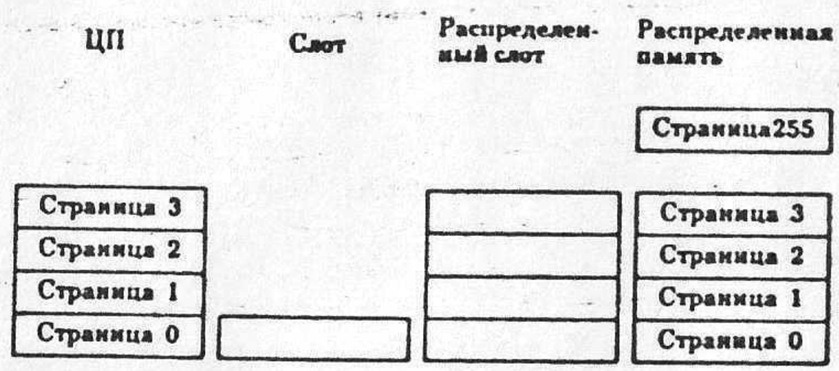
```
GET P3: JP      MY GET P3       ;взять сегмент страницы 3
VECSIZ EQU      $-MAPVEC
MYOWN:
                LD B,...        ;проверить размер памяти
                                ;и
                RET             ;возвратить его в регистр B
```

### Л-4. Спецификация маппера MSX-2

Маппер представляет собой устройство, помещаемое в слот для замены логических адресов памяти центрального процессора (ЦП) физическими адресами и расширения памяти до 4 Мбайт. Можно распределить максимум 256 страниц (16 Кбайт). Распределение логических страниц на физические страницы определяется содержанием регистра распределения. Регистры распределения находятся по адресам ввода-вывода от FCH до FFH. Эти регистры - как для чтения, так и для записи. Маппер в MSX2 имеет минимальную распределяемую память 64 К.

| Адрес ввода-вывода | Страница |
|---|---|
| FCH | 0 |
| FDH | 1 |
| FEH | 2 |
| FFH | 3 |

**Логическая структура маппера**




**Физическая структура маппера**


**Системное программное обеспечение маппера**

Система MSX-2 инициализирует регистр распределения следующим образом:АЕДЕЙЯ

| Адрес ввода-вывода | Начальные данные | Логическая страница | Физическая страница |
|---|---|---|---|
| FCH | 03H | 0 | 3 |
| FDH | 02H | 1 | 2 |
| FEH | 01H | 2 | 1 |
| FFH | 00H | 3 | 0 |





Инициализация маппера записывает вышеуказанные значения в порты ввода-вывода, поэтому она не затрагивает аппаратное обеспечение, не имеющее маппера. Инициализация не выполняет задачу определения наличия маппера. После инициализации маппера системное программное обеспечение выполняет базовые поиски ПЗУ и ОЗУ. Находится ли ОЗУ ЦП в Распределенном слоте или нет, полностью зависит от аппаратного обеспечения слота.

Слот и количество распределенной памяти также зависят от аппаратного обеспечения, поэтому прикладные программы должны выяснить это самостоятельно.

Примечания:

Прикладные программы, использующие маппер, должны инициализировать адрес распределения, когда они возвращают управление Бейсику или MSX DOS.

# СРЕДСТВА КОММУНИКАЦИИ

ПРИЛОЖЕНИЕ М

Этот последний раздел относится только к пользователям компьютеров MSX, оснащенных интерфейсом RS-232C. Разъем RS-232C может использоваться для передачи данных через модем другим компьютерам или принтерам.

## М-1. Введение в методы передачи данных

### М-1-1. Параллельная и последовательная передача

Данные могут передаваться по кабелям, если они преобразованы в электрические сигналы. В компьютере числа представлены в двоичном виде, использующем только 0 и 1. Символ "A", код ASCII которого 65, "записывается" в памяти компьютера как 01000001. Этот код становится при передаче последовательностью электрических импульсов: "0" - положительный импульс и "1" - отрицательный. Каждый импульс называется битом. Обратите внимание, что числа от 0 до 127 могут быть представлены в двоичной записи с использованием только 7 битов: нуль представляется как 00000000 и 127 как 11111111.

Однако компьютеры хранят информацию в блоках памяти, именуемых байтами, составленных из 8 битов, допуская тем самым представление чисел от 0 до 255. Мы увидим позже, как восьмой бит используется в передаче данных.

Могут использоваться два метода передачи:

(1) Параллельная передача: биты, принадлежащие байту, передаются по отдельным проводам.

0
1
0
0
"A" = 65 → 0
0
0
1

Этот тип передачи используется для передачи данных принтеру MSX (т. наз. разъем "Центроникс").

(2) Последовательная передача: отдельные биты байта передаются один за другим по одному проводу.

"A" = 65 → 01000001

Этим способом данные передаются через разъемы RS-232C - это единственный способ, который может использоваться для передачи данных через однопроводную телефонную линию.

### М-1-2. Четность и специальные символы

(1) Четность

Восьмой бит байта, называемый битом четности, позволяет принимающей системе проверять содержание остальных 7 битов. Установка четности относится к способу осуществления такой проверки. Имеется три возможности:

| |
|---|
| Проверка на четность |
| Проверка на нечетность |
| Отсутствие проверки |





Установка четности передающей и принимающей систем должно быть равным. Предположим, что обе системы установлены на Четность.

Это означает, что каждый передаваемый байт должен иметь четное количество единиц. 65 содержит две единицы, поэтому будет передано без изменений. 67 ("C"), однако, содержит три единицы (01000011). Поэтому восьмой бит замещается единицей, так что общее число единиц становится четным.

Принимающая система, также установленная на Четность, суммирует общее число единиц в принимаемом байте. Иногда из-за шумов влинии передачи бит передается неправильно. Когда это происходит, существует большая вероятность того, что общее число единиц станет нечетным. В зависимости от программного обеспечения коммуникации, принимающий компьютер или проигнорирует такой байт, или запросит повторение передачи.

Проверка четности работает таким же образом и с установкой на Нечетность.

(2) Специальные символы

В Компьютере MSX кодируется до 256 символов.

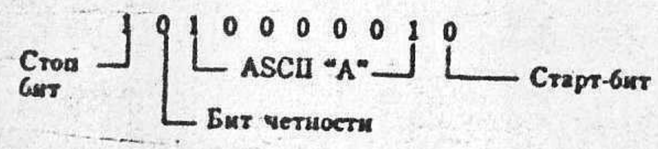
| Код | Символ/Управление |
|---|---|
| 0 - 31 | Управляющие |
| 32 - 127 | Стандартный английский алфавит и другие символы, используемые в MSX Бейсике |
| 128 - 255 | Кириллица и Специальные символы |

Когда четность установлена на Отсутствие четности, для передачи данных могут использоваться восемь битов байта, допуская передачу кодов от 0 до 255.

Количество битов, используемых для передачи символа, называется словом: 7-битное слово допускает передачу 128 различных символов с проверкой четности; 8-битное слово допускает передачу 256 различных символов без проверки четности. Число битов в слове называется длиной слова.

### М-1-3. Скорость передачи

Скорость передачи выражается в битах в секунду (bps). Следовательно, можно сказать, что скорость передачи 1200 б/с соответствовать 1200/8 = 150 символов в секунду. Однако, в действительности это не так. На самом деле, принимающая система должна быть способна определять начало каждого байта в последовательности электрических импульсов. Для достижения этого, к началу каждого байта добавляется нулевой бит (старт-бит) - и бит 1 (стоп-бит) к его концу. Один символ, таким образом, использует 10 битов:




Скорость передачи 1200 б/с, следовательно, соответствует 120 символам в секунду.

Примечание:

Скорость передачи часто измеряется в бодах (Baud rate). Хотя боды и биты в секунду представляют различные понятия, мы будем считать, что они эквивалентны.





### М-1-4. Модем

Телефонные линии разработаны для оптимальной передачи электрических сигналов в полосе акустических частот - и для этой цели используется единственный провод. Модем (МОдулятор-ДЕМодулятор) предполагает две интерфейсные функции:

(1) Модуляция: электрические сигналы из компьютера обрабатываются для передачи по обычным телефонным линиям.

(2) Демодуляция: электрические сигналы из телефонной линии обрабатываются, чтобы компьютер мог их прочитать.

### М-1-5. Режимы передачи

В зависимости от конструкции модема, информация может обрабатываться в трех различных режимах:

(1) Режим симплекс: информация может идти только от передатчика к приемнику (как в радиовещании)

(2) Режим полу-дуплекс: информация может идти в обе стороны, но не одновременно (как в УКВ-приемопередатчиках)

(3) Режим полного дуплекса: информация может идти одновременно в обе стороны, как при телефонном разговоре.

### М-1-6. Метод синхронизации

При приеме данных нужно распознавать начало каждого байта в последовательности битов. Применяются два метода решения этой задачи:

(1) Синхронная коммуникация: в начале передачи посылается специальный управляющий код. Начиная с этой точки данные посылают ся в виде непрерывной последовательности битов.
Принимающая система выделяет каждый байт подсчетом времени.

(2) Асинхронная коммуникация: каждому байту предшествует старт-бит; байт заканчивается стоп-битом. Это обеспечивает передачу данных с произвольным распределением во времени.

## М-2. Интерфейс RS-232C

Интерфейс RS-232C разработан для передачи данных по последовательному каналу.

### М-2-1. Спецификация и варианты интерфейса

| Метод | Старт-стоп бит (асинхронный) |
|---|---|
| Скорость синхронизации | 50, 75, 110, 300, 600, 1200, 2000, 2400, 3600, 4800, 7200, 9600, 19200 bps |
| Старт-бит | Один бит |
| Длина слова | 5, 6, 7, 8 битов |
| Четность | Четная, нечетная, отсутствие четности |
| Стоп-бит | 1, 1.5, 2 бита |
| Уровень сигнала | ВКЛ.: + 5 - + 12В; ВЫКЛ.: - 5 - - 12В |
| Режим передачи | Полный дуплекс |





Коммуникация с другим устройством возможна только когда совпадают вышеуказанные спецификации и варианты. Например, если данный интерфейс работает в режиме полного дуплекса, другое устройство также должно быть установлено на полный дуплекс.

### М-2-2. Спецификации разъема

Следующая таблица описывает функции каждого контакта разъема RS-232C.

| Контакт | Имя сигнала | Функция | Направление |
|---|---|---|---|
| 1 | FG | Заземление корпуса | — |
| 2 | SD (TXD) | Линия передачи | Выход |
| 3 | RD (RXD) | Линия приема | Вход |
| 4 | RS (RST) | Сообщает соответствующему устройству, что компьютер готов переда | Выход |
| 5 | CS (CTS) | Сообщает компьютеру, что устройство готово принимать данные | Вход |
| 6 | DR (DSR) | Сообщает компьютеру, что устройство способно передавать/принимать данные или контрольные сигналы | Вход |
| 7 | SG | Заземление сигнала | — |
| 8 | CD | Сообщает компьютеру, что устройство принимает несущую частоту | Вход |
| 20 | ER (DTR) | Сообщает устройству, что компьютер способен передавать/принимать управляющие сигналы | Выход |

### М-2-3. Режимы коммуникации

Интерфейс RS-232C допускает три режима коммуникации:

(1) Терминальный режим: используется для управления отображения данных, посылаемых другим компьютером (главным компьютером) и для передачи главному компьютеру данных, введенных с клавиатуры.

(2) Режим передачи программ: используется для обмена программами между двумя компьютерами.

(3) Режим передачи данных: используется для передачи данных с помощью команд MSX Бейсика.

### М-2-4. Инициализация интерфейса RS-232C.

Прежде чем начинать какие-либо операции с интерфейсом RS-232C, следует определить несколько параметров. Эти параметры соответствуют вариантам, перечисленным в вышеприведенной таблице. Инициализация осуществляется оператором CALL COMINI:

CALL COMINI("Номер устройства: Длина слова Четность Стоп-бит Xon/off Управление CS Автоперевод строки на передаче Автоперевод строки на приеме SI/SO", Скорость приема, Скорость передачи, Время ожидания)

| | |
|---|---|
| Номер устройства | Всегда 0 с этим интерфейсом |
| Длина слова | Целое число от 5 до 8:<br>5 → 5 битов<br>6 → 6 битов<br>7 → 7 битов<br>8 → 8 битов |





| | |
|---|---|
| Четность | Один символ (E, O, I или N)<br>E → Четная<br>O → Нечетная<br>I → Четность игнорируется при приеме, но проверяется на передаче<br>N → Проверки четности нет<br>• E, O и I не могут использоваться, если "Длина слова" установлена на 8. |
| Стоп-бит | Целое число от 1 до 3; определяет длину стоп-бита:<br>1 → 1 бит<br>2 → 1,5 бита<br>3 → 2 бита<br>• Длина стоп-бита для приема должна быть больше (или равна),чем длина стоп-бита для передачи |
| Xon/off | Один символ (X или N); определяет, действуют или нет управляющие коды Xon/off для предотвращения переполнения буфера:<br>X → действуют<br>N → отменены<br>• Когда код Xoff (&H13) получен во время передачи, передача остановится и возобно ится при получении кода Xon (&H11) Когда буфер приема (общая емкость - 127 байт) заполнен до 113 байтов во время приема, посылается код Xoff (&H13), чтобы сообщить передающему устройству о прекращении передачи; код Xon (&H11) посылается, когда в приемном буфере остается только два байта данных. |

При приеме кода CR (&H0D) во время выполнения оператора LOAD, будет послан код Xoff; код Xon будет послан для возобновления загрузки, когда буфер освободится до 2 байтов.

**Управление CS** — Один символ (H или N); определяет, действуют или нет сигналы на ножке разъема CS:

H → действуют (при передаче данных с использованием PRINT# или SAVE или ввода с клавиатуры в Терминальном режиме - передача прерывается, пока не появится сигнал на контакте CS)

N → сигналы CS не проверяются

* Это может использоваться, если устройство выполняет программу, способную устанавливать сигнал ВКЛ. на CS во время приема данных.

**Автоперевод строки на приеме** — Один символ (A или N); определяет, должен ли добавляться код перевода строки LF к коду возврата каретки CR во время приема:

A → код LF добавляется (CR → CR + LF)

N → код LF не добавляется (CR → CR)

* В зависимости от передающего компьютера, код разделителя данных (включая разделители строк в программе на БЕЙСИКе) состоит только из кода CR или из пары CR + LF. В MSX БЕЙСИКе, разделитель является парой (LF + CR), поэтому должен быть установлен A для исправления данных, получаемых от компьютера, использующего в качестве разделителя явный CR.





**Автоперевод строки на передаче** — Один символ (A или N); определяет, должен ли код LF удаляться из пары CR + LF во время передачи:

A → код LF удаляется (CR + LF → CR)

N → код LF не удаляется (CR + LF → CR + LF)

* Этот параметр следует установить в A при передаче устройству, использующему явный код CR в качестве разделителя данных

**Управление SI/SO** — Один символ (S или N)
Относится только к передаче японских символов. Установите на N.

**Скорость приема/передачи** — Одно из нижеперечисленных чисел; устанавливает скорость передачи отдельно для приема и передачи:

| | | | | |
|---|---|---|---|---|
| 50 | 75 | 110 | 300 | 600 |
| 1200 | 1800 | 2000 | 2400 | 3600 |
| 4800 | 7200 | 9600 | 19200 | |

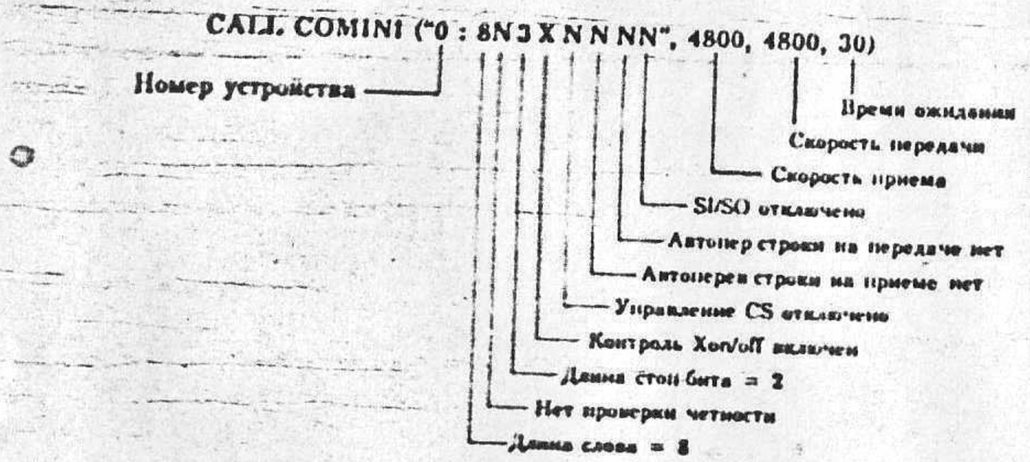
* Скорость передачи должна совпадать со скоростью приема устройства, подключенного к компьютеру, и наоборот. Общепринятой скоростью для модемов является 300 bps; можно выбрать более высокую скорость коммуникации данных с другим компьютером. Обратите внимание, что компьютер MSX не может обрабатывать данные со скоростью, равной 19200 bps.

**Время ожидания** — Число от 0 до 255

* Этот параметр принимается во внимание только в том случае, если управление CS включено. Если "время ожидания" выйдет до того, как включится CS, то произойдет ошибка.

**При этом, если Время ожидания установлено на 0, компьютер будет ожидать включения CS. Установка по умолчанию 0, и в обычных условиях рекомендуется 4-6.**

**Пример:**




## М-3. Терминальный режим

В терминальном режиме Вы не можете управлять главным компьютером (х о с т о м) через клавиатуру своего компьютера: программа главного компьютера работает, в то время как Ваша отменена; дисплей Вашего компьютера будет отображать принятые данные и передавать введенные через Вашу клавиатуру.

Главный компьютер может быть подключен непосредственно через кабель RS-232C - или через устройство-посредник (модем, акустическое устройство сопряжения). Следует внимательно изучить установки и спецификации как главного компьютера, так и устройства-посредника, если таковое имеется, для правильной инициализации интерфейса RS-232C (CALL COMINI).





### М-3-1. Переход в терминальный режим

Переход в терминальный режим осуществляется оператором расширения CALL COMTERM:

CALL COMTERM("Номер устройства")

Номер устройства всегда 0. Прежде чем ввести эту команду, убедитесь, что система подключена необходимым образом.

Примечание:

Сигнал ER включается при включении компьютера. Сигнал ES включен, когдасистема в терминальном режиме. Сигнал CS начинает передачу после подтверждения включения сигнала CS оператором CALL COMINI.

Ошибка в процессе инициализации или неправильная операция управления вызовут неправильную работу дисплея или полное отсутствие ответа.

Если это произойдет, нажмите CTRL + STOP для повторной инициализации. Некоторые компьютеры, однако, могут быть инициализированы заново посредством включени/выключения без какого-либо ввода с клавиатуры, в то время как порт RS-232C ожидает ввода.

Для выхода из терминального режима нажмите CTRL + STOP.

### М-3-2. Функции терминального режима.

(1) SHIFT + F1 включает и выключает литерный режим. Когда ли терный режим включен, управляющие символы (ниже &H20) будут отображаться в виде двух символов: экспоненциального символа (^) и символа, код которого равен управляющему коду + &H40.
Однако, коды Xon/off не будут отображаться.

(2) [SHIFT] + [F2] включает и выключает эхо-сигнал. Когда эхо-сигнал включен, введенные с клавиатуры данные передаются, а затем отображаются на экране.

(3) [SHIFT] + [F3] то же самое, что и (2), но данные отображаются на принтер.

(4) [STOP] посылает сигнал перерыва: линия данных (SD) отключена, пока нажата [STOP].

Примечание:

Некоторые компьютеры в начале строки посылают код DEL (&H7E). Компьютер MSX распознает код DEL как BS (&H08). Следовательно, главный компьютер должен быть установлен на подавление кодов DEL. Если это невозможно, избегайте использования терминального режима. Вместо этого используйте программу коммуникации, способную подавлять коды DEL.

## М-4. Заключение

Приведенные сведения позволяют составить представление о возможностях интерфейса последовательного канала, применяемого в компьютерах MSX-2. В КУВТ "Ямаха MSX-2" применяется специальная модификация интерфейса последовательного канала, что позволяет в совокупности с использованием программных средств организовать локальную сеть класса. Сведения об этих средствах содержатся в руководствах для пользователей локальной сети КУВТ "Ямаха MSX-2".





### М-4-1. Передача программ

Использовать инструкцию SAVE, как показано ниже.

```
SAVE "COM0:"
```

Этим ваш компьютер устанавливается в режим передачи программ, храненых в своей памяти, в память главного компьютера (главный компьютер должен быть MSX-компьютером).


Примечание:

Когда компьютер включается, сигнал ER включен. До передачи программ сигнал ES включен, а после передачи выключен. сигнал CS начинает передачу, если управление CS сделано возможным посредством CALL COMIN.

Если ошибаются в инициализации или управлении, то дисплей будет работать неисправно или совсем не дать индикации. В таком случае нажать на клавиши [CTRL] + [STOP] и повторно произвести инициализацию. Некоторые компьютеры инициализуются путем выключения питания с последующим повторным включением без нажатии на клавиши, причем канал RS-232C должен находиться в состоянии ожидании ввода.

В конце программы код EOF(&H1A) передается посредством инструкции SAVE. конце программы. Этот код останавливает прием программы главным компьютером. Если главный компьютер является MSX-компьютером, то инструкция LOAD выполняется автоматически.

**М-4-2. Прием программ**

Использовать инструкцию LOAD следующим образом.

**LOAD"COM0:"**


Примечание:

Когда компьютер включается, сигнал ER включен. До передачи программ сигнал ES включен, а после передачи выключен.

Если ошибаются в инициализации или управлении, то дисплей будет работать неисправно или совсем не дать индикации. В таком случае нажать на клавиши CTRL + STOP и повторно произвести инициализацию. Некоторые компьютеры инициализуются путем выключения питания с последующим повторным включением без нажатии на клавиши, причем канал RS-232C должен находиться в состоянии ожидании ввода.

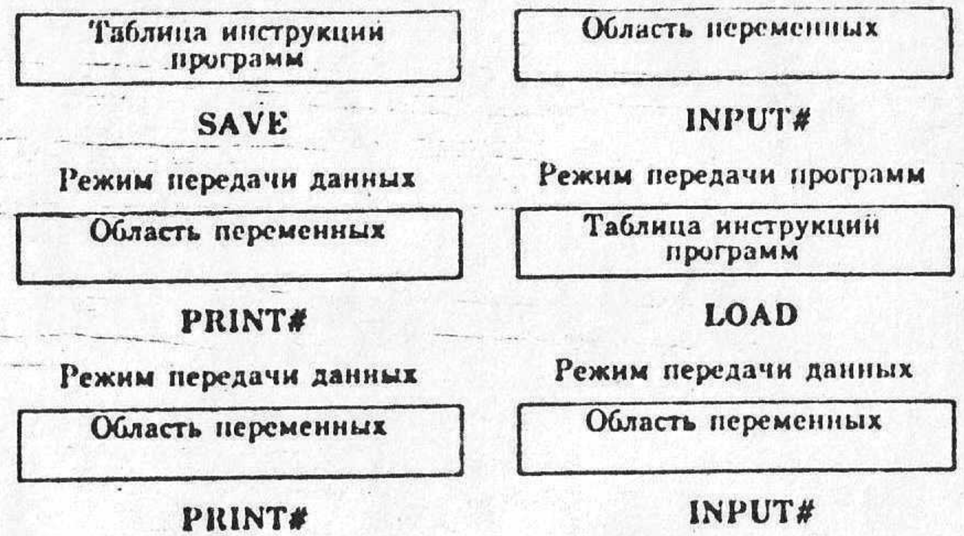
В конце программы код EOF(&H1A) передается посредством инструкции SAVE . Если главный компьютер является MSX-компьютером, то инструкция SAVE выполняется автоматически.

**М-5. Режим передачи данных**

Режим передачи данных предназначен для обмена программами или данными с главным компьютером, иным чем MSX-компьютер. При передаче программы передаваемая программа раз хранится на диске или ленте с форматом ASCII, и затем передается и хранится во внешнем ЗУ принимающего компьютера как файл данных. Вышехраненная программа поддается загрузке и запуске на вашем компьютере в том случае, если язык Бейсик главного компьютера соответствует стандарту MSX. Этот режим передачи данных может использоваться и для передачи программ с одного MSX-компьютера на другой MSX-компьютер.








Примечание:

Для того, чтобы хранить программу в области переменных, необходимо произвести следующие операции.

Раз ее хранить во внешней памяти в формате ASCII, затем инструкцией LINE INPUT# построчно загрузить этот файл ASCII. Каждый раз при вводе строки эта строка передается при помощи инструкции PRINT#.

Когда программа, построчно передаваемая вышеуказанным образом, принимается в область переменных, каждая строка этой программы хранится во внешное ЗУ принимающего компьютера при помощи инструкции PRINT#. Когда вся программа хранена во внешнем ЗУ, возможно ее загрузить посредством LOAD, RUN или MERGE.

**М-5-1. Программа на Бейсике для обмена данными**

Программа на Бейсике для обмена данными имеет следующую структуру.

| | |
|---|---|
| CALL COMINI | Инициализация функций по связи |
| OPEN | Открытие файла RS-232C и подтверждение No. файла |
| PRINT# или INPUT# | Передача (PRINT#) или прием (INPUT#) данных. LINEINPUT$ и INPUT$ также применимы. |
| CLOSE# | Окончание связи. Файл OPEN FOR OUTPUT влечет за собой передачу кода EOF. |

Примечание:

Когда компьютер включается, сигнал ER включен.
Сигнал RS включается при выдаче OPEN, а выключается при выдаче CLOSE.
Сигнал CS запускает передачу посредством PRINT#, если возможно управление CS посредством CALL COMINI.

Если ошибаются в инициализации или управлении, то дисплей будет работать неисправно или совсем не дать индикации. В таком случае нажать на клавиши [CTRL] + [STOP] и повторно произвести инициализацию. Некоторые компьютеры инициализуются путем выключения питания с последующим повторным включением без нажатии на клавиши, причем канал RS-232C должен находиться в состоянии ожидании ввода.

**М-5-2. Пример программы**

Ниже приводится пример программы "межкомпьютерного разговора". Функциональная клавиша [F1] задает, кто начинает ввод с клавиатуры: компьютер у того, кто раньше нажал на [F1],





будет передающим, а другой - принимающим. На передающем компьютере ввод осуществляется с клавиатуры при нажатии на RETURN. Когда передаваемое сообщение полностью введено, нажать на [F2] и тем самым выбрать режим передачи/ приема. Для того, чтобы закончить разговор, нажать на [F3].

```
10  CLEAR 2000:CLS:CALL COMINI(,,,4)
20  ON ERROR GOTO 410:DIM A$(20)
30  A$(0)="Yes":F=0:T=70
40  A$(1)="Hello":A$(2)="Please":A$(3)="Bye
    Bye"
50  FOR I=1 TO 3
60  KEY I,A$(I)+CHR$(13)
70  NEXT I
80  PRINT "RS232C DEMO PROGRAM"
90  OPEN "com0:"AS#1
100 CALL COM(,GOSUB340):CALL COMON
110 D$=INKEY$
120 IF F=2 THEN 260
130 IF D$=" "THEN 110 ELSE PRINT D$;
140 LINEINPUT$:E$:D$=D$+E$
150 IF D$<>A$(1) THEN 110
160 PRINT#1,D$:F=1
170 FOR J=0 TO T
180 IF F=0 THEN J=T+1
190 NEXT J
200 IF J=T+1 THEN 410
210 PRINT "+ + Transmission enabled+ +"
220 LINEINPUT B$
230 PRINT #1,B$
240 IF B$=A$(3) THEN 300
250 IF B$<>A$(2) THEN 220
260 PRINT "**Receiving now **"
270 INPUT#1,C$:PRINT C$
280 IF C$<>A$(3) AND C$<>A$(2) THEN
    270
290 IF C$=A$(2) THEN 210
300 FOR J=0 TO T:NEXT J
310 CLOSE:PRINT:PRINT:PRINT "End"
320 KEY1,"color":KEY2,"auto":KEY3,"goto"
```

```
330  ON ERROR GOTO 0:END
340  INPUT#1,C$
350  IF F=1 THEN GOTO 390
360  IF C$<>A$(1) THEN RETURN
370  PRINT#1,A$(0)
380  CALL COMOFF:F=2:RETURN
390  IF C$<>A$(0) THEN RETURN
400  CALL COMOFF:F=0:RETURN
410  PRINT"Conversation impossible"
420  GOTO 310
```

**М-6. Расширение Бейсика для связи по каналу RS-232C**

Ниже приведен алфавитный список инструкций расширения, используемых для обмена данными.

(1) CLOSE [[#]No. файла[,[#]No. файла...]]

Используется для закрытия файла(-лов). Пояснение к данной инструкции см. в главе 5. Помнить, что CLOSE без определения No. файла приведет к зарытию всех файлов, так же как END, RUN, NEW и CLEAR. Когда файл RS-232C OPEN FOR OUTPUT закрыт, код EOF (&H1A) передается с канала RS-232C.

При закрытии файла RS-232C сигнал TS выключается.

(2) – COM (["No. устройства:"],GOSUB No. строки)

Определяет строку, где начинается программа прерывания после поступления данных на канал RS-232C. Выбор No. устройства по умолчению 0. Данной инструкции должна предшествовать инструкция – COMON. Данная инструкция похожа на предложение "ON событие GOSUB". При одновременном возникновении ошибок и событий будет принят следующий порядок предпочтения.

ON ERROR GOTO
ON KEY GOSUB





ON STOP GOSUB
ON SPRITE GOSUB
ON STRIG GOSUB
ON INTERVAL GOSUB
COM GOSUB

(3) – COM BREAK [(["No. устройства"][,Целое число])]

Передает сигнал прерывания BREAK и поддержитвает строку данных SD на 0. No. устройства по умолчению 0. Целый параметр используется для определения длительности поддержания строки SD на 0 по следующей формуле.

Длительность = (1 + Длина слова + Длина кода останова)*Целое число/ Скорость в бодах

Вышеуказанное целое число варьируется в пределах 3 - 32762 и по умолчению составляет 10.

Необходимо помнить, что файл RS-232C должен быть открыт перед использованием данной инструкции.

(4) – COMDTR (["No. устройства"],Целое число)

Включает строку ER при целом числе <> 0, и выключает ее при целом числе = 0. Когда компьютер включается, строка ER включается. No. устройства по умолчению составляет 0.

Необходимо помнить, что файл RS-232C должен быть открыт перед использованием данной инструкции.

(5) – COMINI (["[No. устройства:]Длина слова Паритет Код останова Хвкл/ выкл Управление CS Автоперенос LF приема Автоперенос LF передачи SI/SO "[,Скорость приема, ]Скорость передачи] [,Время ожидания]])

Данная инструкция уже объяснена на начале этой главы.

Здесь приводим значения каждого параметра цепочки с No. устройства по SI/SO по умолчению.

| | | |
|---|---|---|
| No. устройства | → | 0 |
| Длина слова | → | 8 |
| Паритет | → | N |
| Длина кода останова | → | 1 |
| Хикл/выкл | → | X ( Возможно управление) |
| Управление CS | → | H ( Возможно) |
| Автопереход при приеме | → | N |
| Автопереход при передаче | → | N |
| SI/SO | → | N |

Помнить, что возможно исключать No. устройства в отдельности.
В этом случае двоеточие после него также должно быть снято. Значения последующих цифровых параметров по умолчания будут следующими.

| | | |
|---|---|---|
| Скорость приема | → | 1200 |
| Скорость передачи | → | 1200 |
| Время ожидания | → | 0 |

Помнить, что, если скорость приема или передачи T в бодах дается с минусовым знаком, то фактическая корость связи B будет принята по следующей формуле.

B = -1843200/(T*16)

Данная инструкция - COMINI автоматически выполняется при включении компьютера ( без определения параметров), причем все параметры устанавливаются по умолчению.
Инструкция включает строку ER, но не влияет на строку RS.

(6) - COM (ON/OFF/STOP)("No. устройства: ")
Разрешает(ON), запрещает(OFF) или прекращает(STOP) выполнение программы прерывания, начатой инструкцией COM GOSUB, после приема данных в канал RS-232C.





Данная инструкция по функции очень похожа на предложение "событие {ON/OFF/STOP} ". Помнить, что - COM STOP автоматически выполняется при начале программы прерывания, а - COM ON - при первоначальном вводе RETURN. No.устройства не должно быть исключено и должно составлять 0.

(7) - COM STAT ( "No. устройства: ", переменная)
Возвращает состояние интерфейса RS-232C в целую переменную. Состояние интерфейса дается двоичной конфигурацией целого значения, присвоенного переменной.

Бит 15 Детектирует ошибку, возникшую из-за приема при заполненной буферной памяти.
1: Ошибка 0: Нет ошибки

Бит 14 Детектирует ошибку, возникшую из-за превышения временным интервалом заданной длительности ожидания (строка CS остается выключенной, а управление CS возможно).
1: Ошибка 0: Нет ошибки

Бит 13 Детектирует ошибку, возникшую из-за приема бита 0 в течение временного интервала на код останова.
1: Ошибка 0: Нет ошибки

Бит 12 Детектирует ошибку, возникшую из-за приема данных при заполненной данными буферной памяти.
1: Ошибка 0: Нет ошибки

Бит 11 Детектирует ошибку паритета.
1: Ошибка 0: Нет ошибки

Бит 10 Детектирует ошибку, возникшую при вводе CTRL + STOP
1: Ошибка 0: Нет ошибки

Бит 9 Постоянно 0.

Бит 8 Постоянно 0.

Бит 7 Индикатирует состояние строки CS.
1: вкл 0: выкл

Бит 6 Индикатирует состояние таймера и восстанавливает длительность ожидания управления CS на 10 мсек.
1: таймер вкл 0 таймер выкл

Бит 5 Постоянно 0.

Бит 4 Постоянно 0.

Бит 3 Индикатирует состояние строки DR.
1: вкл 0: выкл

Бит 2 Детектирует прием сигнала BREAK.
1: Строка RD уже настроена в состояние BREAK между концом выполнения последней и началом текущей инструкции -COMSTAT.
2: Сигнал RD не в состоянии BREAK.

Бит 1 Постоянно 0.

Бит 0 Индикатирует состояние строки RD.
1: вкл 0: выкл

Помнить, что перед подачей инструкции **-COMSTAT** файла RS-232C должен быть открыт.





**(8) -COM TERM ("No. устройства:")**
Переключает систему на терминальный режим.

**(9) EOF (No. файла)**
Возвращает -1, когда последним кодом в принимающем буфере является конец файла (&1A), и 0 в других случаях.

**(10) INPUT# No. файла, переменная[,переменная.]**
Читает данные приема, хранение между 2 делителями данных, и присваивает их к переменной. Файл, определенный номером файла, должен быть OPEN FOR INPUT.
Последонательно расположенные данные делятся следующим образом.
Цифровые переменные:
Инструкция читает данные, начиная с первого знака (начальный пробел игнорируется) и кончая пробелом, запятой, кодом CR или CR+LF.
Переменные слов цепочки:
Инструкция читает данные, начиная с первого знака (начальный пробел игнорируется) и кончая запятой, кодом CR или CR+LF.

Примечание:

Код LF считается делителем данных только тогда, когда этот код непосредственно следует за кодом CR.

Данные по типу должны соответствовать переменным.
Если управление Хвкл/ выкл сделано возможным посредством **-COMINI**, то код Хвкл передается и выключает строку RS при наполнении буферной памяти до 113 байт. При освобождении буфера до уменьшения занятой емкости до 2 байт (путем присвоения данных к переменным) код Хвкл выдается и включает строку RS.
Если управление CS также сделано возможным посредством **-COMINI**, то код Хвкл/ выкл передается только после включения строки CS.

(11) LINE INPUT# No. файла, переменная
Функционально похожа на INPUT#, однако принимает только данные цепочки слов ( переменные должны быть переменными слов), причем делителем данных считается только парный код CR+LF. Данная инструкция используется для чтения программы на Бейсике, переданой в формате ASCII, поскольку цепочка данных может включать в себя запятые.

(12) LOAD"COM [No. устройства]:"[,R]
Стирает таблицу инструкций программ и таблицу переменных и загружает программу, принятую в таблицу инструкций программ. Опция R пускает программу без закрытия открытых файлов. Коды управления Хвкл/ выкл и CS действуют также, как и инструкции INPUT#.

(13) LOC(No. файла)
Возвращает число занятых байтов в принимающей буферной памяти.

(14) LOF (No. файла)
Возвращает число свободных байтов во буфере.

(15) MERGE"COM [No. устройства]:"
Заменяет строки текущей программы строками принятой программы. После выполнения инструкции MERGE строка RS включается.
Если управление Хвкл/ выкл сделано возможным посредством -COMINI, то код Хвкл передается после приема кода CR и выключает строку RS. Когда свободная память во буфере уменьшена до 2 байт и последняя строка принятой программы записывается в PIT, строка RS включается и код Хвкл выдается. Если управление CS также сделано возможным посредством -COMINI, то коды Хвкл/ выкл передаются только после включения строки CS.





(16) OPEN"COM [No. устройства]:"[FOR режим]AS[#]No. файла
Открывает файл RS-232C, делая возможным использовать следующие инструкции ( с применением одинакового No. файла).

INPUT# PRINT# COMDTR COMSTAT COMGOSUB COMBREAK

При необходимости открытия других файлов для записи на дисках, индикации в графическом экране и др. использовать MAXFILES и определить максимальное число одновременно открываемых файлов.
Режим возможно выбрать из следующих 3.

| | |
|---|---|
| INPUT( прием) | Буферная память 127 байт предусмотрена. |
| OUTPUT( передача) | После выполнения CLOSE код EOF выдается. |
| Без определения | Одинаковое число файлов для приема и передачи |

Для того, чтобы открывать другой файл под одинаковым No. файла, необходимо сначала выполнить инструкцию CLOSE.
Строка RS включается после выполнения OPEN.

(17) PRINT# No. файла, [выражение[{,/;}][выражение...]]
Передает значение выражения (-ний), которое было бы индикатировано на экране при подаче инструкции PRINT.
Когда более 1 выражения передаются, они принимаются инструкцией LINE INPUT# как 1 выражение. Если последнее выражение не заканчивается запятой или точкой с запятой, то код CR+LF передается за исключением тех случаев, когда автоперенос на следующую строку при передаче заранее задан инструкцией -COMINI и поэтому передается только CR.

Если последнее выражение заканчивается запятой или точкой с запятой, то код CR+LF не передается, а пробел передается вместо запятой.
Если управление CS сделано возможным посредством –COMINI, то передача происходит только после включения строки CS.

(18) PRINT# No. файла USING "формат"; выражение[[{,/;}] [ выражение#...]
Имеет одинаковую функцию, что PRINT#, однако позволяет определять формат данных более подробно ( см. PRINT USING).

(19) RUN"COM [No. устройства]:"
Загружает и автоматически запускает программу, принятую в формате ASCII. Выполнение программы начинается после приема кода EOF(&H1A).
RUN включает строку RS.
Если управление Хвкл/ выкл сделано возможным посредством –COMINI, то код Хвкл передается и выключает строку RS при наполнении буферной памяти до 113 байт.
Когда данные переданы к PIT и буфер освобождается до занятой емкости 2 байт, строка RS включается и код Хвкл выдается. Если управление CS также сделано возможным, то коды Хвкл/ выкл передаются только после включения строки CS.

(20) SAVE"COM [No. устройства]:"
Передает текущую программу. К концу программы код EOF передается, коды &H00, разделяющие строки, преобразовываются в парный код CR+LF для обеспечения соответствия стандарту ASCII. Однако, если автоперенос на следующую строку при передаче заранее задан посредством –COMINI, передается только код CR.
Скорость передачи выше 9600 бод может привести к неисправностям. SAVE включает строку RS. Если управление CS сделано возможным, то передача происходит после включения строки CS.