**MP/M II™**

**Операционная система**

**РУКОВОДСТВО ПРОГРАММИСТА**

Авторское право © 1981

Digital Research

P.O. Box 579

801 Lighthouse Avenue

Pacific Grove, CA 93950

(408) 649-3896

TWX 910 360 5001

Все права защищены

**Авторское право**

Авторское право 1981 Digital Research. Все права защищены. Любая часть этой публикации не может быть воспроизведена, передана, записана, сохранена в поисковой системе или переведена на любой язык или компьютерный язык, в любой форме или каким-либо образом, электронным, механическим, магнитным, оптическим, химическим, ручным или иначе, без предварительного письменного разрешения Digital Research, Post Office Box 579, Pacific Grove, California, 93950.

Однако, это руководство является по природе учебником. Поэтому читателю предоставляется разрешение включать примеры программ полностью или частично в его собственные программы.

**Исключение ответственности**

Digital Research не дает никаких заверений или гарантий в отношении содержания данного документа, включая любые гарантии товарности или пригодности для какой-либо конкретной цели. Кроме того, Digital Research сохраняет за собой право пересмотреть эту публикацию и время от времени вносить изменения в содержание настоящего документа без обязательства Digital Research уведомить любое лицо, о таком пересмотре или изменениях.

**Торговые марки**

CP/M® - зарегистрированная торговая марка Digital Research. CP/NET, MP/M, MP/M II являются торговыми марками Digital Research.

"Руководство программиста MP/M II™ было подготовлено с использованием средства форматирования текста TEX-80 Digital Research и распечатано Commercial Press/Monterey в Соединенных Штатах Америки.

Первое издание: сентябрь 1981

Второе издание: август 1982

**Предисловие**

MP/M II™ многопользовательская операционная система для микрокомпьютеров, которая использует Intel 8080, Zilog Z80 или аналогичную 8-разрядную архитектуру. Она поддерживает многотерминальный доступ с мультипрограммированием в каждом терминале. Она использует ту же базовую дисковую операционную систему (BDOS), что и CP/M, таким образом, обеспечивает совместимость с существующими программами, работающими в CP/M.

Минимальными требованиями к оборудованию для работы MP/M II являются: процессор 8080 или Z80, 32K байт оперативной памяти (ОЗУ), системная консоль и часы реального времени. Типичное ядро MP/M II занимает приблизительно 15K байт.

Это руководство описывает интерфейс программирования MP/M II. Оно содержит общее описание модулей, из которых состоит операционная система, порядок, в котором MP/M II управляет ресурсами памяти и контролирует запуск процессов, а также подробное описание всех точек входа системы. Также включены описания нескольких служебных программ, которые полезны для создания и отладки программ в MP/M II. Данное руководство не предназначено для использования в качестве учебника. Таким образом, знакомясь с материалом, описанным в руководстве пользователя, знание архитектуры процессора и языка ассемблера в целом не требуется.

**Содержание**

Раздел 1 Введение в MP/M II 7

1.1 Обзор функций MP/M II 7

1.2 Ядро MP/M II 9

1.2.1 Диспетчеризация процессов 9

1.2.2 Управление очередями 11

1.2.3 Управление флагами 12

1.2.4 Опрос устройств 12

1.2.5 Управление устройством консоли и печати 13

1.2.6 Управление памятью 13

1.2.7 Функции системного времени 14

1.3 Структура памяти MP/M II 15

1.4 Процесс терминальных сообщений 18

1.5 Интерпретатор командной строки 19

1.6 Транзитные программы 21

1.7 Резидентные системные процессы 22

1.8 Соглашения о вызовах BDOS и XDOS 24

Раздел 2 Интерфейс BDOS 25

2.1 Интерфейс BDOS устройств ввода-вывода консоли и печати 25

2.2 Файловая система BDOS 26

2.2.1 Соглашения о присвоении имен файлам 28

2.2.2 Дисковод и структура файлов 29

2.2.3 Определение блока управления файлом 30

2.2.4 Соглашения о номере пользователя 34

2.2.5 Метки каталога и XFCB 35

2.2.6 Пароли файла 36

2.2.7 Метки времени и даты файла 37

2.2.8 Режимы открытия файла 38

2.2.9 Защита файлов 40

2.2.10 Одновременный доступ к файлам 41

2.2.11 Многосекторный ввод-вывод 43

2.2.12 Блокирование и разблокирование XIOS 43

2.2.13 Сброс, доступ и освобождение диска 44

2.2.14 Обработка ошибок BDOS 46

2.3 Инициализации базовой страницы 51

2.4 Вызовы функций BDOS 54

Раздел 3 Интерфейс XDOS 94

3.1 Введение 94

3.2 Структура данных дескриптора процесса 94

3.3 Структуры данных очереди 97

3.3.1 Циклические очереди 98

3.3.2 Связные очереди 99

3.3.3 Блок управления очередью пользователя 100

3.3.4 Соглашения о присвоении имен очереди 100

3.4 Структура данных дескриптора памяти 101

3.5 Системная страница данных 101

3.6 Сегмент внутренний данных XDOS 103

3.7 Обработка ошибок XDOS 104

3.8 Вызовы функции XDOS 104

Раздел 4 Ассемблер 8080 ASM 120

4.1 Обзор 120

4.2 Формат программы 121

4.3 Формирование операнда 122

4.3.1 Метки 122

4.3.2 Числовые константы 123

4.3.3 Зарезервированные слова 123

4.3.4 Строковые константы 124

4.3.5 Арифметические и логические операторы 125

4.3.6 Приоритет операторов 126

4.4 Директивы ассемблера 126

4.4.1 Директива ORG 127

4.4.2 Директива END 127

4.4.3 Директива EQU 128

4.4.4 Директива SET 128

4.4.5 Директивы IF и ENDIF 129

4.4.6 Директива DB 130

4.4.7 Директива DW 130

4.4.8 Директива DS 130

4.5 Коды операций 131

4.6 Сообщения об ошибках 134

Раздел 5 RDT 136

5.1 Обзор RDT 136

5.2 Вызов RDT 136

5.3 Соглашения о записи команд RDT 136

5.4 Завершение RDT 137

5.5 Команды RDT 138

5.5.1 Команда A (ассемблирование) 138

5.5.2 Команда B (Установка/сброс битов битового массива) 138

5.5.3 Команда D (Отображение) 138

5.5.4 Команда F (Заполнение) 139

5.5.5 Команда G (Перейти к выполнению) 139

5.5.6 Команда I (Ввод хвоста команды) 140

5.5.7 Команда L (Список) 140

5.5.8 Команда M (Перемещение) 140

5.5.9 Команда N (Нормализовать) 140

5.5.10 Команда R (Чтение) 141

5.5.11 Команда S (Установка) 141

5.5.12 Команда T (Трассировка) 142

5.5.13 Команда U (Без трассировки) 142

5.5.14 Команда V (Значение) 143

5.5.15 Команда W (Запись) 143

5.5.16 Команда X (Изучение состояния ЦП) 143

Раздел 6 Другие программные утилиты 145

6.1 GENHEX 145

6.2 GENMOD 145

6.3 PRLCOM 145

6.4 DUMP 146

6.5 LOAD 146

Раздел 7 Создание файла PRL 147

7.1 Формат PRL 147

7.2 Генерация PRL 147

Раздел 8 Создание RSP 149

8.1 RSP и резидентные системные процедуры 149

8.2 Генерация RSP 149

8.3 Код RSP 149

8.4 RSP с коммутацией банков 150

Раздел 9 Создание SPR 151

9.1 Системные странично-перемещаемые файлы 151

9.2 Генерация SPR 151

Приложение A Назначения флагов 152

Приложение B Назначения приоритетов процесса 152

Приложение C Перечень функции BDOS 153

Приложение D Перечень функции XDOS 155

Приложение E Пример странично-перемещаемой программы 156

Приложение F Пример резидентного системного процесса 160

Приложение G Сокращения и обозначения 163

Приложение H Глоссарий 164

Приложение I ASCII и шестнадцатеричные преобразования 168

Алфавитный указатель 170

Примечания к выпуску 174

Расширенная блокировка файла 175

Атрибуты совместимости 177

Рекомендации по программированию 180

Подавление отображения загрузчика MP/M™ 183

Установка/сброс режима необработанного ввода-вывода консоли 184

Изменение минимального размера буфера PRL файла 185

Доступ к внутренним TOD MP/M™ 186

Дисковые контроллеры с DMA в системах с коммутацией банков 187

Использование функции Send CLI Command 189

Создание submit файла из прикладной программы 191

Совместный доступ к файлам 193

Программное управление переключателем Ctrl-P 194

Автозагрузка при холодном старте 195

Усовершенствования SUBMIT 196

Модификации утилиты SPOOL 197

Блокирование/Разблокирование записей 198

Усовершенствования GENSYS 199

Изменение приоритета SPOOL.PRL 200

Изменение функций клавиш BACKSPACE и RUBOUT 201

# Введение в MP/M II

## Обзор функций MP/M II

MP/M II является операционной системой для микрокомпьютеров, которая поддерживает несколько терминалов с несколькими выполняющимися программами в каждом терминале. Совместимая снизу вверх с CP/M, система MP/M II предоставляет интерфейс CP/M каждому терминалу. Фактически, большинство программ CP/M без модификации могут работать в MP/M II. Однако, MP/M II не ограничивается этой моделью. Используя мощные возможности многозадачности, MP/M II в одном терминале может выполняться несколько программ. Кроме того, системные функции, используемые MP/M II для управления многозадачной средой, доступны прикладным программам. В результате, MP/M II поддерживает дополнительные функции, отсутствующие в CP/M, такие как межпрограммные связи и синхронизацию независимо работающих программ.

В MP/M II существует важное различие между программой и процессом. Про-грамма просто блок кода, находящийся где-нибудь в памяти или на диске, она чрезвычайно статична. С другой стороны, процесс является динамичным, и может рассматриваться как "логическая машина", которая не только выполняет код программы, но также выполняет код операционной системы. Когда MP/M II загружает программу, она также создает процесс, который связан с загруженной программой. Впоследствии, этот процесс, а не программа управляет доступом ко всем ресурсам системы. Таким образом, MP/M II следит за процессом, а не программой. Это различие является тонким, но жизненно важным для понимания работы системы в целом.

Программы, работающие в MP/M II, попадают в три категории: программы CP/M, системные процессы MP/M II и резидентные системные процессы MP/M II. Первая категория состоит из CP/M*-*подобных программ, которые MP/M II загружает в доступные сегменты памяти. MP/M II поддерживает от 1 до 7 сегментов памяти или разделов, в которые могут быть загружены программы. После загрузки и инициализации, программа связывается с процессом, который обслуживается ядром реального времени MP/M II.

Вторая категория состоит из системных процессов MP/M II, которые выполняют задачи операционной системы. Например, интерпретатор командной строки (CLI) является системным процессом, который загружает и инициирует пользовательские программы.

Последняя категория состоит из процессов, которые могут быть дополнительно интегрированы в MP/M II во время создания системы, таким образом став ее частью. Эти процессы получили название *резидентных системных процессов* (RSP). С помощью RSP, пользователи могут писать пользовательские процессы и включить их в систему вместе с поставляемыми с MP/M II (см. [Раздел 1.7](#_Резидентные_системные_процессы) и [Раздел 8](#_Создание_RSP)). Все процессы, запущенные в MP/M II соревнуются за процессор и другие системные ресурсы на приоритетной основе под управлением ядра реального времени.

В следующем списке кратко изложены возможности MP/M II.

* *Многотерминальная поддержка*. MP/M II поддерживает до 16 терминалов. Кроме того, один процесс может получить доступ к нескольким терминалам.
* *Многозадачность в каждом терминале*. Любая системная консоль может инициировать несколько программ или процессов. Кроме того, процесс может генерировать подпроцессы.
* *Поддержка памяти с коммутацией банков*. Сегменты памяти MP/M II могут находиться в общей памяти, или быть распределены по отдельным сегментам памяти, таким образом, эффективно расширяя объем памяти системы.
* *Связи между процессами, синхронизация и взаимное исключение*. Эти функции обеспечиваются очередями.
* *Механизм логических прерываний с помощью флагов*. Он обеспечивает MP/M II интерфейс с любой структурой физических прерываний.
* *Функции системного времени*. Эти функции позволяют процессам, работающим в MP/M II, вычислять истекшее время, выполнять временные задержки указанных интервалов, и получить доступ и назначить текущую дату и время. Кроме того, пользователь может запланировать выполнение программ по дате и времени. Также, системное время используется для циклического планирования выполнения вычислительных процессов с одинаковыми приоритетами.
* *Выбираемые пользователем параметры во время генерации системы*. Доступны следующие параметры: количество системных консолей, количество, раз-мер и расположение сегментов памяти и максимальное число файлов и заблокированных записей, поддерживаемых системой одновременно. Кроме того, пользователь может выбрать какие резидентные системные процессы (RSP) включать в MP/M II во время генерации системы.

Функционально, MP/M II состоит из трех различных модулей: *базовой дисковой операционной системы* (BDOS), *расширенной дисковой операционной системы* (XDOS) и *расширенной системы ввода-вывода* (XIOS). BDOS MP/M II является совместимой снизу вверх с однопользовательской версией BDOS CP/M. В большинстве случаев программы CP/M, которые делают вызовы ввода-вывода BDOS, или прямые вызовы BIOS к консолям ввода-вывода и принтеру, могут работать в MP/M II без изменений. Однако, BDOS MP/M II расширена для обеспечения поддержки нескольких консолей и устройств печати (принтеров). Кроме того, расширена файловая система для предоставления функций, необходимых в многопользовательских средах.

Два основных расширения файловой системы:

* *Блокировка файлов*. Обычно файлы открытые в МР/М II не могут быть открыты или удалены другими пользователями. Эта функция предотвращает случайные конфликты с другими пользователями.
* *Совместный доступ к файлам*. В качестве специальной опции, независимые пользователи могут открыть один и тот же файл в совместно используемом или неблокируемом режиме. MP/M II поддерживает команды блокирования и разблокирования файлов, открытых в этом режиме, и защищает файлы, открытые в режиме совместного доступа от удаления другими пользователями.

Модуль XDOS обеспечивает MP/M II ее многозадачные возможности. Он содержит ядро реального времени, которое контролирует выполнение процессов и разрешает конфликты при распределении ресурсов системы. Он также включает *процесс терминальных сообщений* (TMP), который читает и повторяет командные строки в системных консолях и *интерпретатор командной строки* (CLI), который принимает командные строки TMP и инициирует пользовательские программы и резидентные системные процессы (RSP). XDOS также содержит набор расширенных функций MP/M II, к которым могут получить доступ пользовательские программы.

Модуль XIOS похож на модуль BIOS CP/M, но расширен в нескольких направлениях. Примитивные функции, такие как консольный ввод-вывод, изменены для поддержки нескольких консолей. Несколько новых примитивных функций поддерживают дополнительные функции MP/M II. Кроме того, добавлены новые возможности для устранения циклов задержки. XIOS является аппаратно-зависимым модулем, который согласовывает интерфейс MP/M II с конкретной аппаратной средой. Несмотря на то, что стандартный XIOS предоставлен Digital Research, XIOS обычно настраивается для поддержки аппаратной среды пользователя. Примечание: процессы, работающие в MP/M II, могут выполнить прямые вызовы XIOS только для ввода-вывода на консоль и устройства печати.

Когда MP/M II настроена для одной консоли и выполнения одной программы, ее быстродействие приближается к CP/M. Накладные расходы диспетчера MP/M II в такой среде составляют 7-15%. В средах, где выполняются несколько процессов или пользователей скорость выполнения каждого отдельного процесса снижается пропорционально количеству операций ввода-вывода и вычислительных ресурсов. Процесс, который выполняет большое количество операций ввода-вывода относительно вычислений показывает только незначительное уменьшение быстродействия. Это также относится к процессу, который выполняет большой объем вычислений, но выполняется одновременно с другими процессами, которые осуществляют много операций ввода-вывода. Однако, в тех средах, где выполняется более чем один вычислительный процесс происходит значительное уменьшение быстродействия.

## Ядро MP/M II

MP/M II управляется многозадачным ядром реального времени, которое находится в модуле XDOS. Это ядро выполняет диспетчеризацию процессов, управление памятью и системную синхронизацию задач. Оно также выполняет управление очередями, управление флагами, опрос устройств и управление консолями и устройствами печати. Следующие разделы описывают эти функции более подробно. Многие из системных функций, которые выполняют эти задачи также могут быть вызваны пользовательскими программами с помощью функций XDOS.

Несмотря на то, что MP/M II является многозадачной операционной системой в любой конкретный момент времени только один процесс имеет доступ к ресурсу процессора. Если ему специально не указано осуществлять взаимодействие или синхронизировать выполнение с другими процессами, он работает не зная, что другие процессы могут конкурировать за ресурсы системы. В итоге система приостанавливает выполнение процесса и дает возможность запустить другой процесс.

### Диспетчеризация процессов

Основной задачей ядра является передача ресурса центрального процессора (ЦП) от одного процесса к другому. Эта задача называется *диспетчеризацией* и осуществляется частью ядра называемым *диспетчером*. В MP/M II, каждый процесс связан со структурой данных, названной *дескриптором процесса* (см. [Раздел 3.2](#_Структура_данных_дескриптора_1)), Диспетчер использует эту структуру данных для сохранения и восстановления состояния текущего рабочего процесса. Каждый процесс в системе находится в одном из трех состояний: *готовый*, *рабочий* или *приостановленный*. Готовый процесс один из ожидающих ресурса ЦП. Приостановленный процесс один из ожидающих некоторого другого системного ресурса или определённого события. Рабочий процесс - тот, который в настоящее время выполняет процессор.

Операция диспетчеризации выполняющегося процесса может быть описана следующим образом:

1. Диспетчер приостанавливает выполнение процесса и сохраняет текущее состояние в дескрипторе процесса.
2. Диспетчер сканирует все приостановленные процессы в списке готовых и выбирает один с самым высоким приоритетом.
3. Диспетчер восстанавливает состояние выбранного процесса из его дескриптора процесса и предоставляет ему ресурс ЦП.
4. Процесс выполняется до тех пор, пока он не сделает системный вызов или не произойдет прерывание, или такт (эталонный интервал) системных часов. Затем, диспетчеризации повторяется.

Только процессы, помещенные в список готовых (*Ready List*), могут быть выбраны во время диспетчеризации. По определению процесс находится в списке готовых, если он ожидает только ресурс ЦП. Процессы, ожидающие другие системные ресурсы, не могут выполняться, пока их требования к ресурсам не будут удовлетворены. В системе MP/M II, процесс блокируется от выполнения, если он ожидает:

* Сообщение из очереди, так что он может завершить операцию чтения очереди.
* Места, чтобы стать доступным в очереди, поэтому он может завершить операцию записи очереди.
* Установки системного флага.
* Доступности консоли или устройства печати.
* Конкретного количества тактов системных часов, прежде чем он сможет быть удален из системного списка приостановленных процессов.
* Завершения события ввода-вывода.

Эти ситуации рассматриваются более подробно в следующих разделах.

MP/M II является системой с управлением на основе приоритетов. Это означает, что диспетчер выбирает готовый процесс с самым высоким приоритетом и предоставляет ему ресурс ЦП. Выполнение процессов с одинаковым приоритетом планируется с использованием "циклического алгоритма" (round-robin). Т.е. им предоставляются равные кванты процессорного времени при выполнении кода процессором. При приоритетной диспетчеризации управление никогда не передается процессу с более низким приоритетном, если в списке готовых процессов присутствует процесс с более высоким приоритетом. Поскольку выполнение вычислительных процессов с высоким приоритетом, имеет тенденцию монополизировать ресурс ЦП, желательно понизить их приоритет во избежание снижения общей производительности системы. Кроме того, вычислительные процессы, могут периодически выполнить вызовы диспетчеризации XDOS, чтобы содействовать совместному использованию ресурса ЦП в системах, которые не поддерживают часы. Когда процесс выполняет вызов диспетчеризации, вызов создает процесс с пустой операцией, но позволяет другим процессам получать доступ к ресурсу ЦП.

MP/M II требует, чтобы в любом случае работал по крайней мере один процесс. Чтобы гарантировать это, система поддерживает процесс IDLE (ожидания) в списке готовых процессов, таким образом, он может быть диспетчеризирован, при отсутствии других доступных процессов. Процесс IDLE работает с очень низким приоритетом и никогда не блокируется от исполнения. Он не выполняет полезную задачу, а просто обеспечивает системе процесс для выполнения при отсутствии других готовых процессов.

### Управление очередями

Очереди выполняют несколько важных функций для процессов, работающих в MP/M II. Они используются для передачи сообщений между процессами, для выполнения синхронизации процессов и для взаимного исключения. Очереди являются специальными структурами данных, реализованными в MP/M II как "файлы в памяти", которые содержат пространство для конкретного числа сообщений фиксированной длины (см. [Раздел 3.3](#_Структуры_данных_очереди_1)). Как и файлы, очереди могут быть созданы, открыты, удалены, прочитаны из, и записаны в при помощью вызовов функций XDOS. При создании очереди с помощью команды XDOS *Make Queue*, ей присваивается имя из 8 символов, которое идентифицирует очередь в командах XDOS *Open Queue* (открытие очереди). Как и предполагает название, сообщения считываются из очереди на основе метода "первым пришел - первым вышел".

Процесс может считать сообщения из очереди или записать сообщения в очередь двумя способами: *условно* или *безусловно*. Если в очереди не существует ни одного сообщения, когда выполняется условное чтение, или очередь переполнена когда выполняется условная запись, система возвращает процессу вызова код ошибки. Однако, если процесс выполняет безусловное чтение из пустой очереди, система приостанавливает процесс выполнения, пока другой процесс не запишет сообщение в очередь. Процесс, приостановленный таким способом, помещается в *список исключенных из очереди* (Dequeue list). Аналогичная ситуация возникает, когда процесс делает безусловную запись в полную очередь. Процесс, приостановленный таким образом, помещается в *список поставленных в очередь* (Enqueue list). MP/M II использует списки Enqueue/Dequeue для синхронизации выполнения процессов.

Если несколько процессов находится в очереди в списке постановки в очередь или списке исключения из очереди, предпочтение отдается процессам с более высоким приоритетом. Конфликты с участием процессов с одинаковым приоритетом разрешаются на основе принципа первый пришел первый обслужен.

*Очереди взаимного исключения* в MP/M II представляют собой специальный тип очереди. Они содержат одно сообщение нулевой длины и присвоенное имя, начинающееся с прописных букв MX. В действительности, очередь взаимного исключения является *двоичным семафором*. Очереди взаимного исключения одновременно обеспечивают доступ к ресурсу только одному процессу. Доступ к ресурсу, защищенному очередью взаимного исключения происходит следующим образом:

1. Процесс выдает безусловной вызов чтения очереди из очереди, защищающей ресурс, тем самым приостанавливая себя, пока сообщение не станет доступно.
2. Процесс получает доступ к защищенному ресурсу.
3. Процесс записывает сообщение обратно в очередь, когда он закончил использовать защищенный ресурс, таким образом освобождая ресурс для других процессов.

Например, очередь взаимного исключения дисковой системы MXdisk, гарантирует, что процессы последовательно получают доступ к файловой системе BDOS.

У очередей взаимного исключения есть еще одна функция, которая отличается от обычных очередей. Когда процесс считывает сообщение из очереди взаимного исключения, ядро сохраняет адрес дескриптора процесса для процесса чтения сообщения в область 2 байтового буфера очереди. Если процесс прерывается, пока он владеет сообщением взаимного исключения, ядро автоматически записывает сообщение обратно в очередь для прерывания процесса, таким образом позволяя другим процессам получить доступ к защищенному ресурсу.

### Управление флагами

Ядро MP/M II использует флаги для передачи сигналов и синхронизации процессов с определенными событиями. Процессы получают доступ к флагам системы с помощью функций XDOS: *Flag Set* (Установка флага) и *Flag Wait* (Ожидание флага). По сути дела, флаг может находиться в двух состояниях: *установлен* или *сброшен*. Состояние сброса делится на две категории:

* Отсутствие процесса ожидающего установки флага.
* Процесс ожидает установки флага и заблокирован от выполнения, пока он не установлен.

Примечание: два процесса не могут ожидать один и тот же флаг. Эта ошибочная ситуация, называется "under-run" (недогрузка) флага. Аналогично, попытка процесса установить уже установленный флаг, является другой ошибочной ситуацией, названной "over-run" (переполнение) флага.

Флаги обеспечивают систему логических прерываний независимо от системы физических прерываний микрокомпьютера. Прежде всего они предназначены для использования модулем XIOS для поддержки обработчика прерываний. Например, когда обработчик прерываний получает физическое прерывание, указывающее о завершении операции ввода-вывода, он устанавливает флаг и осуществляет переход к диспетчеру. Выполнение процесса приостанавливается, потому что он ожидает установки флага, он помещается в список готовых процессов, что делает его пригодным для выбора во время диспетчеризации. После диспетчеризации, процесс может предположить, что операция ввода-вывода завершена.

MP/M II поддерживает 32 флага, часть из которых зарезервирована. Например, флаг 1 зарезервирован для тактов системных часов. Из-за их ограниченного количества, использования их модулем XIOS и однопроцессной природе их конструкции - флаги не должны использоваться в прикладных программах, кроме совершенно особых ситуаций. В большинстве случаев взаимодействие и синхронизацию процессов лучше выполнять с помощью очередей.

### Опрос устройств

Опрос устройства является еще одним механизмом, который процесс может использовать для ожидания ввода-вывода или внешнего события без использования флагов или использования ресурса процессора программным циклом задержки. Опрос реализуется исключительно в модуле XIOS. Например, предполагая, что XIOS поддерживает опрос консольного ввода, когда процесс выполняет вызов BDOS ввод с консоли, процесс в конечном счете достигает подпрограммы XIOS ввода с консоли, которая фактическая выполняет аппаратно-зависимые операции ввода. Прежде, чем выполнить операцию ввода, ядро проверяет готовность символа к вводу. Если он готов, ядро выполняет операцию ввода, и выполнение процесса продолжается. Если символ не готов, процесс должен ожидать его. В однопользовательской среде CP/M BIOS может просто в цикле опрашивать состояние консоли, пока символ не будет считан. В MP/M II этот метод не может быть задействован, потому что он использует ресурс ЦП. Если у цикличного процесса высокий приоритет, любым другим процессам с более низким приоритетом в списке готовых процессов будет отказано в предоставлении ресурса ЦП.

Опрос устройства позволяет избежать этой ситуации, потому, что проверку состояния консоли осуществляет диспетчер. Если символ не готов, XIOS выполняет вызов опроса XDOS. Он приостанавливает выполнение процесса, помещая его в системный *список опроса*. Впоследствии, при каждой операции диспетчеризации, диспетчер позволяет однократно выполнить вызов процесса опроса состояния консоли. Когда вызов опроса состояния указывает, что символ готов, ядро удаляет процесс из списка опроса и помещает его в список готовых процессов. Таким образом, опрос устройства является один из способов, с помощью которого процесс может ожидать наступления внешнего события или события ввода-вывода не монополизируя ресурс ЦП.

### Управление устройством консоли и печати

В MP/M II консоль и устройство печати являются особыми ресурсами. Когда система предоставляет консоль или устройство печати процессу, она внутренне сохраняет адрес дескриптора процесса, таким образом, регистрируя владение устройством этим процессом. Если другой процесс пытается использовать устройство, ядро приостанавливает процесс вызова и помещает его в *список ожидания устройства*. Оно остается в этом списке пока процесс, владеющий устройством, не завершит выполнение или не отсоединится от устройства. Когда это происходит, ядро выбирает ожидающий процесс с самым высоким приоритетом, предоставляет ему устройство, помещает его в список готовых процессов и выполняет диспетчеризацию.

Процессы могут владеть несколькими консолями или устройствами печати. Поля в дескрипторе процесса определяют, какое устройство должно использоваться в операциях ввода-вывода. Процесс приобретает право собственности на устройства с помощью механизма, называемого *присоединением* (attaching). Если процесс присоединяет устройство, когда устройство свободно, процесс приобретает право собственности на устройство. В противном случае, процесс приостанавливает выполнение, как описано выше. Как вариант, процесс можно условно присоединить к устройству, в этом случае он уведомляется, если другой процесс владеет устройством. Условное присоединение дает процессу больший контроль над своим собственным выполнением вместо того, чтобы предоставить его ядру. Таким образом, процесс может избежать приостановки, когда он не зависит от конкретного устройства.

### Управление памятью

Ядро MP/M II может управлять от одного до восьми сегментами памяти. Эти сегменты имеют фиксированную длину, и используются, прежде всего, в качестве областей для загрузки транзитных программ. Разделы являются выровненными страницами, что означает, что они должны начинаться на границе страницы. Поскольку страница определяется как 256 байт, граница страницы всегда начинается по адресу, у которого младший байт равен 0. Ядро управляет ресурсом памяти с помощью функций XDOS, которые выделяют и освобождают сегменты памяти. *Рисунок 1-1* иллюстрирует, как организована память в MP/M II.

Заштрихованные области представляют те области, которые могут поддерживать сегменты памяти. Если память с коммутацией банков не используется, доступная память ограничена нулевым банком. Общее количество сегментов памяти, в дополнение к их размеру и местонахождению банка, являются параметрами генерации системы. Сегмент 0, однако, является специальным сегментом, зарезервированным для системных модулей и RSP. Он всегда находится непосредственно под областью операционной системы в банке 0.

Верхняя граница памяти

┌──────────────┐

│ Операционная │

(Общий) │ система │

│ MP/M II │

│ │

Верхняя граница ├──────────────┤ ┌────────┐ ┌────────┐

переключаемой │ │ │////////│ │////////│

памяти │ Сегмент 0 │ │////////│ │////////│

│ │ │////////│ │////////│

│ │ │////////│ │////////│

│ │ │////////│ │////////│

│ Расширение │ │////////│ │////////│

(Коммутируемый │ MP/M II │ │////////│ │////////│

банк) ├──────────────┤ │////////│ │////////│

│//////////////│ │////////│ │////////│

│//////////////│ │////////│ │////////│

│//////////////│ │////////│ │////////│

Нижняя область │//////////////│ │////////│ │////////│

памяти │//////////////│ │////////│ │////////│

│//////////////│ │////////│ │////////│

└──────────────┘ └────────┘ └────────┘

Банк 0 Банк 1 .... Банк N

Рисунок ‑ Организаций памяти MP/M II

В системах с коммутацией банков модуль операционной системы находится в общей памяти. Кроме того, все дескрипторы процессов и очереди должны находиться в области общей памяти. Как правило, объем общей памяти - 16K, но размер может измениться в системах, способных к переключению памяти в модулях, меньших, чем 16K. В результате типичный максимальный размер сегмента памяти - 48K. Самый большой сегмент пользовательской памяти, который может быть выделен для банка 0, как правило, значительно меньше этого значения.

Более одного сегмента памяти могут быть определены в одном банке. Сегменты памяти, которые не начинаются с 0, могут использоваться только для выполнения *странично-перемещаемых программ* (PRL). Сегменты памяти, начинающиеся с 0, могут выполнить программы COM или PRL.

### Функции системного времени

Функции системного времени MP/M II включают: сохранение времени суток, задержки выполнения процесса в течение установленного периода времени и планирование загрузки программ с диска и выполнения. Внутренний процесс XDOS, CLOCK, обеспечивает время суток в системе. Этот процесс выдает вызов *Flag Wait* ожидание флага односекундного системного флага (флага 2). Когда обработчик прерываний XIOS устанавливает этот флаг, он будит процесс CLOCK, который увеличивает внутреннее время и дату. Затем, процесс CLOCK делает еще один вызов ожидание флага и приостанавливает себя, пока флаг не установлен снова. MP/M II обеспечивает функции, которые позволяют пользователю устанавливать и получать доступ к внутренней дате и времени. Кроме того, BDOS использует внутреннее время и дату, чтобы записать, когда файл обновлен, создан, или в последний раз к нему был получен доступ.

Функция XDOS *Delay* (задержка) заменяет типичный программный цикл задержки для задержки выполнения процесса. Функция задержки требует, чтобы такты поддерживались в XIOS и что бы флаг 1, флаг системных тактов, был установлен каждые 16 - 20 миллисекунд (обычно 60 раз в секунду). Когда процесс выполняет вызов задержки, он определяет число тактов в течении которых, должно быть приостановлено его выполнение. Система сохраняет адрес дескриптора этого процесса во внутреннем *списке задержки* вместе с его текущим счетчиком тактов задержки. Системный процесс, TICK, ожидает флаг такта и уменьшает, этот счетчик задержки с каждым системным тактом. Когда счетчик задержки обнуляется, процесс удаляется из списка задержки и помещается в список готовых процессов.

MP/M II может запланировать выполнение резидентной программы или резидентного системного процесса, только если резидентный системный процесс SCHED, включен в систему во время генерации.

## Структура памяти MP/M II

Структура памяти операционной системы MP/M II показана на *Рисунке 1-2*.

Старшие адреса ┌─────────────┐ <─────┐<───┐

памяти │ SYSTEM.DAT │ │ │

├─────────────┤ │ │

│ TMPD.DAT │ │ │

├─────────────┤ │ │

│ USERSYS.STK │ │ │

├─────────────┤ │ │

│ XIOSJMP.TBL │ │ ├─── Общая

├─────────────┤ │ │ память

│ RESBDOS.SPR │ │ │

├─────────────┤ │ │

│ XDOS.SPR │ │ │

├─────────────┤ │ │

│ RSP │ │ │

├─────────────┤ │ │

│ BNKXIOS.SPR │ │<───┘

├─────────────┤ │

│ BNKBDOS.SPR │ ├─── Сегмент памяти 0

├─────────────┤ │ Банк 0

│ BNKXDOS.SPR │ │

├─────────────┤ │

│ TMP.SPR │ │

├─────────────┤ │

│ BRS │ │

├─────────────┤ │

│ LCKLSTS.DAT │ │

Младшие адреса ├─────────────┤ │

памяти │ CONSOLE.DAT │ │

└─────────────┘ <─────┘

Рисунок ‑ Структура памяти MP/M II

Точные адреса памяти каждого из сегментов памяти, показанных выше изменяются в зависимости от версии MP/M II и зависят от параметров заданных пользователем в процессе генерации системы.

Если хост-система с коммутацией банков, модули выше модуля BNKXIOS.SPR должны находиться в общей памяти. Общая память всегда доступна независимо от используемого банка. Модули ниже модуля BNKXIOS.SPR должны находиться в банке 0, который определяется в качестве активного банка памяти при загрузке MP/M II. Сам модуль BNKXIOS.SPR может находиться частично в общей памяти и частично в банке 0. Если коммутация банков не используется, то вся память является общей. Показанные на *Рисунке 1-2* сегменты памяти описаны ниже.

Сегмент SYSTEM.DAT содержит 256 байт, используемых программой GENSYS MP/M II для динамической настройки системы. После загрузки система использует эту область для хранения системных данных, таких как submit флаги (инициализации выполнения). Смотрите [Раздел 3.5](#_Системная_страница_данных) для получения информации о сегменте SYSTEM.DAT.

Размер сегмента TMPD.DAT зависит от числа консолей в системе, указанных во время процесса генерации системы. MP/M II поддерживает от 1 до 16 консолей, и связанных с каждой консолью процессов терминальных сообщений (TMP), определенных как TMP0 - TMP15. TMP обеспечивает поддержку командной строки для каждой консоли. Каждая консоль использует 64 байта в сегменте TMPD.DAT, для хранения дескриптора процесса TMP. Размер сегмента TMPD.DAT изменяется в зависимости от числа консолей, как показано в *Таблице 1-1*.

Таблица ‑ Размер сегмента TMPD.DAT

|  |  |
| --- | --- |
| **Размер** | **Число консолей** |
| 000H | нет консолей |
| 100H | от 1 до 4 консолей |
| 200H | от 5 до 7 консолей |

Сегмент USERSYS.STK включается, если пользователь выбирает опцию добавить стек для системных вызовов пользователя во время генерации системы. Если сегмент включен, система временно использует 64 байта стекового пространства в этом сегменте когда программы пользователя делают вызовы функций BDOS. Эта опция позволяет пользователям выполнять CP/M \*.COM файлы в MP/M II. Некоторые вызовы функций BDOS, особенно функции консольного ввода-вывода, в MP/M II используют больше стека, чем в CP/M. Система выделяет место для пользовательских системных стеков из сегмента USERSYS.STK для каждого сегмента пользовательской памяти. Размер сегмента USERSYS.STK изменяется в зависимости от числа сегментов памяти, как показано в *Таблице 1-2*.

Таблица ‑ Размер сегмента USERSYS.STK

|  |  |
| --- | --- |
| **Размер** | **Число сегментов памяти** |
| 000H | в системе нет стека пользователя |
| 100H | от 1 до 4 сегментов памяти |
| 200H | от 5 до 7 сегментов памяти |

Сегмент XIOSJMP.TBL - копия первой страницы модуля BNKXIOS.SPR. Он требуется, потому что система делит модуль BDOS на два подмодуля, RESBDOS.SPR и BANKBDOS.SPR. Модуль RESBDOS получает доступ к BNKXIOS через модуль XIOSJMP.TBL. Модуль BANKBDOS получает доступ к модулю BNKXIOS непосредственно. Модуль XIOSJMP.TBL имеет размер 256 байтов.

Сегмент RESBDOS.SPR содержит резидентную часть модуля BDOS. Функции BDOS, поддерживаемые этим сегментом, включают не связанные с файловой системой BDOS, такие как ввод-вывод в консоль и на устройство печати. Сегмент RESBDOS.SPR имеет длину приблизительно 0B00H байтов.

Сегмент XDOS.SPR содержит ядро MP/M II и расширенную дисковую операционную систему. Этот сегмент имеет длину приблизительно 2300H байтов.

RSP могут использовать в MP/M II два сегмента. Первый сегмент находится в общей памяти и существует, только если один или несколько RSP включены во время генерации системы. Этот сегмент RSP в общей памяти содержит все дескрипторы процессов и очереди RSP. Второй сегмент, названный сегментом BRS, существует в необщей части сегмента памяти 0. Он присутствует только, когда один или несколько RSP с поддержкой коммутации банков включены во время генерации системы (См. [Раздел 1.7](#_Резидентные_системные_процессы_1)).

Модуль BNKXIOS.SPR содержит настроенную пользователями основную и расширенную систему ввода-вывода в странично-перемещаемом формате (PRL). Он может выйти за рамки границы общей памяти. В общем случае код, поддерживающий файловую систему BDOS, может находиться в банке 0, в то время как код поддерживающий ввод-вывод на консоль и устройство печати, должен находиться в общей памяти. Обратитесь к *Системному руководству MP/M II* для получения дополнительной информации относительно модуля BNKXIOS.

Модуль BNKBDOS.SPR содержит нерезидентную часть модуля BDOS. Все функции BDOS, связанные с файловой системой, поддерживаются этим сегментом. Этот сегмент имеет размер приблизительно 2300H байтов.

Модуль BNKXDOS.SPR содержит нерезидентную часть модуля XDOS. Размер этого сегмента изменяется с версией MP/M II.

Модуль TMP.SPR содержит реентерабельный (повторно входимый) код процесса терминальных сообщений. Этот модуль имеет размер приблизительно 300H байтов.

Сегмент BRS содержит данные и код RSP с поддержкой коммутации банков, который не должны быть в общей памяти. RSP с поддержкой коммутации банков имеют важное значение, так как они минимизируют требования к общей памяти.

Сегмент LCKLSTS.DAT - специальная структура данных, которая поддерживает запись в открытые файлы и блокировку записей в системе. Каждый открытый файл и заблокированная запись используют 10-байтовую запись в этом сегменте. Размер этого сегмента определяется параметрами, указанными во время генерации системы.

Размер сегмента CONSOLE.DAT зависит от числа консолей, определенных для системы во время процесса генерации системы. MP/M II поддерживает от 1 до 16 консолей, и связанных с каждой консолью процессов терминальных сообщений, обозначенных TMP0 - TMP15. TMP обеспечивает поддержку командной строки для каждой консоли. Каждая консоль использует 256 байтов в сегменте CONSOLE.DAT, которые содержат стек и буферы для ее TMP. Код TMP реентерабельный и располагается в сегменте TMP.SPR.

Остающаяся память доступна для выделения сегментам пользовательской памяти. Размер, расположение банка и число сегментов пользовательской памяти - параметры генерации системы. MP/M II использует эти сегменты памяти для загрузки и выполнения транзитных программ.

## Процесс терминальных сообщений

Процесс терминальных сообщений (TMP) является одним из перечня системных процессов XDOS, которые связаны с системными консолями. Каждой системной консоли назначается ее собственный TMP от TMP0 до TMP15. Количество реализованных системных консолей зависит числа, поддерживаемых XIOS и заданного количества во время генерации системы. Очевидно, что количество системных консолей, не может превышать числа поддерживаемого XIOS. Однако, меньшее число, чем максимально поддерживаемое XIOS может быть определено во время генерации системы.

Система сохраняет буферы, стек и локальные переменные для каждого TMP в области каждой системной консоли сегмента CONSOLE.DAT. Дескрипторы процесса TMP находятся в сегменте TMPD.DAT. Совместно используемый код всеми TMP, является реентерабельной подпрограммой в модуле TMP.SPR. Таким образом, хотя каждый TMP выполняет ту же функцию для каждой системной консоли, они конкурируют друг с другом, а также с любыми другими рабочими процессами за ресурс ЦП.

TMP предоставляет поддержку командной строки для системных консолей в MP/M II. Она включает отображение системного приглашения в консоли:

0A>

и чтение командной строки. TMP читает командную строку из одного из двух источников: консоли или Submit файла. Обычно, он читает из консоли с помощью команды чтения BDOS *Read Buffer*. Альтернативно, он читает из файла $N$.SUB (N = номер консоли) на системном диске MP/M II. Это происходит, только если пользователь ранее ввел команду submit в консоли с объектом SUBMIT.

После чтения командной строки TMP выполняет одно из двух действий в зависимости от типа введенной команды. Если командная строка - новая спецификация диска:

0A>**B:**

TMP выдает вызов BDOS *Select Disk* для выбора нового диска. Если система поддерживает недавно выбранный диск, TMP обновляет поле диска в своем дескрипторе процесса, отображает новую строку:

0B>

и ожидает следующую команду в командной строке.

Если команда находится в какой-либо другой форме, TMP присваивает свою консоль другому системному процессу, интерпретатору командной строки (CLI). Затем TMP отправляет командную строку вместе с полями, определяющими ее диск по умолчанию, номер пользователя, устройство печати и номер консоли в CLI с помощью команды XDOS *Send CLI*. Затем пытается присоединить консоль. Это приостанавливает выполнение TMP, потому что он больше не владеет системной консолью. Когда консоль становится свободной, TMP повторно выдает приглашение и цикл повторяется.

Примечание: на уровне команд значения по умолчанию диска и текущего номера пользователя сохраняются в дескрипторе процесса TMP для каждой системной консоли. Эта информация выводится на экран в системном приглашении. Если прикладная программа изменяет текущий диск или код пользователя, сделав явный вызов BDOS, значения в дескрипторе процесса TMP не изменяются. Утилита USER действительно обновляет код пользователя в дескрипторе процесса TMP, когда она устанавливает номер пользователя в новое значение. Чтобы сделать это, она определяет местоположение дескриптора процесса TMP, связанного с консолью, и обновляет поле номера пользователя.

## Интерпретатор командной строки

Когда интерпретатор командной строки (CLI) получает командную строку, отправленную в него с помощью XDOS *Send CLI Command*, он интерпретирует команду и инициирует запрошенную транзитную программу или RSP. Как правило, командную строку в CLI отправляет TMP. Однако другие процессы также могут использовать функцию *Send CLI Command*. Кроме того, функция BDOS *Program Chain* реализована с помощью встроенной функции *Send CLI Command*. Примечание: Любой процесс, делающий вызов *Send CLI Command*, должен сначала назначить свою консоль CLI.

Функция *Send CLI Command* отправляет командную строку в CLI, пытаясь записать сообщение командной строки в системную очередь "CLIQ". Кроме командной строки ASCII сообщение командной строки содержит текущий диск, номер пользователя, устройство печати и номер системной консоли. CLIQ является единственной очередью сообщений с длиной 129 байтов. Если CLIQ уже содержит сообщение командной строки, ядро приостанавливает процесс, выдавший *Send CLI Command* и помещает его в список поставленных в очередь (*Enqueue List*) CLIQ, в котором он остается, пока CLI не прочтет сообщение. Как только CLI прочтет сообщение, процесс должен конкурировать с любыми другими, которые могут также находиться в списке *Enqueue List* для возможности записать свое сообщение и восстановить состояние готовности. Процесс с самым высоким приоритетом, который был не первым в списке всегда выбирается первую очередь.

CLI принимает сообщения командной строки, читая CLIQ. Если очередь пуста, CLI блокируется от выполнения, когда он выдает команду чтения CLIQ. В этом случае CLI приостановлен в списке удаления из очереди (*Dequeue List*) CLIQ до тех пор, пока другой процесс не выдает *Send CLI Command*, после чего CLI удаляется из списка *Dequeue List* и помещается в список готовых процессов. Когда он получает ресурс ЦП, операция CLI чтения очереди завершена, и он получает сообщение командной строки.

Командная строка, считанная CLI, должна быть в кодах ASCII и обычно принимает форму:

<command> <command tail>

где

<command> => {d:}filename{;password} или

=> queuename

<command tail> => <file spec> или

=> <file spec><delimiter><file spec>

<file spec> => {d:}filename{.typ}{;password}

<delimiter> => один или несколько пробелов или символов

табуляции или один из следующих: "=,/[]<>"

d: => спецификация диска MP/M II, от "A" до "P"

filename => имя файла от 1 до 8 символов

typ => расширение файла от 1 до 3 символов

password => значение пароля от 1 до 8 символов

queuename => от 1 до 8 символов имени очереди

резидентного системного процесса

Поля, включенные в фигурные скобки, необязательные. Если спецификация диска {d:} не указана, предполагается текущий диск по умолчанию. Если поле расширения файла {.typ} опущено, подразумевается поле расширения полностью заполненное пробелами. Если поле пароля {;password} опущено, подразумевается поле пароля полностью заполненное пробелами. В спецификацию файла <command> поле расширения не включено, потому что CLI предполагает расширение PRL или COM.

После того, как CLI считает командную строку, он выполняет следующие шаги:

1. Анализирует командную строку, чтобы найти поле <command>.
2. Если отсутствуют спецификация диска или поле пароля, CLI пытается открыть очередь, с именем из поля команды. Если открытие очереди успешно, CLI предполагает, что очередь принадлежит RSP и пытается назначить консоль этому RSP. Если имя RSP совпадает с именем очереди, выполняется назначение консоли. Фактически, это путь которым RSP определяет, получит или нет он ресурс консоли, если инициирован с помощью CLI. Затем CLI пишет сообщение <command tail> вместе с текущим диском, номером пользователя, устройством печати и номером системной консоли в очередь RSP. Поскольку RSP обычно блокируется от выполнения из-за чтения очереди из этой очереди, эта последовательность инициирует RSP для выполнения.
3. Если поле команды не является именем очереди RSP, что обозначается неуспешным открытием очереди или присутствует спецификация диска, или поле пароля, CLI предполагает, что это имя файла, расположенного по умолчанию или на указанном диске. Тогда он пытается открыть файл, filename.PRL. Если открытие неудачно, он пытается снова с именем файла filename.COM. Если текущий номер пользователя ненулевой и открываемый файл не существует с этим номером пользователя, BDOS пытается открыть файл с номером пользователя 0. Операция открытия успешна, если файл с номером пользователя 0 существует и имеет установленный атрибут *системный*.

Если ни один не открывается успешно, и отсутствует явное указание диска, CLI повторяет ту же последовательность для системного диска MP/M II. (Системный диск определяется во время генерации системы). CLI не предпринимает эту вторую попытку, если системный диск был указан при первой попытке. Кроме того, независимо от номера пользователя у файла, только файлы с установленным атрибутом системный принимаются во второй последовательности открытия.

Во всех случаях, если пароль файла, определенный в поле <command>, не соответствует паролю файла, защищенному в режиме чтения, операция открытия CLI завершается с ошибкой пароля.

1. Если командный файл открыт успешно, CLI выполняет различные действия в зависимости от того, имеет ли открытый файл расширение PRL или COM. Для файлов PRL CLI выбирает наименьший сегмент памяти в который может поместиться файл PRL. Для файлов COM CLI выбирает первый доступный абсолютный сегмент памяти для загрузки файла. Примечание: в системе с коммутацией банков может существовать несколько абсолютных сегментов памяти.
2. Если никакой сегмент памяти не доступен, загрузка программы CLI завершается, и система возвращает сообщение об ошибке. В противном случае, CLI загружает программу в свой выбранный сегмент памяти, начиная с BASE+100H (BASE = начальный адрес сегмента в памяти). Если командный файл имеет расширение PRL, и выбранный сегмент памяти не является абсолютным, CLI в это время выполняет операцию перемещения (См. [Раздел 1.6](#_Транзитные_программы)).

Операция загрузки может быть прервана, если происходит ошибка чтения, или в случае файлов COM, если выбранный сегмент памяти не достаточно большой для размещения файла.

1. После загрузки программы, CLI инициализирует память сегмента базовой страницы в сегменте, начинающуюся в BASE+000H. Инициализация базовой страницы более подробно описана в [Разделе 2.3](#_Инициализации_базовой_страницы).
2. Как только базовая страница инициализирована, CLI устанавливает дескриптор процесса для загруженной программы и присваивает процессу имя командного файла. CLI также присваивает, полученные в сообщении командной строки, значения полям дескриптора процесса текущего диска, номера пользователя, устройства печати и номера консоли и дает процессу 20-байтовый стек. Затем инициирует транзитную программу с помощью XDOS вызова Create Process. После этого CLI готов к чтению следующей командной строки и повторению цикла.

## Транзитные программы

В MP/M II транзитной программой является та, которую загружает и инициирует CLI. Как подразумевает название транзитная, программа не является системным резидентным объектом. Система должна загружать ее в доступный сегмент памяти при каждом ее выполнении.

MP/M II может выполнять два типа транзитных программ. Первый тип, программы в абсолютных адресах, должны работать в абсолютном сегменте памяти. Абсолютным сегментом памяти является тот, у которого начальный адрес равен нулю (BASE = 0000H). Командные файлы абсолютных транзитных программ определяются полем расширения файла "COM". Файл COM содержит абсолютный образ памяти в файле, начинающийся с 100H. Поэтому CLI загружает файл COM в память, начиная с адреса 100H. Файлы COM MP/M II совпадают с аналогичными в CP/M.

Второй тип транзитной программы, странично-перемещаемые программы (PRL), могут работать в перемещаемых или абсолютных сегментах памяти. Командные файлы PRL имеют поле расширения "PRL". Файл PRL содержит три области: одностраничный заголовок, область кода и битовый массив перемещения. У заголовка есть поле, содержащее длину области кода программы и поле, определяющее минимальное количество дополнительного пространства данных, требуемое программе. CLI использует эту информацию для выбора сегмента памяти для программы. Область кода содержит код и инициализированные данные программы. CLI загружает область кода в память, начинающуюся в BASE+100H, где BASE - адрес начала сегмента памяти.

Битовый массив состоит из битовой строки, в которой каждый бит соответствует байту в области кода. Первый бит соответствует первому байту, который загружается по адресу BASE+100H. Поскольку битовый массив следует непосредственно за областью кода в файле PRL, смещение битового массива равняется значению длины программы, сохраненному в заголовке PRL. Каждый бит, равный 1, идентифицирует старший байт поля адреса, которое требует перемещения. Во время операции загрузки программы CLI добавляет старшим байтам или смешение адреса BASE страницы к байтам, определенным для перемещения битовым массивом. Это динамично перемещает программу и позволяет ей работать в перемещаемых сегментах памяти. PRL, загруженная в абсолютные сегменты памяти, не требует никакого перемещения. Примечание: преобразовать файл COM в файл PRL не возможно. Однако обратная операция возможна и выполняется с помощью утилиты PRLCOM (см. [Раздел 6.3](#_PRLCOM)).

В рамках операции загрузки программы, CLI инициализирует базовую страницу сегмента памяти следующим образом:

BASE+000H : Прямой XIOS и переход к программе завершения

BASE+005H : Переход к функциям BDOS и XDOS

BASE+050H : Диск командного файла

BASE+051H : Адрес пароля 1-го файла в хвосте команды

BASE+053H : Длина пароля 1-го файла в хвосте команды

BASE+054H : Адрес пароля 2-го файла в хвосте команды

BASE+056H : Длина пароля 2-го файла в хвосте команды

BASE+05CH : Разобранный FCB для 1-го файла в хвосте команды

BASE+06CH : Разобранный FCB для 2-го файла в хвосте команды

BASE+080H : Хвост команды

Во время выполнения транзитная программа делает системные вызовы BDOS или XDOS, вызывая BASE+5. Прямые вызовы XIOS выполняются переходом к адресу BASE+000H. Примечание: Прямые вызовы XIOS ограничены вводом-выводом на консоль и устройства печати. Вся память в сегменте расположенная ниже адреса, содержащегося в BASE+6, доступна транзитной программе. Таким образом транзитные программы могут использовать этот адрес для определения размера памяти. Остальная информация, помещенная в базовую страницу является данными разобранными из командной строки. Эта информация предоставлена для удобства программиста и подробно рассматривается в [Разделе 2](#_Интерфейс_BDOS).

Когда CLI инициирует транзитную программу, он присваивает 20-байтовую область стека процессу. Этот стек инициализируется таким образом, что если программа возвращается к системе, ее выполнение завершается. Процесс может также завершить выполнение с помощью перехода по адресу BASE+000H, вызову BDOS *сброс системы* или вызову XDOS *завершение процесса*.

## Резидентные системные процессы

Резидентные системные процессы (RSP) являются необязательными процессами, которые могут быть включены в MP/M II во время генерации системы. Существует два типа RSP: стандартные и поддерживающие коммутацию банков памяти. Стандартный RSP является странично-перемещаемым файлом, который имеет расширение файла RSP. При интеграции в MP/M II, стандартный RSP находится в области общей памяти. RSP поддерживающий коммутацию банков состоит из двух странично-перемещаемых файлов, оба имеющих одно и то же имя файла, но различные поля расширения файла RSP и BRS соответственно. Когда поддерживающий коммутацию банков RSP включен в MP/M II, файл RSP загружается в общую память, тогда как файл BRS загружается в сегмент памяти 0 в банке 0. Поскольку все дескрипторы процессов и очереди должны находиться в общей памяти, общий модуль RSP с поддержкой коммутации банков содержит свой дескриптор процесса и любые дополнительные дескрипторы процессов и очереди.

Поле сегмента памяти дескриптора процесса RSP определяет, является ли RSP стандартным или с поддержкой коммутации банков. Стандартные RSP устанавливают поле сегмента памяти равным FFH. RSP с поддержкой коммутации банков обнуляют поле. При выборе RSP в процессе генерации системы GENSYS проверяет это поле и если оно установлено в 0, включает файл BRS в сегмент памяти 0.

RSP загружаются в память как часть операции MPMLDR и инициируются после инициализации системного вызова XIOS и до инициализации TMP. После инициализации, RSP выполняется как любой другой процесс в системе, конкурирующий за ЦП и другие системные ресурсы на приоритетной основе.

Если RSP должен быть вызван как встроенная команда из консольной командной строки, он должен выполнить следующие шаги:

1. Создать очередь с длиной сообщения достаточно большой, чтобы принять хвост команды. Имя очереди является именем команды RSP. Поскольку CLI преобразует командные строки в верхний регистр, имена очереди RSP должны быть в верхнем регистре. Если CLI должен присвоить консоль RSP, имя дескриптора процесса RSP должно быть таким же, как имя его очереди.
2. Выполнить безусловный вызов *Read Queue* (Чтение очереди). Это приостанавливает RSP на его очереди в список постановки (Enqueue) в очередь, пока CLI не записывает это сообщение командной строки.
3. Выполнить свою задачу, делая вызовы функций BDOS и XDOS, с помощью сообщения командной строки, содержащей текущий диск, пользователя, устройство печати и номер системной консоли, полученных путем чтения из очереди. Примечание: RSP не делает системные вызовы, вызывая расположение 5. Система инициализирует первые два байта стандартного RSP и первые два байта общего модуля расширенного RSP для содержания адреса системной точки входа. Система устанавливает первые два байта банка ноль модуля RSP с поддержкой коммутации банков равными начальному адресу соответствующего общего модуля. RSP должны использовать эти адреса для выполнения системных вызовов.
4. После выполнения своей задачи RSP должен выполнить вызов XDOS *Detach Console* (Отсоединение консоли) и вызов XDOS *Detach List* (Отсоединение устройства печати), если ему назначена консоль с помощью CLI. Затем он возвращается к шагу 2 и ждет другую командную строку.

Другой специальный тип RSP - *резидентная системная процедура*. Резидентная системная процедура обеспечивает способ последовательного использования блока кода в качестве системного ресурса. Резидентная системная процедура устанавливается RSP. Процесс создает очередь с именем резидентной системной процедуры и отправляет ему единственное двухбайтовое сообщение, содержащее адрес процедуры, к которой получат доступ последовательно. Как только она выполнена, RSP завершает себя.

К резидентной системной процедуре получают доступ, открывая очередь и читая двухбайтовое сообщение для получения фактический адреса процедуры в памяти. Поскольку только одно сообщение находится в очереди, только один процесс может получить доступ к процедуре. Когда процесс освобождает процедуру, он записывает сообщение, содержащее адрес процедуры назад в очередь. Это позволяет следующему ожидающему процессу использовать резидентную системную процедуру.

## Соглашения о вызовах BDOS и XDOS

Системные функции BDOS и XDOS MP/M II могут быть доступны как для транзитных программ так и RSP. Транзитные программы делают системные вызовы через основную точку входа в расположении BASE+005H, где BASE равняется начальному адресу сегмента памяти транзитной программы. Стандартные RSP получают адрес системной точки входа из первых двух байтов программы. Для RSP с коммутацией банков первые два байта общего модуля содержат адрес системной точки входа. Первые два байта модуля нулевого банка содержат адрес общего модуля.

MP/M II использует стандартный протокол для вызовов системных функций. Этот же протокол используется CP/M. В общем случае, при создании системного вызова, регистр C содержит номер функции и регистровая пара DE содержит адрес информации. Функции возвращают однобайтовые значения в регистре A и двухбайтовые значения в регистровой паре HL. Любой системный вызов, сделанный с номером вне диапазона или неподдерживаемой функции возвращает 0FFFFH в регистровой паре HL. Примечание: CP/M возвращает HL равным 0 при недопустимом вызове функции. Для совместимости регистр A равняется L, и регистр B равняется H при возврате во всех случаях. Соглашение о распределении регистров в MP/M II согласовано с используемым в системном языке программирования PL/M Intel.

При вводе транзитной программы система устанавливает указатель стека в 10-уровневый стек с адресом, содержащим BASE+001H, продвинутым в стек. Таким образом возврат в систему эквивалентен переходу к BASE+000H. Как правило, этот стек достаточно большой, чтобы обработать системные вызовы. Однако, большинство транзитных программ устанавливает свой собственный стек и возвращаются в систему с помощью перехода к расположению BASE+000H. Из-за способа интеграции RSP в систему, они должны установить и инициализировать свой собственный стек.

Программист должен знать, что вызовы функций BDOS и XDOS не восстанавливают регистрам их входные значения при возврате в программу вызова. Ответственность за сохранение и восстановление любых важных значений в регистрах лежит на вызывающем процессе.

В качестве примера, следующая транзитная программа показывает использование системных вызовов. Эта программа постоянно читает символы, пока не встретится звездочка, и в этот момент она завершает выполнение, возвращаясь в систему.

ORG 0000H

BASE EQU $ ; Начало сегмента памяти

BDOS EQU BASE+0005H ; Системная точка входа MP/M II

CONIN EQU 1 ; Функция ввода с консоли

;

ORG 100H ; Начало области транзитных программ

NEXTC MVI C,CONIN ; Номер функции чтения след. символа

CALL BDOS ; Возврат символа в регистре A

CPI '\*' ; Завершение процесса

JNZ NEXTC ; Цикл, если нет

RET ; Завершить программу

END

# Интерфейс BDOS

## Интерфейс BDOS устройств ввода-вывода консоли и печати

Основная цель дизайна MP/M II состояла обеспечении совместимости с CP/M. Таким образом, с точки зрения прикладных программ существуют только незначительные различия между CP/M и MP/M II в отношении функций ввода-вывода BDOS на консоли и устройства печати. Эти различия описаны в [*Разделе 2.4, Вызовы функции BDOS*](#_Вызовы_функций_BDOS_3).

У каждой программы, выполняющейся в MP/M II, есть структура данных, называемая *дескриптором процесса*, которая определяет характеристики процесса. Один байт дескриптора процесса идентифицирует устройства ввода-вывода консоли и печати, в настоящее время используемые процессом. Старшие 4 бита этого байта, помеченное поле CONSOLE/LIST, содержит номер устройства печати. Младшие 4 бита содержат номер устройства консоли. Функции BDOS ввода-вывода на устройства консоли и печати получают номер соответствующего устройства из поля CONSOLE/LIST дескриптора процесса для вызова подпрограммы XIOS консоли или устройства печати.

Для получения доступа к устройству процесс должен быть присоединен к консоли или устройству печати. Это относится к вызовам функций BDOS и прямым вызовам XIOS. Система MP/M II перехватывает все вызовы функции BDOS и прямые вызовы XIOS для устройств консоли и печати, чтобы определить, присоединено ли указанное устройство к процессу вызова. Вызов функции разрешается, только если устройство в настоящее время свободно, или присоединено к процессу вызова. Если устройство присоединено к некоторому другому процессу, MP/M II выполняет XDOS вызов *Attach* для прикрепления заданного устройства. Затем процесс вызова блокируется, приостанавливая выполнение, пока устройство не будет отсоединено от процесса, владеющего устройством, и вызывается процесс с самым высоким приоритетом, требующим устройство. Присоединение конкретного устройства к процессу может быть выполнено явно, сделав вызов XDOS *Attach Console* или *Attach List*, или неявно делая вызовы функций BDOS и прямые вызовы XIOS, которые вынуждают присоединить устройства.

MP/M II поддерживает таблицы процессов, владеющих в настоящее время устройствами консоли и печати. Эти таблицы содержат адреса дескрипторов процессов. Таким образом, один процесс может владеть несколькими устройствами консоли или печати при наличии адреса его дескриптора процесса в нескольких записях таблицы. Несколько устройства могут быть присоединены, неоднократно используя вызов функции XDOS *Set Console* или *Set List*, сопровождаемого впоследствии вызовом XDOS *Attach* (присоединить). Позже, когда должен быть выполнен фактический ввод-вывод, определенное устройство консоли или печати должны быть установлены в дескрипторе процесса, делая конкретный вызов XDOS функции *Set Console* или *Set List*.

Вся консоли и устройства печати отсоединяются от процесса при его завершении, позволяя процессам ожидающим устройства возобновить выполнение.

При выполнении функций BDOS консольного ввода-вывода существует несколько *управляющих символов ASCII*, которые заставляют MP/M II предпринять конкретные действия. Символ ^C может прервать процесс, владеющий консолью. Символ ^D вынуждает процесс, владеющий консолью отсоединиться от консоли, позволяя другому ожидающему процессу получить доступ к консоли, и затем присоединяет консоль снова перед продолжением. Символы ^S и ^Q используются, чтобы остановить и перезапустить вывод на консоль. Символ ^S заставит консольный вывод быть приостановленным. В этой точке ^Q может быть введен, чтобы возобновить консольный вывод, или может быть введен ^C, чтобы прервать процесс, владеющий консолью. Ввод любого другого символа, когда вывод приостановлен, заставит MP/M II отправлять в консоль символ ASCII (^G) звуковой сигнализации.

## Файловая система BDOS

Базовая дисковая операционная система поддерживает файловую систему с именами на одном - шестнадцати логических дисках. Каждый логический диск разделен на две области: область каталога и область данных. *Область каталога* определяет файлы, которые существуют на диске, и идентифицирует пространство области данных, которое принадлежит каждому файлу. *Область данных* содержит данные файла, определенные каталогом. Область каталога подразделена на шестнадцать логически независимых каталогов, каждый из которых обозначен кодом пользователя от 0 до 15. В целом только файлы, принадлежащие текущему номеру пользователя "видимы" в каталоге. Например, утилита MP/M II DIR выводит на экран только файлы, принадлежащие текущему номеру пользователя.

Файловая система BDOS автоматически выделяет пространство для каталога и области данных, когда файл создается или расширяется и возвращает ранее выделенное место свободному пространству при удалении файла. Если никакое пространство для каталога или пространство данных не доступны для требуемой операции, BDOS возвращает ошибку процессу вызова. Эти действия являются прозрачными для вызывающего процесса. В результате пользователю не нужно иметь дело с организацией каталогов и дисков при использовании функций файловой системы.

*Поле имени* файла из восьми символов и *поле расширения* из трех символов идентифицируют каждый файл в каталоге. Также файлу может быть назначен *пароль* из восьми символов для защиты от несанкционированного доступа. Все функции BDOS, которые предполагают файловые операции, определяют для требуемого файла поля имени и расширения файла. Несколько файлов могут быть определены *неоднозначной ссылкой*. Неоднозначная ссылка использует один или больше символов "?" в поле имени или расширения, чтобы указать, что любые символы соответствуют этой позиции. Таким образом, спецификация имени и расширения из всех "?" (эквивалентна спецификации файла командной строки "\*.\*") соответствует всем файлам в каталоге, которые принадлежат текущему номеру пользователя.

Файловая система BDOS поддерживает четыре категории функций: функции доступа к файлу, функции каталога, функции связанные с диском и прочие функции. Категория *доступа к файлу* включает функции создать новый файл (*make*), открыть существующий файл (*open*) и закрыть существующий файл (*close*). Функции создания и открытия активируют файл для последующего доступа с помощью функций записи (*write*) и чтения (*read*). После открытия файла, последующие функции BDOS могут читать или писать записи файла последовательно или в произвольном порядке. Команды BDOS чтения и записи передают данные в 128-байтовых логических единицах, которая является основным размером записи в файловой системе. Функция *close* выполняет два шага, чтобы завершить доступ к файлу. Во-первых, она указывает файловой системе, что процесс вызова завершил доступ к файлу. Файл после этого становится доступным другим процессам. Кроме того, функция делает все необходимые обновления в каталоге окончательно записывая текущее состояние файла.

*Функции каталога* BDOS воздействуют на существующие записи файла в каталоге диска. Эта категория включает функции, поиска одного или нескольких файлов, удаление одного или нескольких файлов, переименование файла, установка атрибутов файла, присвоение файлу пароля и вычисление размера файла. Функции BDOS поиска и удаления, единственные функции, которые позволяют неоднозначные файловые ссылки. Все остальные функции имеющие отношение к каталогу и файлам, требуют конкретной ссылки на файл. Файловая система BDOS не позволяет процессу удалять, переименовывать или устанавливать атрибуты файла открытого в настоящее время другим процессом.

*Связанные с диском* функции BDOS включают: выбор диска по умолчанию, вычисление свободного места на диске, опрос состояния и присвоение диску метки каталога. *Метка каталога* диска определяет, нужно ли использовать пароли файла, и тип меток даты и времени, присваиваемые файлам на диске. Также в эту категорию включены функции сброса определенных дисков и управления возможностью сброса определенных дисков другими процессами. Когда диск сброшен, следующие операции с диском повторно активируют, регистрируя его. Функция операции входа в систему должна инициализировать диск для операций с каталогом и файлами. В MP/M II, успешная операция сброса диска должна быть выполнена для дисков, которые поддерживают съемные носители, прежде чем сменять диски.

*Прочие функции* включают: установку текущего адрес DMA, доступ и обновление текущего номера пользователя, цепочку к новой программе и сброс внутреннего буфера блокирования/разблокирования. Также включены функции установки многосекторного счетчика BDOS и режима ошибок BDOS. *Многосекторный счетчик* BDOS определяет число 128-байтовых записей, которые будут обработаны функциями BDOS чтения, записи, блокировки записей и снятия блокировки записей. Он может изменяться от одной до шестнадцати 128-байтовых записей. Значение по умолчанию – одна запись. Режим ошибок BDOS определяет, перехватывает ли файловая система ошибки BDOS, или возвращает все ошибки процессу вызова.

В следующем списке перечислены операции, выполняемые в файловой системе BDOS:

Сброс дисковой системы;

Выбор диска;

Создание файла;

Открытие файл;

Закрытие файл;

Поиск в каталоге;

Удаление файла;

Переименование файла;

Произвольное или последовательное чтение;

Произвольная или последовательная запись;

Опрос выбранных дисков;

Установка адреса DMA;

Индикаторы Установки/Сброса файла;

Сброс диска;

Получить Доступ/Освободить диск;

Случайная запись с заполнением нулями;

Блокирование и разблокирование записи;

Установка многосекторного счетчика;

Установка режима ошибок BDOS;

Получить свободное место на диске;

Цепочка к программе;

Сброс буферов;

Установка метки каталога;

Возврат метки каталога;

Чтение и запись файла XFCB;

Установить/Получить дату и время;

Установить значение пароля по умолчанию;

Возврат серийного номера BDOS.

Следующие разделы содержат информацию о важных темах, связанных с файловой системой BDOS. Читатель должен быть знаком с содержанием этих разделов прежде, чем пытаться использовать функции BDOS, описанные индивидуально в [Разделе 2.4](#_Вызовы_функций_BDOS_1).

### Соглашения о присвоении имен файлам

В MP/M II имена файлов состоят из четырех частей: кода выбора диска (d), поля имени файла, поля расширения файла и поля пароля файла. Общий формат *спецификации файла* командной строки показан ниже:

{d:}filename{.typ} {;password}

Поле кода выбора диска определяет диск на котором расположен файл. Поля имени и расширения файла идентифицируют файл. Поле пароля определяет пароль, если файл защищен паролем.

Поля диска, расширения и пароля необязательные и разделители ":", "." и ";" требуются только при определении связанного с ними поля. Коду выбора дисковода может быть присвоено значение от "A" до "P", где фактические коды диска, поддерживаемые в данной системе, определяются реализацией XIOS. Если код диска не определен, текущий диск по умолчанию обозначен. Поле имени файла может содержать от одного до восьми символов исключая разделители, поле расширения файла от одного до трех символов исключая разделители, и поле пароля от одного до восемь символов исключая разделители. Все буквенные символы должны быть в верхнем регистре. Кроме того, все три поля дополняются пробелами при необходимости. Исключение необязательных полей расширения или пароля подразумевает заполнение всего поля спецификации пробелами.

Функция МП/М II разбора имени файла распознает определенные символы ASCII в качестве допустимых разделителей при анализе файла из командной строки. Допустимые символы показаны в *Таблице 2-1*.

Таблица ‑ Допустимые разделители имени файла

|  |  |
| --- | --- |
| **ASCII** | **Шестнадцатеричный эквивалент** |
| :  .  ;  =  ,  /  [  ]  <  > | 3A  2E  3B  3D  2C  2F  5B  5D  3C  3E |

Функция разбора имени файла также исключает все управляющие символы из полей файла и переводит все строчные буквы в верхний регистр.

Следует избегать использование символов "(" и")" в полях имени и расширения файла, так как они часто используются в качестве разделителя. Символы "\*" и "?" не должны использоваться в полях имени файла и расширения, если они не используются в качестве неоднозначной ссылки. Если функция разбора имени файла встречает "\*" в поле имени или расширения, она дополняет остаток поля символами "?". Например, имя файла "X\*.\*" преобразуется в "X???????.???". Функции BDOS поиска и удаления обрабатывают символ "?" в полях имени файла и расширения следующим образом: "?" в любой позиции соответствует любому соответствующему символу в поле любой записи каталога, принадлежащего текущему коду пользователя. Таким образом, операция поиска "X???????.???" находит все текущие пользовательские файлы в каталоге, начинающемся с "X". Большая часть других функций BDOS связанных с обработкой файлов расценивает присутствие символа "?" в поле имени или расширения файла как ошибку.

Не обязательно следовать соглашениям MP/M II о присвоении имен файлам при создании или переименовании файла с помощью функций BDOS. Однако соглашения должны использоваться, если к файлу нужно получить доступ из командной строки. Например, CLI не может определить местоположение командного файла в каталоге, если его поле имени или расширения файла содержит строчную букву.

Как правило, поле расширения файла обозначает категорию определенного файла, в то время как имя файла различает отдельные файлы в каждой категории. Несмотря на то, что они обычно произвольны, расширения файлов упомянутые ниже относятся некоторым общепринятым категориям.

|  |  |  |  |
| --- | --- | --- | --- |
| ASM | Исходный файл на ассемблере | INT | Промежуточный файл |
| HEX | Шестнадцатеричный машинный код | REL | Перемещаемый модуль |
| PLI | Исходный файл на языке PL/I | BAS | Исходный файл Basic |
| TEX | Источник форматирования TEX | COM | Командный файл |
| PRL | Странично перемещаемый | SYS | Системный файл |
| SPR | Системный странично-перемещаемый | BAK | Резервная копия ED |
| BRS | Файл RSP с поддержкой коммутации | SYM | Символьный файл SID |
| RSP | Резидентный системный процесс | $$$ | Временный файл |
| PRN | Файл, предназначенный для печати | DAT | Файл с данными |
|  |  |  |  |

### Дисковод и структура файлов

Файловая система BDOS может поддерживать от одного до шестнадцати логических дисков. Максимальный размер файла поддерживаемый на диске составляет 32 мегабайта. Максимальную емкость диска определяет размер блока данных, указанный для диска в XIOS. Размер блока данных является основной единицей, в которой BDOS выделяет место на диске для файлов. *Таблица 2-2* показывает взаимосвязь между размером блока данных и емкостью диска.

Таблица ‑ Емкость логического диска

|  |  |
| --- | --- |
| **Размер блока данных** | **Максимальная  емкость диска** |
| 1K  2K  4K  8K  16K | 256 килобайт  64 мегабайта  128 мегабайт  256 мегабайт  512 мегабайта |

Логические диски разделены на две области: область каталога и область данных. Область каталога содержит от одного до шестнадцати блоков, расположенных в начале диска. Фактический номер определен в XIOS. Эта область содержит записи, которые определяют, какие файлы существуют на диске. Записи каталога, соответствующие определенному файлу, определяют, какие блоки данных в области данных диска принадлежат файлу. Эти блоки данных содержат записи файла. Область каталога логически разделена на шестнадцать независимых каталогов, определенных как пользователь от 0 до 15. Каждый независимый каталог совместно использует фактическую область каталога на диске. Однако записи файла в каталоге не могут существовать больше чем под одним номером пользователя. В общем, только файлы принадлежащие текущему номеру пользователя видимы в каталоге.

Каждый файл на диске состоит из набора до 242 144 128-байтных записей. Каждая запись в файле определяется ее позицией в файле. Эта позиция называется *номером произвольной записи*. Если файл создается последовательно, первая запись имеет позицию ноль, а последняя запись имеет позицию на единицу меньше, чем число записей в файле. Такой файл может быть считан последовательно в порядке следования позиций, начиная с нулевой записи, или в произвольном порядке следования позиций. С другой стороны, если файл создается в произвольном порядке, записи добавляются в файл с указанием позиции. Файл, создаваемый таким образом, называют "разреженным", если в файле существуют позиции в которые запись не производилась.

BDOS автоматически распределяет блоки данных файлу, чтобы содержать его записи на основе использованных позиций записей. Таким образом разреженный файл, который содержит две записи, одну в нулевой позиции, другую в позиции 242 143, использовал бы только два блока данных в области данных. Разреженные файлы могут быть созданы и доступны только в произвольном порядке, не последовательном. Обратите внимание, что любой блок данных, выделенный файлу, выделяется файлу постоянно, пока файл не будет удален. Не существует другого способа, поддерживаемого BDOS, освобождения блоков данных, принадлежащих файлу.

Исходные файлы в MP/M рассматриваются как последовательность символов ASCII, в которой каждая "строка" исходного файла сопровождается последовательностью перевода строки и возврата каретки (0DH сопровождаемый 0AH). Таким образом единственная 128-байтная запись может содержать несколько строк исходного текста. Конец файла обозначается ASCII символом Ctrl-Z (1AH) или реальным концом файла, возвращенным операцией чтения BDOS. Символы Ctrl-Z, находящиеся в файлах машинного кода, таких как COM или файлы PRL, игнорируются. Условие конец файла, возвращенное BDOS, используется для завершения операции чтения.

### Определение блока управления файлом

Блок управления файлом (FCB) - структура данных, используемая функциями BDOS для доступа к файлу и каталогу. Все эти функции ссылаются на FCB, чтобы определить файл или файлы которые будут использованы в операциях. Также, определенные поля в FCB используются для вызова специальных параметров, связанных с некоторыми функциями. Другие функции используют FCB, чтобы вернуть данные вызывающему процессу. Самое главное, когда процесс открывает файл и впоследствии получает доступ к нему с помощью функций чтения, записи, блокировки и разблокировки записей, файловая система BDOS сохраняет текущее состояние файла и позицию в FCB пользователя. Кроме того, все функции BDOS произвольного ввода-вывода определяет номер произвольной записи в 3-байтном поле в конце FCB.

При осуществлении вызова функции BDOS доступа к файлу или каталогу, регистровая пара DE должна адресовать FCB. Длина области данных FCB зависит от функции BDOS. Для большинства функций требуемая длина составляет 33 байта. Для функций произвольного ввода-вывода и функции вычисления размера файла, длина FCB должна составить 36 байтов. Когда функции BDOS открытия или создания файла определяют файл, который должен быть открыт в неблокируемом режиме, FCB должен иметь длину 35 байтов. Ниже показан формат FCB.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| dr | f1 | f2 | ... | f8 | t1 | t2 | t3 | ex | s1 | s2 | rc | d0 | ... | dn | cr | r1 | r2 | r3 |
| 00 | 01 | 02 | ... | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | ... | 31 | 32 | 33 | 34 | 35 |

где

|  |  |
| --- | --- |
| dr | Код диска (0 - 16).  0 => использовать для файлов диск по умолчанию,  1 => автоматический выбор в качестве диска дисковода A,  2 => автоматический выбор в качестве диска дисковода B,  ...  16 => автоматический выбор в качестве диска дисковода P. |
| f1 ... f8 | Содержат имя файла в верхнем регистре ASCII со старшим битом = 0.  f1' ... f8' обозначают старший бит этих позиций и биты атрибутов файлов. |
| t1, t2, t3 | Содержат расширение файла в верхнем регистре ASCII со старшим  битом = 0.  t1', t2' и t3' обозначают старший бит этих позиций и биты атрибутов файлов.  t1' = 1 => файл только для чтения,  t2' = 1 => системный файл ,  t3’ = 1 => файл был заархивирован. |
| ex | Содержит номер текущего экстента, обычно устанавливается процессом вызова равным 0, но может изменяться от 0 до 31 во время файлового ввода-вывода. |
| cs | Содержит значение контрольной суммы FCB для, открытого FCB. |
| rs | Зарезервирован для внутреннего использования системой, устанавливается в ноль при вызовах OPEN, MAKE, SEARCH |
| rc | Счетчик записей для экстента "ex" принимает значения от 0 до 128 |
| d0 ... dn | Заполняются MP/M, зарезервированы для использования системой |
| cr | Текущая запись в операциях последовательного чтения или записи файла, обычно обнуляется процессом вызова, когда файл открыт или создан |
| r0, r1, r2 | Необязательный номер произвольной записи в диапазоне 0-242 143 (0 - 3FFFFH).  r0, r1, r2 представляют собой 18-битовое значение с младшим байтом r0, средним байтом r1 и старшим байтом r2. |

Примечание: при успешном открытии файла в неблокируемом режиме в байтах r0 и r1 возвращается 2-байтовый идентификатор файла (см. [Раздел 2.2.8](#_Режимы_открытия_файла_2)).

Для функций BDOS каталога процесс вызова должен инициализировать в FCB байты c 0 до 11 перед выдачей вызовов функции. Функции XFCB установки метки каталога и записи файла также требует, чтобы процесс вызова инициализировал байт 12. Функция BDOS переименования файла, требует, чтобы процесс вызова поместил новое имя и расширение файла в байтах 17 - 27.

Вызовы функций BDOS open или make требуют, чтобы процесс вызова инициализировал в FCB байты с 0 до 12 до выполнения вызова функций открытия или создания файла. Обычно, байт 12 обнулен. Кроме того, если файл должен быть обработан с начала, используя функции последовательного чтение или записи, байт 32 (cr) должен быть обнулен. После того, как FCB активирован операцией открытия или создания, FCB не должен изменяться пользователем. Открытые FCB проверяются на соответствие контрольной суммы для защиты целостности файловой системы. В общем случае, если процесс изменяет открытый FCB, следующее вызовы функций чтения, записи или закрытия возвратятся с ошибкой контрольной суммы. Смотрите [Раздел 2.2.9](#_Защита_файлов_1) для получения дополнительной информации о контрольных суммах FCB. Обычно, функции последовательного чтение или записи не требуют инициализации открытого FCB. Однако, функции произвольного ввода-вывода требуют, чтобы процесс установил в байтах 33 - 35 требуемый номер произвольной записи до выполнения вызова функции.

Элементы каталога файлов, сохраняемые в области каталога каждого дисковода, имеют формат аналогичный FCB (исключая байты 32 - 35), за исключением байта 0, который содержит номер пользователя файла. Функции открытия файла и создания файла переносят эти элементы (исключая байт 0) в расположенный в памяти FCB, определенный процессом вызова. Все операции чтения и операции записи файла должны определить FCB, активированный этим способом. Иначе, возвращается ошибка контрольной суммы. BDOS обновляет копию FCB в памяти во время обработки файла для поддержания текущей позиции в файле. Во время операций записи файла BDOS обновляет копию FCB памяти для записи выделенных данных в файл, а при окончании обработки файла, функция закрытия файла окончательно записывает эту информацию на диск. Обратите внимание, что операции записи данных, размещаемых в файл не записываются полностью в каталог, до тех пор, пока вызывающий процесс не выдает вызов закрытия файл. Поэтому, процесс, который создает или изменяет файлы обязательно должен закрывать файлы при завершении любой обработки записи, иначе, данные могут быть потеряны.

Как правило в MP/M II, процесс должен закрыть файлы, как только они больше не нужны, даже если они не были изменены. Файловая система BDOS поддерживает запись в системном списке блокировки (сегмент памяти LCKLSTS.DAT) для каждого файла, открытого каждым процессом в системе. Эта запись не удаляется из системного списка блокировки, пока файл не закрыт, или процесс владеющий записью не завершается. Файловая система BDOS использует эту запись, чтобы препятствовать другим процессам получения доступа к файлу, если файл не был открыт в режиме, который поддерживает совместный доступ. Обычно, процесс должен закрыть файл, прежде чем другие процессы в системе смогут получить доступ к файлу.

Имейте в виду, что в MP/M II место в системном списке блокировки является ограниченным ресурсом. Если процесс пытается открыть файл и места в системном списке блокировки нет, или процесс превышает предел открытых файлов (определяется во время генерации системы), BDOS отклоняет операцию открытия и обычно прерывает процесс вызова.

Старшие биты в полях FCB имени файла (f1',..., f8') и расширения (t1', t2', t3') называют битами атрибутов. Биты атрибутов - 1 битные логические поля, где 1 указывает включен или истина, и 0 указывает выключен или ложь. Биты атрибутов имеют две функции в файловой системе: атрибуты файла и атрибуты интерфейса.

Атрибуты файла (f1',..., f4' и t1', t2', t3') используются для обозначения, что файл имеет определенный атрибут. Эти биты записываются в FCB файла в каталоге. Атрибуты файла могут быть установлены или сброшены только с помощью функции BDOS установки атрибутов файла. Когда функция BDOS создания файла создает файл, она обнуляет все атрибуты файла. Процесс может опросить атрибуты файла в FCB, активированных функцией BDOS открытия файла или в FCB в каталоге, возвращенных функциями BDOS поиск первого и поиск следующего. Примечание: BDOS игнорирует биты атрибута файла в файловой системе, когда она пытается определить местоположение файла в каталоге.

Атрибуты файла (t1', t2', t3') определены файловой системой следующим образом:

t1': Атрибут только для чтения - файловая система предотвращают операции записи в файл с установленным атрибутом только для чтения.

t2': Атрибут системный - этот атрибут, если задан, определяет файл в качестве системного файла MP/M II. Системные файлы обычно не отображается утилитой MP/M II DIR. Кроме того, системные файлы только для чтения пользователя 0 могут быть доступны на основе других номеров пользователя (см. [Раздел 2.2.8](#_Режимы_открытия_файла_1)).

t3': Архивный атрибут - этот атрибут предназначен для написанных пользователем программ архивирования. Когда программа архивирования копирует файл в резервное хранилище, она устанавливает архивный атрибут скопированным файлам. Файловая система автоматически сбрасывает архивный атрибут FCB в каталоге, который был задан командой записи. Программа архивирования может протестировать этот атрибут в каждом FCB файла в каталоге с помощью BDOS функций Search и Searchn. Если все FCB в каталоге имеют установленный архивный атрибут, это указывает, что файл не был изменен с момента предыдущего архивирования. Обратите внимание, что утилита MP/M II PIP поддерживает архивацию файла.

Атрибуты f1' - f8' доступны для определения пользователем.

Атрибутами интерфейса являются f5' - f8'. Эти атрибуты не могут использоваться в качестве атрибутов файла. Атрибуты интерфейса f5' и f6' используются для запроса параметров при вызовах BDOS, требующих адрес FCB в регистровой паре DE. Они используются функциями BDOS открытия, создания, закрытия и удаления файла. *Таблица 2-3* показывает определения атрибутов интерфейса f5' и f6' для этих функций.

Таблица ‑ Атрибуты интерфейса

|  |  |
| --- | --- |
| Функция открытия  Функция создания  Функция закрытия  Функция удаления | f5' = 1 : Открыть в разблокированном режиме  f6' = 1 : Открыть в режиме только для чтения  f5' = 1 : Открыть в разблокированном режиме  f6' = 1 : Назначить пароль файлу  f5' = 1 : Частичное закрытие  f5' = 1 : Удалить только файлы XFCB |

Атрибуты интерфейса подробно описываются для каждой из вышеупомянутых функций в [Разделе 2.4](#_Вызовы_функций_BDOS_2). Атрибуты f5' и f6' всегда сбрасываются, когда управление возвращается к процессу вызова. Атрибуты интерфейса f7' и f8' зарезервированы BDOS для внутреннего пользования файловой системой.

Функции BDOS поиска и удаления допускают неоднозначную ссылку (на несколько файлов). В целом, символ ? в имени файла, расширении или поле экстента соответствует любому значению в соответствующих позициях FCB каталога во время операций поиска в каталоге. Функции BDOS поиска также распознают символ ? в поле кода диска и, если определено, они возвращают все записи каталога на диске независимо от номера пользователя, включая пустые записи. FCB в каталоге, начинающийся с E5H, является пустой записью каталога.

### Соглашения о номере пользователя

Функция пользователя MP/M II делит каждый каталог диска на шестнадцать логических независимых каталогов, определяемых как пользователь 0 - пользователь 15. Физически, все пользовательские каталоги совместно используют область каталога диска. Однако, в большинстве других аспектов они независимы. Например, файлы с тем же именем без конфликтов могут существовать с различными номерами пользователя на одном диске. Однако один файл не может иметь больше одного номера пользователя.

Номер только одного пользователя активен для процесса в одно время, и номер текущего пользователя применяется ко всем дискам в системе. Кроме того формат FCB не содержит поля, которое может быть использовано для переопределения номера текущего пользователя. В результате все операции с файлами и ссылками в каталоге, связанные с номером текущего пользователя. Тем не менее, процесс может получить доступ к файлам с другими номерами пользователей, установив номер пользователя для номера пользователя файла с помощью команды BDOS задать пользователя до создания вызова необходимой функции BDOS для файла. Следует отметить, что этот метод должен использоваться с осторожностью. Если процесс попытается прочитать или записать в файл под номером пользователя, не совпадающий с номером пользователя, который был активен, когда файл был открыт, файловая система BDOS возвратит ошибку контрольной суммы FCB.

Когда CLI инициирует транзитную программу или RSP, номер его пользователя установлен в значение, установленное процессом, выдавшим XDOS *Send CLI Command*. Обычно, процессом отправки является TMP. Тем не менее, процессом отправки может быть другой процесс, например, транзитная программа, которая выполняет вызов BDOS цепочки программы. Транзитная программа может изменить свой номер пользователя, сделав вызов BDOS установить пользователя. Изменение номера пользователя таким образом не влияет на номер пользователя командной строки, выведенный на экран TMP. Таким образом, когда транзитная программа, которая изменила свой номер пользователя завершается, исходный номер пользователя консоли восстанавливается, когда TMP восстанавливает управление.

В MP/M II пользователь 0 обладает особыми свойствами. С некоторыми ограничениями файловая система автоматически открывает файл пользователя 0, если он не присутствует под текущим номером пользователя. Конечно, это действие выполняется только, когда текущий номер пользователя не равен нулю. Кроме того, файл пользователя 0 должен иметь атрибут системный (t2'), чтобы иметь право на эту операцию. Эта процедура позволяет утилитам, которые могут включать в себя оверлеи и любые другие часто используемые файлы, помещенные в области пользователя 0, быть доступными из других областей пользователя. В результате это избавляет от необходимости копирование часто используемых утилит во все области пользователя в каталоге и дает MP/M II пользовательский элемент управления, с помощью которого файлы пользователя 0 непосредственно доступны из других областей пользователей. Обратитесь к [Разделу 2.2.8](#_Режимы_открытия_файла) для получения дополнительной информации по этой теме.

### Метки каталога и XFCB

Файловая система BDOS включает два специальных типа FCB: XFCB и метки каталога. XFCB - "расширенный" FCB, который может дополнительно быть связан с файлом в каталоге, если он существует. Он содержит поле пароля файла, и информацию меток даты и времени. Формат XFCB показан ниже:

**Формат XFCB**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| dr | Имя | | | | Расширение | | | pm | s1 | s2 | rc | Пароль | | | ts1 | | ts2 | | |
| 00 | 01 | 02 | ... | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | ... | 24 | 25 | ... | | 29 | ... |

где

|  |  |
| --- | --- |
| dr | Код диска (0 - 16) |
| Имя | Поле имени файла |
| Расширение | Поле расширения файла |
| pm | Режим пароля  бит 7 – режим чтения  бит 6 – режим записи  бит 5 – режим удаления  \*\* - битовые ссылки справа налево, по отношению к 0 |
| s1, s2, rc | Зарезервированы для использования системой |
| Пароль | Поле пароля размером 8 байт (зашифрованный) |
| ts1 | Поле метки времени создание или доступа размером 4 байта |
| ts2 | Поле метки времени обновления размером 4 байта |

XFCB может быть создан для файла двумя способами: автоматически, как часть функции BDOS Make или явно, функцией BDOS запись XFCB файла. Файловая система BDOS автоматически не создает XFCB для файла, если метка каталога файла не присутствует на диске. Функция BDOS Чтения XFCB файла возвращает XFCB файла, если он существует в каталоге. Обратите внимание на то, что в каталоге, XFCB идентифицирован значением байта диска (байт 0 в FCB) равный 16 + N, где N равняется номеру пользователя.

Метка каталога определяет для диска, необходимы ли пароли для файлов защищенных паролем, должно ли быть выполнено создание меток даты и времени для файлов и должны ли автоматически создаваться XFCB для файлов функцией Make. Формат метки каталога похож на XFCB как показано ниже.

**Формат метки каталога**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| dr | Имя | | | | Расширение | | | d1 | s1 | s2 | rc | Пароль | | | ts1 | | ts2 | | |
| 00 | 01 | 02 | ... | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | ... | 24 | 25 | ... | | 29 | ... |

где

|  |  |
| --- | --- |
| dr | Код диска (0 - 16) |
| Имя | Имя метки каталога |
| Расширение | Расширение метки каталога |
| d1 | Байт данных метки каталога  бит 7 – требует паролей для файлов  бит 6 – создавать метку времени доступа  бит 5 – создавать метку времени обновления  бит 4 – Make создает XFCB  бит 0 – Метки каталога существует  \*\* - битовые ссылки справа налево, по отношению к 0 |
| s1, s2, rc | н/д |
| Пароль | Поле пароля размером 8 байт (зашифрованный) |
| ts1 | Поле метки времени создание размером 4 байта |
| ts2 | Поле метки времени обновления размером 4 байта |

Только одна метка каталога может существовать в каталоге диска. Поля имени и расширения метки каталога не используются для поиска в каталоге метки каталога. Они могут использоваться, для идентификации дискеты или диска. Метка каталога может быть создана, или ее поля могут быть обновлены функцией BDOS *Set Directory Label* (Установка метки каталога). Также эта функция может присвоить метке каталога пароль. Пароль метки каталога, если присвоен, не может быть обойден, когда защитой паролем файлов управляет метка каталога. Таким образом, пароль доступа к метке каталога обеспечивает своего рода состояние супер пользователя для этого диска.

Примечание: Файловая система BDOS не имеет функции для непосредственного чтения метки каталога FCB. Однако байт данных метки каталога может быть считан непосредственно с помощью функции BDOS *Return Directory Label* (вернуть метку каталога). Кроме того, функция BDOS *поиск* (‘?' в FCB диска), может использоваться для поиска метки каталога на диске по умолчанию. В каталоге метка каталога идентифицируется по значению байта диска (байт 0 в FCB) равного 32 (20H).

### Пароли файла

Пароли файлам могут быть присвоены двумя способами: функцией *Make File* (Создание файла), если метка каталога определяет автоматическое создание XFCB или функцией *Write File XFCB* (Запись XFCB файла). Пароль файла может также быть изменен функцией запись XFCB файла, если исходный пароль предоставлен. Однако пароль файла не может быть изменен без исходного пароля, даже если отключена защита диска паролем метки каталога.

Защита паролем предоставляется в одном из трех режимов. *Таблица 2-4* показывает разницу в разрешенных уровнях доступа для функций BDOS, когда пароль не введен.

Таблица ‑ Режимы защиты паролем

|  |  |
| --- | --- |
| **Режим пароля** | **Разрешенный уровень доступа, если пароль не введен** |
| 1. Чтение 2. Запись 3. Удаление | Файл не может быть прочитан  Файл можно читать, но нельзя изменять.  Файл может быть изменен, но не может быть удален. |

Если файл защищен паролем в режиме чтения, пароль должен быть предоставлен, чтобы открыть файл. Файл, защищенный в режиме записи, не может быть записан в без пароля. Файл, защищенный в режиме удаления, позволяет доступ для чтения и доступ для записи, но пользователь должен определить пароль, чтобы удалить файл, переименовать файл или изменить атрибуты файла. Таким образом защита паролем в режиме 1 подразумевает режимы защиты 2 и 3 и режим защиты 2 подразумевают режим защиты 3. Все три режима требуют, чтобы пользователь определил пароль, чтобы удалить файл, переименовать файл или изменить атрибуты файла.

Если задан правильный пароль, или если защита паролем отключена, или если защита паролем отключена меткой каталога, то доступ к функциям BDOS такой же, как для файла, который не защищен паролем. Кроме того, функции поиск первого и поиск следующего не зависят от паролей файлов.

*Таблица 2-5* перечисляет функции BDOS проверяющие пароль.

Таблица ‑ Функции BDOS, которые проверяют пароль

|  |
| --- |
| 15. Открытие файла  19. Удаление файла  23. Переименование файла  30. Установка атрибутов файла  100. Установка метки каталога  103. Запись XFCB файла |

Пароли файла имеют длину восемь байтов. Они сохраняются в XFCB и метке каталога в зашифрованном виде. Чтобы сделать вызов функции BDOS для файла, который требует пароль, процесс должен поместить пароль в первые восемь байтов текущего DMA или определить его с помощью функции BDOS установить пароль по умолчанию, до создания вызова функции. Примечание: BDOS поддерживает присвоенный пароль по умолчанию в системной консоли и сохраняет его до завершения процесса.

### Метки времени и даты файла

Файловая система BDOS может записать, когда файл был создан или в последний раз получен к нему доступ, и/или он был в последний раз обновлен. Она записывает метку создания только, если XFCB автоматически создается с помощью функции *создания файла*. Если XFCB создается функцией *создания XFCB файла*, метка создается обнуленной. Функция *закрытия файла* записывает обновленную метку, если сделана операция записи в открытый файл. Функция *открытия файла* записывает метку доступа, если файл успешно открыт. Метка даты создания перезаписывается, когда записывается метка доступа, потому что только два поля даты и времени находятся в XFCB и метки доступа и времени создания совместно используют одно поле.

*Метка каталога диска* определяет тип меток даты и времени, поддерживаемых для файлов на диске. Если у диска нет метки каталога, или если он только для чтения, или если метка каталога диска не определяет метки даты и времени, то запись меток даты и времени для файлов не выполняется. Кроме того, файл должен иметь XFCB, чтобы иметь право на запись меток даты и времени. Для самой метки каталога метки времени записывают, когда она была создана и обновлялась. Запись меток доступа для меток каталога не поддерживается.

Процесс может непосредственно получить доступ к меткам даты и времени для файла, используя функцию *чтения XFCB файла*. Механизм непосредственного обновления поля даты и времени XFCB отсутствует.

Файловая система BDOS использует внутреннюю дату и время MP/M, когда она записывает метки даты и времени. В системах MP/M II, которые не поддерживают часы, штампы даты и времени, записывают последнее инициализированное значение для системной даты и время. Для установки системной даты и времени может использоваться MP/M II утилита TOD.

### Режимы открытия файла

Файловая система BDOS обеспечивает три различных режима открытия файлов. Они определены следующим образом:

**Режим блокировки:**

Процесс может открыть файл в режиме блокировки, только если файл в настоящее время не открыт другим процессом. После открытия в режиме блокировки, никакой другой процесс не может открыть файл, до тех пор, пока он не закрыт. Таким образом, если процесс успешно открывает файл в режиме блокировки, этот процесс в действительности владеет файлом, пока файл не будет закрыт, или процесс не завершится. Файлы, открытые в режиме блокировки, поддерживают операции чтения и записи, за исключением файлов только для чтения (установлен атрибут t1'), или файлов защищенных паролем в режиме записи, и пароль не указан при вызове BDOS *открытие файла*. В обоих этих случаях разрешены только операции чтения файла. Примечание: Режим блокировки является режимом по умолчанию для открытия файлов в MP/M II.

**Неблокируемый режим:**

Процесс может открыть файл в неблокируемом режиме, если файл в настоящее время не открыт, или если файл был открыт другим процессом в неблокируемом режиме. Этот режим позволяет нескольким процессам открывать тот же файл. Файлы, открытые в неблокируемом режиме, поддерживают операции чтения и записи, за исключением файлов только для чтения (установлен атрибут t1'), или файлов защищенных паролем в режиме записи, и пароль не указан при вызове BDOS *открытие файла*. Однако, когда файл, открытый в неблокируемом режиме, расширяется операцией записи, BDOS выделяет место файлу в модулях блока данных, не в 128-байтовых единицах записей, как в обычных случаях. Функции BDOS блокирования и разблокирования записей поддерживаются только для файлов открытых в неблокируемом режиме.

При открытии файла в неблокируемом режиме процесс должен зарезервировать 35 байт в FCB, потому что функция открытия файла возвращает, 2-байтное значение, называемое *Идентификатор файла*, в байтах FCB r0 и r1. Идентификатор файла является обязательным параметром для команд BDOS блокировки и разблокировки записи.

**Режим только для чтения**

Процесс может открыть файл в режиме только для чтения, если файл в настоящее время не открыт другим процессом, или файл был открыт другим процессом в режиме только для чтения. Этот режим позволяет нескольким процессам открывать тот же файл для доступа в режиме только для чтения.

Для файлов открытых в режиме блокировки или только для чтения функция открытия выполняет следующие шаги. Если текущий пользователь ненулевой, и файл, который будет открыт, не существует в текущей области пользователя, функция открытия ищет файл в области пользователя 0. Если файл существует в области пользователя 0 и у файла установлен системный атрибут (t2'), файл открывается из области пользователя 0. После этого режим открытия автоматически устанавливается только для чтения. Для получения дополнительной информации смотрите [Раздел 2.2.4](#_Соглашения_о_номере).

Также функция открытия выполняет следующие действие для файлов, открытых в режиме блокировки, когда текущий номер пользователя 0. Если файл существует в области пользователя 0 и имеет установленные атрибуты системный (t2') и только для чтения (t1'), режим открытия автоматически устанавливается только для чтения. Таким образом атрибут только для чтения определяет, может ли системный файл пользователя 0 быть одновременно открыт процессом пользователя 0 и процессами других пользователей, когда каждый процесс открывает файл в режиме блокировки по умолчанию. Если атрибут только для чтения установлен, все процессы открывают файл в режиме только для чтения и одновременный доступ к файлу разрешен. Однако, если атрибут только для чтения сброшен, процесс пользователя 0 открывает файл в режиме блокировки. Если он успешно открывает файл, никакой другой процесс не может открыть его. Если у другого процесса есть открытый файл, операция его открытия отклоняется.

*Таблица 2-6* показывает определение атрибутов интерфейса f5' и f6' FCB для функции BDOS открыть файл.

Таблица ‑ Атрибуты интерфейса FCB f5' и f6' функции открытия файла

|  |  |
| --- | --- |
| f5' = 0, f6' = 0 | - открытие в режиме блокировки (режим по умолчанию) |
| f5' = 1, f6' = 0 | - открытие в неблокируемом режиме |
| f5' = 0 или 1, f6' = 1 | - открытие в режиме только для чтения |

Атрибут интерфейса f5' определяет режим открытия для функции BDOS создание (*Make*) файла. *Таблица 2-7* показывает определения атрибута интерфейса FCB f5' для функции создания файла.

Таблица ‑ Атрибут интерфейса FCB f5' функции создания файла

|  |  |
| --- | --- |
| f5' = 0 | - открытие в режиме блокировки (режим по умолчанию) |
| f5' = 1 | - открытие в неблокируемом режиме |

Примечание: функция создания файла не позволяет открывать файл в режиме только для чтения.

### Защита файлов

В целом меры безопасности, реализованные в файловой системе BDOS, предназначены для предотвращения случайных столкновений между запущенными процессами. В MP/M II не возможно обеспечить общую безопасность, потому что файловая система BDOS сохраняет информацию о размещении открытых файлов в FCB в области памяти пользователя, и MP/M II не поддерживает защиту памяти. В худшем случае программа, которая «падает» в MP/M II, может привести в нерабочее состояние всю систему. Поэтому, MP/M II требует, чтобы все процессы, работающие в системе, были "дружественными". Однако файловая система BDOS предназначена для того, что бы несколько процессов могли использовать одну файловую систему, не мешая друг другу. Она делает это двумя способами:

* Выполняет проверку контрольных сумм открытых FCB.
* Контролирует все открытые файлы и заблокированные записи с помощью системного списка блокировок (LCKLSTS.DAT).

FCB пользователя подвергаются проверке контрольной суммы перед операциями ввода-вывода для защиты целостности файловой системы от поврежденных FCB. Функции открытия и создания файла вычисляют и присваивают контрольные суммы FCB. Функции чтения, записи, блокировки записи, разблокировки записи и закрытия файла впоследствии проверяют и повторно вычисляют контрольные суммы при изменении FCB. Если BDOS обнаруживает ошибку контрольной суммы FCB, она не выполняет требуемую команду. Вместо этого она или завершает процесс вызова с ошибкой, или если процесс находится в режиме возврата ошибок BDOS (см. [Раздел 2.2.13](#_Сброс,_доступ_и_2)), она возвращается в процесс с кодом ошибки.

Системный список блокировки устанавливается в процессе генерации системы, во время которого пользователь может определить размер списка и также указать ограничения на количество файлов, которые может открыть один процесс и количество записей, которые может заблокировать один процесс. Каждый раз, когда процесс успешно открывает файл или блокирует запись, файловая система BDOS выделяет запись в системном списке блокировки, чтобы записать этот факт. Файловая система использует эту информацию для:

* препятствования процессу удаления, переименования или обновления атрибутов файла открытого другим процессом.
* препятствования процессу открытия файла, открытого в настоящее время другим процессом, если оба процесса не открывают файл в режиме блокировки или только для чтения.
* препятствования процессу сброса диска, на котором у другого процесса есть открытый файл.
* препятствования процессу блокирования или обновления записи, в настоящее время заблокированной другим процессом. Смотрите [Раздел 2.2.10](#_Одновременный_доступ_к) для получения дополнительной информации о блокировании и разблокировании записей.

По соображениям эффективности, только для некоторых функций файловая система проверяет, использует ли другой процесс для указанного FCB открытый файл. Этими функциями являются: открытие файла, создание файла, удаление файла, переименование файла и установка атрибутов файла. Для открытых FCB контрольная сумма FCB определяет, может ли процесс использовать FCB. По определению допустимая контрольная сумма FCB подразумевает, что файл был успешно открыт, и запись о файле находится в системном списке блокировки. Когда процесс закрывает файл постоянно, файловая система удаляет файл из системного списка блокировки и аннулирует поле его контрольной суммы FCB.

Существует еще несколько ситуаций, когда файловая система удаляет записи открытых файлов из системного списка блокировки для процесса. Например, если процесс делает вызов удаление файла, который он имеет открытым в режиме блокировки, файловая система удаляет файл и также удаляет запись о файле из системного списка блокировки. Удаление открытого файла не рекомендуется использовать в MP/M, но поддерживается для файлов, открытых в режиме блокировки (режим открытия по умолчанию) для обеспечения совместимости с программным обеспечением, написанным для более ранних выпусков MP/M и CP/M. Обратите внимание, что файловая система не удаляет файл, открытый в неблокируемом режиме или режиме только для чтения.

Чтобы гарантировать, что процесс не использует FCB соответствующий удаленному файлу, файловая система впоследствии проверяет все открытые процессом FCB, чтобы гарантировать, что элемент списка блокировки существует для FCB. Каждый открытый FCB проверяется при последующем использовании. Если запись о файле существует в списке блокировки, операция продолжается. В противном случае возвращается ошибка контрольной суммы FCB.

Файловая система выполняет эту проверку открытых FCB для всех ситуаций, когда она очищает запись об открытом файле из системного списка блокировки. Ниже описаны эти ситуации:

* Процесс удаляет файл открытый в режиме блокировки.
* Процесс переименовывает файл открытый в режиме блокировки.
* Процесс обновляет атрибуты с помощью команды BDOS *установить атрибуты файла*, открытого в режиме блокировки.
* Процесс выполняет вызов *освободить диск* на котором у него есть открытый файл.
* Обнаружена смена носителя на диске, у которого есть открытые файлы. Эта ситуация представляет собой особый случай, поскольку процесс не может контролировать происходит ли это и не может влиять на более чем один процесс. Обратитесь к [Разделу 2.2.13](#_Сброс,_доступ_и_1) для получения дополнительной информации об этой ситуации.

Автоматическая проверка открытых FCB в файловой системе после удаления записей о файле из системного списка блокировки может повлиять на производительность. Каждая проверка требует операций поиска в каталоге. По этой причине, настоятельно рекомендуется избегать таких ситуаций в новых программах, разработанных для MP/M II.

### Одновременный доступ к файлам

Несколько процессов могут получить доступ к тому же самому файлу, если каждый процесс открывает файл в одном и том же режиме совместного доступа. BDOS поддерживает два режима совместного доступа: неблокированный и только для чтения. Режим только для чтения функционально идентичен режиму блокировки по умолчанию за исключением того, что несколько процессов могут получить доступ к файлу, и никакой процесс не может изменить его. Файлы, открытые в неблокируемом режиме, представляют более сложную ситуацию, потому что файл, открытый в этом режиме, может быть изменен несколькими процессами одновременно. В результате неблокированный режим отличается в некоторых важных отношениях от других режимов открытия.

Когда процесс открывает файл в неблокируемом режиме, файловая система возвращает 2-байтное поле называемое *идентификатор файла* в байтах FCB r0 и r1. Идентификатор файла является обязательный параметром функций BDOS *блокировка записи*, и *разблокировка записи*.

Файловая система поддерживает два механизма, которые позволяют процессам координировать операции обновления файлов, открытых в неблокируемом режиме. Функции блокирования и разблокирования записей позволяют процессу устанавливать и отказываться от временного владения определенными записями. Блокировка записи не препятствует тому, чтобы другой процесс читал заблокированную запись. Блокируются только операции записи и блокировки для других процессов. Как альтернатива, функция тестирования и записи проверяет текущее содержание записи прежде, чем позволить продолжиться операции записи.

Функции блокирования и разблокирования записи и функция тестирования и записи обеспечивают два принципиально разных подхода координации обновления записи. Когда запись заблокирована, файловая система размещает запись в списке блокировки системы, идентифицируя заблокированную запись и связывая ее с процессом вызова. Функции разблокирования записи удаляет заблокированную запись из списка. Пока запись о заблокированной записи существует в системном списке блокировки, никакой другой процесс не может заблокировать или записать в эту запись. Поскольку системный список блокировки является в MP/M ограниченным ресурсом, процесс ограничен в отношении числа записей, которые он может блокировать.

Функция тестирования и записи, с другой стороны, выполняет эту проверку на уровне ввода-вывода. В рамках одной неделимой операции она проверяет, что текущая версия записи пользователей соответствует версии на диске прежде, чем позволить продолжиться операции записи. В результате она не ограничена как функция блокировки записи. Однако, координация обновления записи обычно может более эффективно выполняться с помощью функций блокировки.

Файловая система BDOS выполняет дополнительные шаги для операций чтения и операций записи в файл, открытый в неблокируемом режиме. Эти дополнительные шаги требуются, поскольку файловая система BDOS поддерживает текущее состояние открытого файла в FCB пользователя. Если несколько процессов имеют один и тот же открытый файл, FCB для этого же файла существуют в памяти каждого процесса. Чтобы гарантировать, что все процессы имеют текущую информацию, файловая система обновляет каталог сразу, когда FCB неблокируемого файла изменяется. Кроме того, файловая система проверяет ошибочные ситуации, такие как конец файла или чтения незаписанных данных из каталога прежде, чем возвратить ошибку. В результате операции чтения и записи менее эффективны для файлов, открытых в неблокируемом режиме, по сравнению с эквивалентными операциями для файлов, открытых в блокируемом режиме по умолчанию.

Расширение файла является также специальной ситуацией для файлов, открытых в неблокируемом режиме. Обычно, когда файл увеличивается, размер файла устанавливается в номере произвольной записи в значение последней записи + 1. Однако, когда увеличивается файл, открытый в неблокируемом режиме, размер файла устанавливается в номер произвольной записи + 1 из последней 128-байтовой записи в последнем блоке данных файла. Процесс должен отслеживать фактическую последнюю запись расширенного файла, открытого в неблокируемом режиме, если это требуется.

### Многосекторный ввод-вывод

Файловая система BDOS обеспечивает возможность чтения или записи нескольких 128-байтных записей в одном вызове функции BDOS. Эта многосекторная функция, может быть представлена как режим "группового обмена" BDOS, позволяя процессу завершить многократные операции ввода-вывода без помех от других работающих процессов. Использование этой функции в прикладной программе может улучшить ее производительность, и также улучшить общую пропускную способность системы. Например, утилита PIP выполняет свой последовательный ввод-вывод с многосекторным числом - 8. Многосекторный ввод-вывод оказывает большое влияние, на производительность обработки последовательного ввода-вывода в системах MP/M II, которые поддерживают блокирование/разблокирование записей в своих XIOS. Повышение производительности достигается, путем избавления от большого процента физических операций предварительного чтения записей XIOS.

Число записей, которое может поддерживать многосекторный ввод-вывод, находятся в диапазоне от 1 до 16. Для транзитных программ значение по умолчанию - 1, потому что CLI инициализирует многосекторный счетчик транзитной программы значением 1 при инициировании программу. Функция BDOS *установка многосекторного счетчика* может использоваться для присвоения счетчику другого значения.

Многосекторный счетчик определяет число операций, которые будут выполнены следующими функциями BDOS:

* Функции последовательного чтения и записи
* Функции произвольного чтения и записи, включая запись с заполнением нулями и тестирования и записи
* Функции блокировки и разблокировки записей

Если многосекторный счетчик равняется N, вызов одной из вышеупомянутых функций эквивалентен созданию N вызовов функций. Если многосекторная операция ввода-вывода прервана ошибкой, файловая система возвращает число успешно обработанных 128-байтовых записей в старшем полубайте регистра H.

### Блокирование и разблокирование XIOS

Дополнительный механизм блокирования и разблокирования физической записи может быть реализован как часть XIOS когда необходимо поддержать физические записи на диске в модулях, больше чем 128 байт. В общем, блокирование и разблокирование записи в XIOS являются прозрачными для файловой системы BDOS, а также программ, которые делают вызовы BDOS файловой системы.

Если этот механизм реализован, XIOS посылает данные или получает данные от файловой системы BDOS в логических записях 128-байт, но обращается к диску с большим размером физической записи. XIOS использует внутренний буфер физической записи, имеющий размер физической записи, чтобы буферизировать логические записи. Процесс создания физических записей из 128-байтовых логических записей называют, *блокирование*, и он требуется для операций записи BDOS. Обратный процесс называют, *разблокирование*, и он требуется для операций чтения BDOS. Для операций записи BDOS XIOS откладывает физическую операцию записи для постоянных дисков (см. [Раздел 2.2.13](#_Сброс,_доступ_и)), если операция записи не в каталог. Для операций чтения BDOS XIOS выполняет физическое чтение, только если текущий физический буфер записи не содержит требуемую логическую запись. Кроме того, если физическая запись является "отложенной" в результате предыдущей операции записи, XIOS выполняет физическую операцию записи до операции чтения.

Отсрочка операции записи физической записи имеет последствия для некоторых прикладных программ. Для тех программ, которые предполагают частое обновление файла, критически важно гарантировать, что состояние файла на диске эквивалентно состоянию файла в памяти после обновления файла. Это является проблемой только для систем, которые реализуют блокирование и разблокирование записей из-за отсрочки операции физической записи. Если система может зависнуть, в тот момент пока физический буфер находится в ожидании, данные будут потеряны. Чтобы предотвратить это, может быть вызвана функция BDOS *очистки буферов*, чтобы выполнить запись любых отложенных физических буферов в XIOS.

Примечание: XDOS автоматически вызывает эту функцию при завершении процесса. Кроме того, файловая система BDOS автоматически выполняет вызов очистки буферов в функции закрытия файла.

### Сброс, доступ и освобождение диска

Функции BDOS сброс дисковой системы, сброс диска, доступ к диску и освобождение диска позволяют процессу контролировать, когда каталог диска должен быть повторно инициализирован для операций с файлами. Когда MPMLDR инициирует MP/M II, все диски инициализируются в сброшенном состоянии. Впоследствии, при обращениям к дискам, они автоматически регистрируются файловой системой. Операция входа в систему инициализирует диск для файловых операций BDOS. В общем случае, как только диск зарегистрирован, нет необходимости повторно регистрировать диск, если не должна быть выполнена смена носителя диска. Однако, MP/M II требует успешного выполнения сброса диска перед сменой носителей. Если диск находится в состоянии сброса, когда носители изменены, следующий доступ к диску регистрирует диск в системе. Обратите внимание, что функции сброс дисков системы и сброс диска обладают схожими действиями за исключением того, что функция сброса дисков системы отправляется всем дискам в системе. Пользователь может определить любую комбинацию дисков, которые будут сброшены с помощью функции сброс диска.

В MP/M II операция сброса диска имеет условный характер. Иначе говоря, файловая система не может сбросить диск для процесса, если другой процесс имеет открытый файл на диске. Однако, точное действие, предпринятое в операции сброса диска зависит от того, является ли диск, который будет сброшен, постоянным или съемным. MP/M II определяет, постоянный дисковод или съемный, путем опроса бита в блоке параметров диска (DPB) в XIOS (Смотрите *Руководство системы MP/M II* для подробного описания DPB). Старший бит 1 в поле размера таблицы контрольной суммы DPB определяет диск как постоянный. В MP/M II, назначение диска имеет решающее значение для операции сброса, которая описана ниже.

BDOS сначала определяет, есть ли в настоящее время на диске, который будет сброшен, какие-либо открытые файлы. Если нет ни одного, происходит сброс. В противном случае, если диск является постоянным диском и если диск не только для чтения, операция сброса не выполняется, но возвращается успешный результат процессу вызова. Однако, если диск является съемным или только для чтения, файловая система определяет, есть ли у других процессов открытые файлы на диске. Если они существуют, операция сброса диска отклоняется, и процессу вызова возвращается код ошибки. Если все файлы, открытые на диске, принадлежат процессу вызова, файловая система выполняет операцию "частичного" сброса диска и возвращает успешный результат процессу вызова. Это означает, что в следующий раз привод доступен, операция регистрации производится, только если на диске обнаружена смена носителя. Логика осуществления операции сброса дисковода показана на *Рисунке 2-1*.

┌────────────────┐

│ Файлы на диске │ Да

│ открыты ? ├────────┐

└────────┬───────┘ │

│ Нет │

│ │

│ ┌───────┴───────┐

│ │ Диск │ Да

│ │ съемный ? ├───────────────┐

│ └───────┬───────┘ │

│ │ Нет │

│ ┌───────┴───────┐ │

│ │ Диск только │ Да │

│ │ для чтения ? ├───────────────┤

│ └───────┬───────┘ │

│ │ Нет │

┌─────┴────┐ ┌───────┴───────┐ ┌────────┴───────┐

│ Сброс │ │ Не сбрасывать │ │ Открытые файлы │ Да

│ диска │ │ диск │ │ принадлежат ├────────┐

└─────┬────┘ └───────┬───────┘ │ другому │ │

│ │ │ процессу? │ │

│ │ └────────┬───────┘ │

│ │ │ Нет │

│ │ ┌───────┴──────┐ │

│ │ │ Выполняется │ │

│ │ │ частичный │ │

│ │ │ сброс │ │

│ │ └───────┬──────┘ │

┌─────┴────┐ │ │ ┌────┴────┐

│ Успех │ │ │ │ Отказ в │

│ сброса ├───────────┴───────────────────────┘ │ сбросе │

│ диска │ │ диска │

└──────────┘ └─────────┘

Рисунок ‑ Сброс дисковой системы

Если файловая система обнаруживает, смену носителя на диске после частичного сброса, она производит удаление из системного списка блокировки всех открытых файлов на диске и впоследствии проверяет, что все открытые FCB в файловых операциях для процесса владельца (см. [Раздел 2.2.9](#_Защита_файлов)). Диск также повторно регистрируется. Во всех других случаях, когда обнаруживается смена носителей на диске, файловая система выполняет следующие шаги: Все открытые файлы на диске удаляются из системного списка блокировки, и все процессы, владеющие очищенным файлом, помечаются для автоматической проверки открытых FCB. Диск затем переводится в состояние только для чтения. Он повторно не регистрируется, пока не выдается сброс диска. Примечание: Если процесс ссылается на файл, удаленный из системного списка блокировки в команде BDOS, которая пытается открыть FCB, он возвращает ошибку контрольной суммы FCB файловой системы BDOS.

Функции *доступ к диску* и *освобождение диска* выполняют в MP/M II специальные действия. Функция доступ к диску вставляет "фиктивный" элемент открытого файла в системный список блокировки для каждого указанного диска. Пока этот элемент существует в системном списке блокировки, диск не может быть сброшен другим процессом. Функция *освобождение диска* производит очистку списка блокировки всех открытых элементов, включая элементы открытых файлов, принадлежащих процессу вызова на указанных дисках. Любая последующая ссылка на эти файлы вызовом функции BDOS, требующим открытия FCB, приводит к возврату ошибки контрольной суммы FCB.

Функция BDOS *защита диска от записи* в MP/M II имеет особые свойства. Эта функция может использоваться, чтобы установить указанному диску состояние только для чтения. Однако MP/M II не позволяет процессу устанавливать диску состояние только для чтения, если у другого процесса есть открытый файл на диске. Это относится к сменным и постоянным дискам. Если процесс успешно установил состояние только для чтения, он может препятствовать тому, чтобы другие процессы сбросили диск, или открыли файл на диске, или выполнили для диска вызов *Access Drive* (доступ к диску). Пока открытый файл или "фиктивный" элемент, принадлежащий процессу, находятся в системном списке блокировки, никакой другой процесс не может сбросить диск чтобы вывести его из состояния только для чтения.

### Обработка ошибок BDOS

Файловая система BDOS имеет расширенные возможности обработки ошибок. При обнаружении ошибки, она может ответить тремя способами:

1. Возвратить вызывающему процессу код возврата в регистре A, H и L, определяющий ошибку.
2. Вывести сообщение об ошибке на консоль и прервать процесс.
3. Вывести сообщение об ошибке на консоль и вернуться к процессу вызова как в методе 1.

Файловая система обрабатывает большинство ошибок которые она обнаруживает с помощью метода 1. Виды ошибок, которые файловая система обрабатывает с помощью методов 2 и 3 называются "физические" и "расширенные" ошибки. Функция BDOS *Set Error Mode* (установка режима ошибок) определяет, как файловая система обрабатывает физические и расширенные ошибки. Режим ошибок BDOS может существовать в трех состояниях. В *состоянии по умолчанию*, BDOS отображает сообщение об ошибке и завершает процесс вызова (метод 2). В *режиме возврата* ошибки, BDOS возвращает управление вызывающему процессу с ошибкой определенной в регистрах A, H и L (метод 1). В *режиме возврата и отображения*, BDOS возвращает управление вызывающему процессу с ошибкой определенной в регистрах A, H и L, а также отображает сообщение об ошибке в консоли (метод 3). Оба режима возврата гарантируют, что MP/M II не прекращает процесс из-за физической или расширенной ошибки. Режим возврата и отображения также позволяет вызывающему процессу использовать встроенные сообщения об ошибках файловой системы BDOS. Физические и расширенные ошибки отображается на консоли в следующем формате:

BDOS Err on d: Сообщение об ошибке

BDOS function: nn File: filename.type

где "d" - имя выбранного диска при обнаружении ошибки. "Сообщение об ошибке" идентифицирует ошибку; "nn" - номер функции BDOS, и "filename.type" определяет файл, указанный функцией BDOS. Если функция BDOS не включает FCB, информация о файле опущена.

**Физические ошибки** BDOS идентифицируются следующими сообщениями об ошибках:

* Bad Sector
* Select
* File R/O
* R/O

Ошибка "Bad Sector" (Плохой сектор) является результатом ошибки возвращаемой в BDOS из модуля XIOS. Файловая система создает вызовы чтения и записи XIOS, чтобы выполнить соответствующие файловые вызовы BDOS. Если подпрограмма XIOS чтения или записи обнаруживает ошибку, она возвращает код ошибки BDOS, приводящей к этой ошибке.

Ошибка "Select" (выбора) также следует из состояния ошибки, возвращенного в BDOS из модуля XIOS. BDOS создает вызов XIOS *Select Disk* (выбор диска) перед доступом к диску для выполнения запрашиваемой функции BDOS. Если XIOS не поддерживает выбранный диск, он возвращает код ошибки, приводящий к этой ошибке.

Ошибка "File R/O" возвращается каждый раз, когда процесс делает операцию записи в файл с установленным атрибутом только для чтения.

Ошибка "R/O" возвращается каждый раз, когда процесс делает операцию записи на диск, который находится в состоянии только для чтения. Диск может быть помещен в состояние только для чтения явно с помощью функции BDOS *Write Protect Disk* (защита диска от записи), или неявно, если файловая система обнаруживает смену носителя в дисководе.

**Расширенные ошибки** BDOS идентифицируются следующими сообщениями об ошибках:

* File opened in Read/only Mode
* File Currently Opened
* Close Checksum Error
* Password Error
* File Already Exists
* Illegal ? in FCB
* Open File Limit Exceeded
* No Room in System Lock List

Ошибка "File Opened in Read/only Mode" возвращается, если процесс пытается записать в файл открытый в режиме только для чтения. Файл может быть открыт в режиме только для чтения явно, или неявно двумя способами. Если открыт файл пользователя 0, когда текущий номер пользователя ненулевой, файл открывается в режиме только для чтения. Кроме того, если файл защищен паролем в режиме записи, и пароль не предоставляется при вызове *open*, возвращается эта ошибка при попытке записи в файл.

Ошибка "File Currently open" возвращается, если процесс пытается удалить, переименовать или изменить атрибуты файла, открытого другим процессом. Эта ошибка также возвращается, если процесс пытается открыть файл в режиме несовместимом с режимом, в котором файл был открыт другим процессом.

Сообщение "Close Checksum Error" возвращается, если BDOS обнаруживает ошибку контрольной суммы в FCB переданной в файловую систему вызовом BDOS закрыть файл.

Ошибка "File Password" возвращается, если пароль файла не предоставлен, или он неправильный.

Ошибка "File Already Exists" возвращается для функции BDOS создать файл и переименовать файл, если BDOS обнаруживает конфликт в имени и расширении файла.

Ошибка "Illegal ? in FCB" возвращается каждый раз, когда BDOS обнаруживает "?" в поле имени файла или расширения переданном FCB для функций BDOS переименовать файл, установка атрибутов файла, открытие файла и создание файла.

Ошибка "Open File Limit Exceeded" возвращается если процесс превышает предел блокировок файла, определенный в системной таблице блокировок во время генерации системы. Возвратить эту ошибку могут функции открыть файл, создать файл и доступ к диску.

Ошибка "No Room in System Lock List" возвращается, если нет места для новых записей в системном списке блокировок для новых записей. Размер системного списка блокировок является параметром генерации системы. Функции открыть файл, создать файл и доступ к диску могут возвратить эту ошибку.

Следующие пункты описывают соглашения о возвращаемых кодах ошибок функций файловой системы BDOS. Большинство функций файловой системы BDOS делятся на три категории в отношении кодов возврата. Они возвращают код ошибки, код каталога или флаг ошибки. Соглашения об ошибках разработаны, чтобы позволить программам, написанным для более ранних версий CP/M и MP/M работать без изменений.

Следующие функции BDOS возвращают **код ошибки** в регистре A:

20. Read Sequential

21. Write Sequential

33. Read Random

34. Write Random

40. Write Random w/Zero Fill

41. Test and Write Record

42. Lock Record

43. Unlock Record

Определения кодов ошибок в регистре A показаны в *Таблице 2-8*.

Таблица ‑ Коды ошибок BDOS

| **Код** | **Определение** |
| --- | --- |
| 00 | Успешное завершение функции |
| 255 | Физическая ошибка: обратитесь к регистру H |
| 01 | Чтение незаписанных данных  Нет места в каталоге (последовательная запись) |
| 02 | Не доступен блок данных |
| 03 | Нет возможности закрыть текущий экстент |
| 04 | Поиск до незаписанного экстента |
| 05 | Нет места в каталоге |
| 06 | Номер произвольной записи вне диапазона |
| 07 | Ошибка соответствия записи (проверка и запись) |
| \* 08 | Запись заблокирована другим процессом  (ограничивается файлами, открытыми в разблокированном режиме) |
| 09 | Недопустимый FCB (предыдущий вызов BDOS чтения или записи, возвратил код ошибки и признал недопустимым FCB) |
| 10 | Ошибка контрольной суммы FCB |
| \* 11 | Ошибка проверки освобождения нераспределенного блока |
| \*\* 12 | Процесс превысил лимит блокировки записи |
| \*\* 13 | Недопустимый идентификатор файла |
| \*\* 14 | Нет места во внутренней таблице блокировки |
| \* - возвращаются только для файлов, открытых в неблокируемом режиме | |
| \*\* - возвращается только функцией *Lock Record* (блокировка записи) для файлов открытых в неблокируемом режиме | |

Следующие функции BDOS возвращают **код каталога** в регистре A:

15. Open File

16. Close File

17. Search For First

18. Search For Next

19. Delete File

22. Make File

23. Rename File

30. Set File Attributes

100. Set Directory Label

101. Read File XFCB

102. Write File XFCB

Определения кодов каталога в регистре A показаны в *Таблице 2-9*.

Таблица ‑ Коды каталога BDOS

| **Код** | **Определение** |
| --- | --- |
| 00 - 03 | Успешное завершение функции |
| 255 | Неуспешное завершение функции |

За исключением функций BDOS поиска, значения кодов каталога (0-3) не имеют никакого значения, кроме указания успешного результата. Вместе с тем, для функций поиска, успешный код каталога определяет относительную начальную позицию элемента каталога в текущем буфере DMA вызывающего процесса.

Если используется функция *Set BDOS Error Mode* (установка режима ошибок BDOS), для перевода BDOS в режим возврата ошибок, следующие функции возвращают **флаг ошибки** при физических ошибках:

14. Select Disk

35. Compute File Size

38. Access Drive

46. Get Disk Free Space

48. Flush Buffers

101. Return Directory Label Data

Определения флагов ошибок в регистре A показаны в *Таблице 2-10*.

Таблица ‑ Флаги ошибок BDOS

| **Код** | **Определение** |
| --- | --- |
| 00 | Успешное завершение функции |
| 255 | Физическая ошибка: обратитесь к регистр H |

BDOS возвращает в регистре H значения для всех трех указанных выше категорий в следующем формате:

|  |  |
| --- | --- |
| N1 | N2 |

где N1 обозначает старший полубайт и N2 обозначает младший полубайт. Следующие правила определяют присвоения значений для N1 и N2:

N1 Для функций, возвращающих коды ошибок BDOS задает в N1 количество успешно прочтенных или записанных секторов, прежде чем встретится ошибка. Эта информация возвращается только, когда процесс использует функцию установить многосекторный счетчик для установки значения количества логических сектора BDOS, отличного от одного. В противном случае BDOS устанавливает N1 равным нулю. Успешное выполнение функций чтения и записи также устанавливает N1 равным нулю.

Функции, которые возвращают код каталога или флаг ошибки, обнуляют N1.

N2 Значения, содержавшиеся в N2, идентифицируют физические и расширенные ошибки BDOS. BDOS возвращает значения в N2, только если она находится в одном из режимов возврата ошибок. Иначе, она обнуляет N2. *Таблица 2-11* перечисляет физические и расширенные коды ошибок, возвращаемые в N2.

Таблица ‑ Физические и расширенные ошибки BDOS

|  |  |
| --- | --- |
| **Код** | **Определение** |
| 00 | - Нет ошибок или нет ошибок в регистре H |
| 01 | - Плохой сектор : систематическая ошибка |
| 02 | - Только для чтения : диск только для чтения |
| 03 | - Только для чтения : файл только для чтения |
|  | - Файл открыт в режиме только для чтения |
| 04 | - Выбор : ошибка выбора диска |
| 05 | - Файл в настоящее время открыт |
| 06 | - Ошибка контрольной суммы при закрытии |
| 07 | - Ошибка пароля |
| 08 | - Файл уже существует |
| 09 | - Недопустимый ? в FCB |
| 10 | - Превышен лимит открытых файлов |
| 11 | - Нет места в системном списке блокировок |

Примечание: Регистр H равен нулю, если вызванная функция успешна. Кроме того, BDOS устанавливает N2 равным нулю, когда регистр возвращает значение, отличное от 255. Для функций, возвращающих коды каталога, если при возврате регистр А содержит значение 255, N2 обозначает ошибку, когда BDOS находится в режиме возврата ошибок.

Следующие две функции представляют собой особый случай, поскольку они возвращают адрес в регистрах Н и L.

27. Get Addr(Alloc)

31. Get Addr(Disk Parms)

Если BDOS находится в режиме возврата ошибок, и обнаруживает физическую ошибку для этих функций, она возвращает процессу вызова во всех регистрах A, H, и L значение 255. В противном случае, она не возвращает код ошибки.

В МР/М II следующие функции также представляют собой особый случай.

13. Reset Disk System

28. Write Protect Disk

37. Reset Drive

Эти функции возвращают вызывающему процессу во всех регистрах A, H, и L значения установленные 255, если другой процесс имеет открытый файл или выполнил вызов BDOS *Access Drive* (доступ к диску), который предотвращает сброс или операцию защиты от записи (см. [Раздел 2.2.13](#_Сброс,_доступ_и_3)). Если BDOS не находится в режиме возврата ошибок, эти функции также выводят на экран сообщение об ошибке, идентифицируя процесс, который предотвратил запрошенную операцию.

## Инициализации базовой страницы

Область памяти, расположенная начиная с адреса BASE+0000H до BASE+00FFH, называется базовой страницей сегмента памяти (BASE = начальный адрес сегмента памяти). Базовая страница содержит несколько сегментов кода и данных, которые используются транзитными программами при выполнении в MP/M II. Код и области данных приведены ниже для справки. Все адреса приведены относительно начала сегмента памяти.

Таблица ‑ Области базовой страницы

| **Расположение** | | | | **Содержание** |
| --- | --- | --- | --- | --- |
| **от** | | **до** | |
| 0000H | - | | 0002H | Содержит команду перехода к точке входа завершения процесса XIOS в XIOS BASE+3. Эта точка входа может также использоваться для прямых вызовов XIOS опроса состояния консоли, вывода на консоль и примитивных функций вывода на устройства печати. |
| 0003H | - | | 0004H | (Зарезервированы) |
| 0005H | - | | 0007H | Содержит команду перехода к BDOS и XDOS, и служит двум целям: JMP 0005H обеспечивает основную точку входа для BDOS и XDOS и LHLD 0006H помещает поле адреса инструкции перехода в регистровую пару HL. Это значение минус 1 является самым высоким адресом сегмента памяти, доступным транзитной программе. Примечание: программа RDT изменяет это поле адреса, чтобы отразить уменьшение размеров памяти в режиме отладки. |
| 0008H | - | | 003AH | Зарезервированные расположения рестартов прерываний 1 – 7 |
| 003BH | - | | 004FH | (В настоящее время не используются - зарезервированы) |
| 0050H |  | |  | Определяет диск, из которого считана транзитная программа. Значение 0 определяет диск по умолчанию, значение от 1 до 16 определяют диски от А до P. |
| 0051H | - | | 0052H | Содержит адрес пароля - поле первого операнда хвоста команды в буфере DMA по умолчанию, начинающемся в 0080H. CLI обнуляет это поле, если пароль для первого операнда хвоста команды не определен. |
| 0053H |  | |  | Содержит длину поля пароля для первого операнда хвоста команды. CLI также обнуляет это поле, если пароль для первого хвоста команды не определен. |
| 0054H | - | | 0055H | Содержит адрес пароля – поле второго операнда хвоста команды в буфере DMA по умолчанию, начинающемся в 0080H. CLI обнуляет это поле, если пароль для первого операнда хвоста команды не определен. |
| 0056H |  | |  | Содержит длину поля пароля для второго операнда хвоста команды. CLI также обнуляет это поле, если пароль для первого хвоста команды не определен. |
| 0057H | - | | 005BH | (В настоящее время не используются - зарезервированы) |
| 005CH | - | | 007BH | Область первого блока управления файлом (FCB) по умолчанию инициализируется CLI для транзитной программы из первого операнда хвоста команды командной строки (если он существует). |
| 006CH | - | | 007BH | Область второго блока управления файлом (FCB) по умолчанию инициализируется CLI для транзитной программы из второго операнда хвоста команды командной строки (если он существует). Примечание: эта область перекрывает последние 16 байт области первого FCB по умолчанию. Чтобы использовать информацию в этой области, транзитная программа должна скопировать ее в другое расположение перед использованием области первого FCB. |
| 007CH | - | | 007CH | Текущая позиция записи области первого FCB по умолчанию. Это поле области первого FCB по умолчанию используется для последовательной обработки записей. |
| 007DH | - | | 007FH | необязательная позиция произвольной записи по умолчанию. Это поле является расширением области первого FCB по умолчанию, используемого в произвольной обработке записей. |
| 0080H | - | | 00FFH | Буфер по умолчанию, длиной 128 байт (также заполненный хвостом команды, когда консоль загружает транзитную программу). |

CLI инициализирует базовую страницу до инициализации транзитной программы. Поля в BASE+0050H и выше инициализируются из командной строки, вызывающей транзитную программу.

Формат командной строки транзитной программы обычно принимает форму:

<command> <command tail>

где

<command> => {d:}filename{;password}

<command tail> => (no command tail)

=> <file spec>

=> <file spec><delimiter><file spec>

<file spec> => {d:}filename{.type}{;password}

Если диск {d:} определен в поле <command>, CLI инициализирует поле диска команды в 0050H к индексу диска (A = 1, ... , P = 16). Иначе, он обнуляет поле.

FCB по умолчанию в 005CH определен, если введен хвост команды. Иначе, поля в 5CH, с 68H до 6BH установлены в двоичные нули, поля с 5DH до 67H заполнены пробелами. Поля с 51H до 53H установлены, если определен пароль в хвосте команды первой <file spec>. В противном случае эти поля обнулены.

FCB по умолчанию в 006CH определен, если вторая <file spec> существует в хвосте команды. Иначе, поля в 6CH, с 78H до 7BH установлены в двоичные нули, поля от 5DH до 67H заполнены пробелами. Поля с 54H до 56H установлены, если пароль определен в хвосте команды второй <file spec>. В противном случае эти поля обнулены.

Транзитные программы часто используют FCB по умолчанию в 005CH для файловых операций. Этот FCB может также использоваться для произвольного доступа к файлам, потому что три байта, начиная с 007DH доступны для этой цели. Однако транзитная программа должна скопировать содержание FCB по умолчанию в 006CH в другую область перед использованием FCB по умолчанию в 005CH, потому что операция открытия для FCB по умолчанию в 005CH перезаписывает данные FCB в 006CH.

Адрес DMA по умолчанию для транзитных программ - BASE+0080H. CLI также инициализирует эту область, чтобы содержать хвост команды из командной строки. Первая позиция содержит число символов в командной строке, далее расположены символы командной строки. Символам командной строки предшествует начальный пробел и они переводятся в верхний регистр ASCII. Поскольку 128-байтовая область, начинающаяся в BASE+0080H, является DMA по умолчанию, файловая система BDOS перемещает 128-байтовые записи в эту область при операциях чтения и осуществляет доступ к этой области при операциях записи. Транзитная программа должна извлечь информацию о хвосте команды из этого буфера перед выполнением файловых операций, если она явно не изменяет адрес DMA с помощью функции BDOS *Set DMA Address* (Установка адреса DMA*)*. Поля базовой страницы с 0051H до 0056H определяют расположение полей пароля первых двух спецификаций файла в хвосте команды, если они существуют. Эти поля обеспечивают доступ к полям паролей без разбора хвоста команды. Однако транзитная программа должна сохранить пароль или изменить адрес DMA перед выполнением операций с файлами.

В следующем примере показана инициализация полей командной строки базовой страницы. Если предположить, что следующая командная строка введена в консоли:

A:PROGRAM B:FILE.TYP;PASS C:FILE.TYP;PASSWORD

Шестнадцатеричный дамп с BASE+0050H до BASE+00A5H показал бы инициализацию базовой страницы, выполненную CLI.

0050H: 01 8D 00 04 9D 00 08 00 00 00 00 00 02 46 49 4C . . . . . .FIL

0060H: 45 20 20 20 20 54 59 50 00 00 00 00 03 46 49 4C E . .TYP. .FIL

0070H: 45 20 20 20 20 54 59 50 00 00 00 00 00 00 00 00 E . .TYP. .

0080H: 24 20 42 3A 46 49 4C 45 2E 54 59 50 3B 50 41 53 . B:FILE.TYP;PAS

0090H: 53 20 43 3A 46 49 4C 45 2E 54 59 50 3B 50 41 53 S C:FILE.TYP;PAS

00A0H: 53 57 4F 52 44 00 SWORD.

## Вызовы функций BDOS

|  |
| --- |
| **Функция 0: System Reset** (**Сброс системы**) |
| Входные параметры:  Регистр C: 00H |

Функция *System Reset* (Сброс системы) завершает вызывающий процесс, высвобождая все системные ресурсы, принадлежащие процессу. В общем случае процесс может владеть одним или более из следующих ресурсов: сегментами памяти, консолями, принтерами, сообщениями взаимного исключения и элементами списков записей открытых файлов и заблокированных записей. Все высвобожденные ресурсы становятся доступными другим процессам в системе. Например, если системная консоль освобождается процессом завершения, она обычно отдается консолям TMP. Это происходит, когда TMP является процессом с наивысшим приоритетом, ожидающим консоль.

Обычно, функция *System Reset* в MP/M II работает так же, как в CP/M: вызывающая программа завершается, и пользователь получает командную строку. Обратите внимание, что в MP/M II дисковая подсистема не сбрасывается при сбросе системы.

Для транзитных программ сброс системы эквивалентен переходу к BASE+0.

|  |
| --- |
| **Функция 1: Console Input (Ввод с консоли)** |
| Входные параметры:  Регистр C: 01H  Возвращаемые значения:  Регистр C: Символ ASCII |

Функция *Console Input* читает следующий символ из консольного устройства в регистр A. Большинство управляющих символов, включая возврат каретки, перевод строки и забой - Backspace (Ctrl-H) передаются с эхом на консоль. Символы табуляции (Ctrl-I) перемещают курсор на 8 позиций вправо. Однако, символы завершения процесса (Ctrl-C) и отсоединения процесса (Ctrl-D) перехватываются BDOS (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств)). BDOS не возвращает управление процессу вызова, пока символ вводится, таким образом приостанавливая выполнение, если символ не готов.

MP/M II выполняет вызов XDOS *Attach Console* (функция 146, присоединение консоли) для процесса вызова, если он не владеет консолью (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)).

|  |
| --- |
| **Функция 2: Console Output (Вывод в консоль)** |
| Входные параметры:  Регистр C: 02H  Регистр E: Символ ASCII |

Функция *Console Output* (Вывод в консоль) отправляет символ ASCII из регистра E на консольное устройство. Она разворачивает символы табуляции (Ctrl-I), перемещая курсор на 8 позиций вправо и проверяет запуск прокрутки (Ctrl-S), остановку прокрутки (Ctrl-Q), окончание процесса (Ctrl-C) и отсоединение от процесса (Ctrl-D) (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_2)).

MP/M II выполняет вызов XDOS *Attach Console* (функция 146, присоединение консоли) для процесса вызова, если он не владеет консолью (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)).

|  |
| --- |
| **Функция 3: Raw Console Input (Необработанный консольный ввод)** |
| Входные параметры:  Регистр C: 03H  Возвращаемые значения:  Регистр A: Символ ASCII |

Функция *Raw Console Input* читает следующий консольный символ в регистр A. Она читает все символы, включая управляющие символы без проверки или интерпретации.

MP/M II выполняет вызов XDOS *Attach Console* (функция 146, присоединение консоли) для процесса вызова, если он не владеет консолью (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)).

MP/M II не поддерживает CP/M функцию *Reader Input*, потому что система рассматривает все символьные устройства ввода-вывода, такие как считыватель/перфоратор как консоли.

|  |
| --- |
| **Функция 4: Raw Console Output (Необработанный вывод на консоль)** |
| Входные параметры:  Регистр C: 04H  Регистр E: Символ ASCII |

Функция *Raw Console Output* отправляет символ ASCII из регистра E на консольное устройство. Она не проверяет выходной символ, т.е. табуляция не расширяется, и не осуществляется проверка управляющих символов.

MP/M II выполняет вызов XDOS *Attach Console* (функция 146, присоединение консоли) для процесса вызова, если он не владеет консолью (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)).

MP/M II не поддерживает CP/M функцию *Punch Output*.

|  |
| --- |
| **Функция 5: List Output (Вывод на устройство печати)** |
| Входные параметры:  Регистр C: 05H  Регистр E: Символ ASCII |

Функция *List Output* отправляет символ ASCII из регистра E на устройство печати.

MP/M II выполняет вызов XDOS *Attach List* (функция 158) для процесса вызова, если он не владеет устройством печати (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)).

|  |
| --- |
| **Функция 6: Direct Console Input (Прямой ввод-вывод с консоли)** |
| Входные параметры:  Регистр C: 06H  Регистр E: 0FFH (ввод/статус) или  0FEH (статус) или  0FDH (ввод)  char (вывод)  Возвращаемые значения:  Регистр A: Символ или статус (не значение) |

MP/M II поддерживает прямой ввод-вывод с консоли для тех специализированных приложений, где требуется консольный ввод и вывод без каких-либо добавлений. Программист должен использовать прямой ввод-вывод с консоли осторожно, потому что он обходит все нормальные функции управляющих символов. Программы, которые выполняют прямой ввод-вывод через BIOS в предыдущих версиях CP/M, должны быть изменены, чтобы использовать прямой ввод-вывод с новым BDOS, чтобы они могли полностью поддерживаться в будущих выпусках MP/M и CP/M.

Процесс вызывает функцию 6 передавая одно из четырех различных значений в регистре E. Эти данные приведены в *Таблице 2-13*, ниже.

Таблица ‑ Входные параметры функции 6

|  |  |
| --- | --- |
| **Регистр E** | **Значение** |
| 0FFH | Команда консольного ввода/состояния, возвращает входной символ. Если символ не готов, возвращается нулевое значение. |
| 0FEH | Команда опроса состояния консоли (При возврате, регистр A содержит 00, если символ не готов, иначе он содержит FFH.) |
| 0FDH | Команда ввода с консоли, возвращает входной символ. Эта функция приостановит процесс вызова, пока символ не будет готов. |
| ASCII | Функция 6 предполагает, что символ в регистре E содержит допустимый символ ASCII и отправляет его на консоль. |

Примечание: MP/M II не совместима с MP/M 1.1 в отношении функции 6 с параметром E=FFH. В MP/M 1.1 команда прямого ввода с консоли (E=FFH) приостанавливает процесс вызова, пока символ не введен, тогда как в MP/M II немедленно возвращается с нулем, если символ не доступен. Чтобы обновить программы, использующие функцию 6 с E=FFH в MP/M 1.1 для работы в MP/M II, необходимо использовать команду прямого ввода (E=FDH). Изменение по отношению MP/M 1.1 потребовалось, чтобы обеспечить совместимость обработки прямого консольного ввода-вывода с CP/M, MP/M II, CP/M-86 и MP/M-86.

MP/M II выполняет вызов XDOS *Attach Console* (функция 146) для процесса вызова, если он не владеет консолью (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)). MP/M II выполняет диспетчеризацию, если выполненная команда прямого ввода/состояния консоли (E=FFH) возвращает 0, указывающий, что символ не готов.

|  |
| --- |
| **Функция 7: Get I/O Byte (Получить байт ввода-вывода)** |

MP/M II не поддерживает функцию *Get I/O Byte*.

|  |
| --- |
| **Функция 8: Set I/O Byte (Установить байт ввода-вывода)** |

MP/M II не поддерживает функцию *Set I/O Byte*.

|  |
| --- |
| **Функция 9: Print String (Печать строки)** |
| Входные параметры:  Регистр C: 09H  Регистр DE: Адрес строки |

Функция *Print String* отправляет символы строки, расположенной в памяти по адресу, содержащемуся в регистровой паре DE на консоль, пока она не встречается с символом "$" в строке. Функция 9 разворачивает символы табуляции (Ctrl-I), перемещая курсор на 8 позиций вправо. Она также проверяет запуск прокрутки (Ctrl-S), остановку прокрутки (Ctrl-Q), окончание процесса (Ctrl-C) и отсоединение от процесса (Ctrl-D) (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)).

MP/M II выполняет вызов XDOS *Attach Console* (функция 146) для процесса вызова, если он не владеет консолью (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)).

|  |
| --- |
| **Функция 10: Read Console Buffer (Чтение консольного буфера)** |
| Входные параметры:  Регистр C: 0AH  Регистр DE: Адрес буфера  Возвращаемые значения:  Символы в буфере консоли |

Функция *Read Console Buffer* читает строку отредактированного консольного ввода в буфер, адресуемый регистровой парой DE. Она завершает ввод, когда буфер заполнен или когда она встречает символ возврата каретки (Ctrl-M) или перевода строки (Ctrl-J).

Входной буфер, адресуемый DE имеет следующий формат:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DE: | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | ... | +n |
|  | mx | nc | c1 | c2 | c3 | c4 | c5 | c6 | c7 | ... | ?? |

где "mx" - максимальное число символов, которые может содержать буфер, а "nc" - число символов, помещенных в буфер. Символы, введенные оператором, следуют за значением "nc". Значение "mx" должно быть установлено, до вызова Функции 10 и может принимать значения от 1 до 255. Установка "mx" в ноль эквивалентно установке "mx" в единицу. Значение "nc" возвращается вызывающему процессу и может изменяться от нуля до "mx". Если nc < mx, то неинициализированные позиции следуют за последним символом, обозначенным "??" на рисунке выше. Обратите внимание, что завершающий символ возврата каретки или перевода строки не помещается в буфер и не включается в счетчик "nc".

Функция 10 распознает управляющие символы редактирования, приведенные в *Таблице 2-14* ниже.

Таблица ‑ Управляющие символы редактирования буфера консоли

|  |  |
| --- | --- |
| **Символ** | **Управляющие символы редактирования** |
| rub/del | Стирание и дублирование на экране последнего символа |
| Ctrl-C | Вызов процедуры "горячий старт" (если Ctrl-C в начале строки) |
| Ctrl-E | Вызывает физический конец строки |
| Ctrl-H | Забой одной символьной позиции |
| Ctrl-J | (перевод строки) завершает входную строку |
| Ctrl-M | (возврат каретки) завершает входную строку |
| Ctrl-P | Эхо консольного вывода на устройство печати |
| Ctrl-R | Перепечатывает текущую строку после новой строки |
| Ctrl-U | Удаляет текущую строку после новой строки |
| Ctrl-X | Забой к началу текущей строки |

Функции управления, которые возвращают курсор в крайнее левое положение (например, Ctrl-X) делают это только до позиции столбца, где закончилась командная строка (в более ранних версиях, курсор возвращался к самому левому краю). Это соглашение упрощает исправление данных и ввод строки.

MP/M II выполняет вызов XDOS *Attach Console* (функция 146) для процесса вызова, если он не владеет консолью (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)).

|  |
| --- |
| **Функция 11: Get Console Status (Получить статус консоли)** |
| Входные параметры:  Регистр C: 0BH  Возвращаемые значения:  Регистр A: Статус консоли |

Функция *Get Console Status* проверяет был ли символ введен в консоли. Если символ - готов, функция 11 возвращает в регистре A значение 01H. Если символ не готов, она возвращает значение 00H.

MP/M II выполняет вызов XDOS *Attach Console* (функция 146) для процесса вызова, если он не владеет консолью (см. [Раздел 2.1](#_Интерфейс_BDOS_устройств_1)).

|  |
| --- |
| **Функция 12: Return Version Number (Возврат номера версии)** |
| Входные параметры:  Регистр C: 0CH  Возвращаемые значения:  Регистр HL: Номер версии |

Функция *Return Version Number* предоставляет информацию, которая позволяет программирование независящее от версий. Она возвращает двухбайтовое значение в регистровой паре HL: H содержит 01H для MP/M, и L содержит 30H, номер версии файловой системы BDOS. Функция 12 полезна для написания прикладных программ, которые обеспечивают произвольный и последовательный доступ к файлам, и отключают произвольный доступ при работе в ранних версиях CP/M.

Для получения номера версии MP/M может быть вызвана функция XDOS 163.

|  |
| --- |
| **Функция 13: Reset Disk System (Сброс дисковой системы)** |
| Входные параметры:  Регистр C: 0DH  Возвращаемые значения:  Регистр A: Код возврата |

Функция *Reset Disk System* восстанавливает файловую систему в исходное состояние, в котором все диски находятся в режиме чтения/записи (см. Функции 28 и 29), диском по умолчанию становится дисковод A, и адрес DMA по умолчанию сбрасывается к значению BASE+0080H. Эта функция может использоваться, например, прикладной программой, которая требует изменения диска во время работы. Функция 37 *Reset Drive* (сброс диска) также может быть использована для этой цели.

Эта функция в MP/M II является условной. Если у другого процесса есть открытый файл на съемном или диске только для чтения, сброс дисков отклоняется и диски не сбрасываются.

При возврате, если операция сброса прошла успешно, регистр A обнуляется. В противном случае, регистру A присваивается значение 0FFH (255 в десятичной системе). Если режим ошибок BDOS не является режимом возврата ошибок (см. Функцию 45), то сообщение об ошибке отображается в консоли, идентифицируя процесс, владеющий открытым файлом.

|  |
| --- |
| **Функция 14: Select Disk (Выбор диска)** |
| Входные параметры:  Регистр C: 0EH  Регистр E: Выбираемый диск  Возвращаемые значения:  Регистр A: Флаг ошибки  Регистр H: Физическая ошибка |

Функция *Select Disk* определяет диск, указанный в регистре E в качестве диска по умолчанию для последующих файловых операций BDOS. Регистр E устанавливается в 0 для диска A, 1 для диска B, и так далее до 15 для диска P в полной системе из 16 дисков. Кроме того, функция 14 регистрирует определенный диск, если он в настоящий момент в состоянии сброса. Регистрация привода активизирует каталог дисковода до следующих операций сброса дисковой системы или сброса диска.

FCB, содержащий код дисковода ноль (dr = 00H) автоматически ссылаются на выбранный текущий диск по умолчанию. Однако, FCB со значением кода диска от 1 до 16, игнорируют выбранный диск по умолчанию и непосредственно ссылаются на диски от A до P.

При возврате регистр A содержит 0, если операция выбора была успешна. Если произошла физическая ошибка, то функция выбора выполняет различные действия в зависимости от режима ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS находится в режиме по умолчанию, сообщение, указывающее ошибку, отображается в консоли, и процесс вызова завершается. В противном случае, функция выбора возвращает процессу вызова в регистре A значение 0FFH и в регистре H код одной из следующих физических ошибок:

01 : Систематическая ошибка

04 : Ошибка выбора

|  |
| --- |
| **Функция 15: Open File (Открытие файла)** |
| Входные параметры:  Регистр C: 0FH  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код каталога  Регистр H: Физическая или расширенная ошибка |

Функция *Open File* активирует FCB для файла, который существует в каталоге диска с кодом активного в настоящее время пользователя или пользователя 0. Процесс вызова передает адрес FCB в регистровой паре DE, с байтом 0 в FCB определяющем диск, байтами 1 - 11 определяющими имя и расширение файла и байтом 12 определяющем экстент. Обычно, байт 12 в FCB устанавливается равным 0. Атрибуты интерфейса FCB f5' и f6' определяют режим, в котором файл должен быть открыт как показано ниже:

f5' = 0, f6' = 0 - открытие в режиме блокировки (режим по умолчанию)

f5' = 1, f6' = 0 - открытие в неблокируемом режиме

f5' = 0 или 1, f6' = 1 - открытие в режиме только для чтения

Если файл защищен паролем в режиме чтения, корректный пароль должен быть помещен в первые восемь байтов текущего DMA или быть ранее установлен в качестве пароля по умолчанию (см. Функцию 106). Обратите внимание, что текущее поле записи ("cr") FCB должно быть обнулено процессом вызова, если файл должен быть доступен последовательно начиная с первой записи.

Функция *Open File* выполняет следующие шаги для файлов, открытых в заблокированном режиме или режиме только для чтения. Если текущий пользователь не 0, и файл, который будет открыт, не существует с текущим номером пользователя, функция *Open File* ищет файл пользователя 0. Если файл существует у пользователя 0 и имеет установленный атрибут системный (t2'), открывается файл пользователя 0. После этого режим открытия автоматически устанавливается только для чтения.

Также функция *Open File* выполняет следующее действие для файлов, открытых в заблокированном режиме, когда текущий номер пользователя - 0. Если файл существует в каталоге пользователя 0 и имеет установленными оба атрибута системный (t2') и только для чтения (t1'), режим открытия автоматически устанавливается только для чтения. Обратите внимание, что режим только для чтения подразумевает, что одновременно получить доступ к файлу могут другие процессы, если они открывают файл в режиме только для чтения.

Если операция открытия прошла успешно, FCB пользователя активируется для операций чтения и записи следующим образом. Соответствующая информация из каталога копируется из совпадающего FCB каталога в байты FCB с d0 до dn. Вычисляется контрольная сумма и присваивается FCB. Функции BDOS, которые требуют открытого FCB (например, *Read Sequential*) проверяют корректность контрольной суммы FCB перед выполнением операции. Если файл открыт в неблокируемом режиме, байты FCB r0 и r1 устанавливаются в двухбайтовое значение, названное *идентификатором* файла. Идентификатор файла - обязательный параметр для функций BDOS *Lock Record* (блокирование записи) и *Unlock Record* (разблокирование записи). Если режим открытия переведен в режим только для чтения в FCB пользователя атрибут интерфейса f8' устанавливается равным 1. Кроме того, если файл, на который ссылаются, защищен паролем в режиме записи, и корректный пароль не был передан в DMA или не соответствует паролю по умолчанию, атрибут интерфейса f7' устанавливается равным 1. Для активированного FCB операции записи не поддерживаются, если атрибут интерфейса f7' или f8' имеет значение Истина (True).

Файловая система BDOS также создает элемент для открытого файла в системном списке блокировки, чтобы записать успешную операцию открытия файла. Пока этот элемент существует, ни один другой процесс не может удалять, переименовывать или изменять атрибуты файла. Кроме того, этот элемент не позволяет другим процессам открывать файл, если файл был открыт в режиме блокировки. Он также требует, чтобы другие процессы проверяли режим открытия файла, если файл был открыт в неблокированном режиме или только для чтения. Обычно, этот элемент остается в системном списке блокировки, пока файл постоянно не закрыт или не завершится процесс, который открывал файл.

При успешной операции открытия, функция *open* также создает метку даты и времени доступа для открытого файла, если выполняются следующие условия: указанный диск имеет метку каталога, которая запрашивает метку даты и времени доступа, открытый файл имеет XFCB, и указанный диск для чтения и записи.

По возвращении, функция *open* возвращает код каталога в регистре A со значением от 0 до 3, если открытие было успешно, или 0FFH (255 в десятичной системе), если файл не был найден. Регистр Н устанавливается в ноль в обоих случаях. Если возникла физическая или расширенная ошибка, функция *open* выполняет различные действия в зависимости от режима ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS находится в режиме по умолчанию, сообщение идентифицирующее ошибку, выводится в консоль, и процесс завершается. В противном случае, функция *open* возвращает процессу вызова в регистра A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

04 : Ошибка выбора диска

05 : Файл открыт другим процессом или текущим процессом  
в несовместимом режиме

07 : Ошибка пароля файла

09 : Недопустимый ? в поле имени или расширения FCB

10 : Превышен лимит открытых файлов

11 : Нет места в системном списке блокировок

|  |
| --- |
| **Функция 16: Close File (Закрытие файла)** |
| Входные параметры:  Регистр C: 10H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код каталога  Регистр H: Физическая или расширенная ошибка |

Функция *Close File* выполняет инверсию функции *open file*. Процесс вызова передает адрес FCB в регистровой паре DE. Указанный FCB должен быть предварительно успешно активирован вызовом функции создания или открытия файла (см. функции 15 и 22). Атрибут интерфейса f5' определяет, как файл будет закрыт, как показано ниже:

f5' = 0 - Постоянное закрытие (режим по умолчанию)

f5' = 1 - Частичное закрытие

Функция закрытия сначала проверяет, что указанный FCB имеет допустимую контрольную сумму. Если контрольная сумма допустима и указанный FCB содержит новую информацию из-за операций записи в FCB, функция постоянного закрытия записывает новые данные в указанный каталог диска. Обратите внимание, что FCB не содержит новую информацию, и шаг обновления каталога пропускается, если были выполнены только операции чтения и/или обновления указанного FCB. Однако функция закрытия всегда пытается определить местоположение соответствующей записи FCB в каталоге и возвращает ошибку, если запись каталога не найдена.

Если функция закрытия успешно выполняет описанные выше действия, и если атрибут интерфейса f5' указывает, что закрытие является постоянным, функция закрытия удаляет элемент файла из системного списка блокировки. Если FCB был открыт в неблокируемом режиме, он также удаляет все элементы записей блокировки, принадлежащие файлу из системного списка блокировки. Поскольку элемент файла удален из списка блокировки, функция закрытия делает недействительной контрольную сумму FCB, чтобы гарантировать, что указанный FCB, впоследствии не сможет использоваться функциями BDOS, которым требуется открытый FCB, например, *Write Sequential* (последовательной записью).

Функция закрытия также создает метку даты и времени обновления для закрытого файла, если выполняются следующие условия: у указанного диска есть метка каталога, которая запрашивает метку даты и времени обновления, у указанного файла есть XFCB, указанный диск для чтения и была сделана операции записи в файл с момента открытия FCB. Ни один из этих шагов не выполнен для операции частичного закрытия (f5' = 1).

По возвращении, функция закрытия возвращает код каталога в регистре A со значением от 0 до 3, если завершение было успешно, или 0FFH (255 в десятичной системе), если файл не был найден. Регистр Н устанавливается в ноль в обоих случаях. Если возникла физическая или расширенная ошибка, функция закрытия выполняет различные действия в зависимости от режим ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS находится в режиме по умолчанию, сообщение идентифицирующее ошибку, выводится в консоль, и процесс завершается. В противном случае, функция закрытия возвращает процессу вызова в регистра A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

02 : Диск только для чтения

04 : Ошибка выбора диска

06 : Ошибка контрольной суммы FCB

|  |
| --- |
| **Функция 17: Search For First (Поиск первого)** |
| Входные параметры:  Регистр C: 11H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код каталога  Регистр H: Физическая ошибка |

Функция поиск первого сканирует каталог на соответствие с FCB указанным в регистровой паре DE. Могут быть выполнены два типа поиска. Для стандартного поиска процесс вызова инициализирует байты с 0 до 12 указанного FCB, байт 0 определяет каталог диска для поиска, байты 1 - 11 определяют файл или файлы, которые будут разыскиваться и байт 12 определяет экстент. Обычно байт 12 устанавливается в ноль. Вопросительный знак ASCII (63 в десятичной и 3F в шестнадцатеричной системе) в любом из байтов 1 - 12 соответствует всем записям в каталоге в соответствующей позиции. Это средство, названное *неоднозначной ссылкой*, может использоваться для поиска нескольких файлов в каталоге. При вызове в стандартном режиме, функция поиска сканирует до первой записи файла в заданном каталоге, совпадающей с FCB и принадлежащей номеру текущего пользователя.

Функция поиска также инициализирует функцию *Search For Next* (поиск следующего). После того, как функция поиска определила местоположение первой записи каталога, соответствующей указанному FCB, функция поиск следующего может вызываться несколько раз для поиска всех оставшихся совпадающих записей. Однако, с точки зрения последовательности выполнения, вызов поиск следующего должен следовать за вызовом поиск первого или поиск следующего без других промежуточных вызовов дисковых функций BDOS.

Если байт 0 из указанного FCB имеет значение вопросительного знака функция поиска, игнорирует остаток указанного FCB, и находит местоположение первой записи каталога, находящегося на текущем диске по умолчанию. Все остающиеся записи каталога могут быть найдены, делая несколько вызовов поиск следующего. Данный вид операций поиска обычно не производится прикладными программами, но он действительно обеспечивает полную гибкость для сканирования все значений текущего каталога. Обратите внимание, что этот тип операции поиска должен быть выполнен для получения доступ к метке каталога диска (см. [Раздел 2.2.5](#_Метки_каталога_и)).

По возвращении, функция поиска возвращает код каталога в регистре A со значением от 0 до 3, если поиск был успешен, или 0FFH (255 в десятичной системе), если соответствующая запись каталога не была найдена. Регистр H устанавливается равным нулю в обоих случаях. В случае успешного поиска, текущий DMA также заполнен записью каталога, содержащей совпадающую запись, и ее относительную начальную позицию A \* 32 (то есть сдвиг регистра A влево на 5 бит, или ADD A пять раз). Хотя прикладным программам обычно это не требуется, сведения о каталоге можно извлечь из буфера в этой позиции.

Если возникла физическая ошибка, функция поиска выполняет различные действия в зависимости от режима ошибок BDOS (См. Функцию 45). Если режим ошибок BDOS находится в режиме по умолчанию, сообщение идентифицирующее ошибку, выводится в консоль и процесс завершается. В противном случае, функция поиска возвращает процессу вызова в регистра A значение 0FFH и регистру H присваивается один из следующих кодов физической ошибки:

01 : Систематическая ошибка

04 : Ошибка выбора диска

|  |
| --- |
| **Функция 18: Search For Next (Поиск следующего)** |
| Входные параметры:  Регистр C: 12H  Возвращаемые значения:  Регистр A: Код каталога  Регистр H: Физическая ошибка |

Функция *Search For Next* (поиск следующего) идентична *Search For First* (поиск первого), за исключением того, что сканирование каталога продолжается от последней совпадающей записи. Функция 18 возвращает код каталога в регистре A, аналогично функции 17. Примечание: В последовательности выполнения, вызов функции 18 должен следовать или за вызовом функции 17 или за другой функции 18 без других промежуточных вызовов дисковых функций BDOS.

|  |
| --- |
| **Функция 19: Delete File (Удаление файла)** |
| Входные параметры:  Регистр C: 13H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код каталога  Регистр H: Физическая или расширенная ошибка |

Функция *Delete File* удаляет файлы и/или XFCB, которые соответствуют FCB, указанному в регистровой паре DE. Имя файла и тип могут содержать неоднозначные ссылки (т.е., вопросительные знаки в байтах от f1 до t3), но байт "dr" не может быть неоднозначным, как он допустим в функциях поиск и поиск следующего. Атрибут интерфейса f5' определяет тип выполняемой операции удаления, перечисленных ниже:

f5' = 0 – Стандартное удаление (режим по умолчанию)

f5' = 1 – Удаление только XFCB

Если какой-либо из файлов, определенных в указанном FCB, защищен паролем, корректный пароль должен быть помещен в первые восемь байтов текущего буфера DMA или ранее установлен в качестве пароля по умолчанию (см. Функцию 106).

Для стандартных операций удаления, функция удаления удаляет все записи каталога, принадлежащие файлам, которые соответствуют указанному FCB. Все дисковые каталоги и пространство данных, принадлежавшее удаленным файлам, возвращаются в свободное пространство и становятся доступными для выделения другим файлам. Каталоги XFCB, которые принадлежали удаленным файлам, также удаляются из каталога. Если атрибут интерфейса FCB f5' установлен в 1, функция 19 удаляет только соответствующий каталог XFCB указанного FCB. Примечание: Если какой-либо из файлов, соответствующих веденной спецификации FCB не проходит проверку пароля, является только для чтения или в настоящее время открыт другим процессом, то функция *Delete* не удаляет файлы или XFCB. Это относится к обоим типам операций удаления.

Процесс может удалить файл, который он в настоящее время имеет открытым, если файл был открыт в заблокированном режиме. Однако, возвращается ошибка контрольной суммы, если процесс в последующем указывает на файл с вызовом функции BDOS требующей открытого FCB. Файлы, открытые в режиме только для чтения или неблокируемом режиме, не могут быть удалены никаким процессом.

По возвращении, функция удаления возвращает код каталога в регистре A со значением от 0 до 3, если удаление было успешно, или 0FFH (255 в десятичной системе), если файл который определен с помощью FCB не найден. Регистр H устанавливается равным нулю в обоих случаях. Если возникла физическая или расширенная ошибка, функция удаления выполняет различные действия в зависимости от режима ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS является режимом по умолчанию, сообщение идентифицирующее ошибку, выводится в консоль, и процесс завершается. В противном случае, функция удаления возвращает процессу вызова в регистра A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

02 : Диск только для чтения

03 : Файл только для чтения

04 : Ошибка выбора диска

05 : Файл открыт другим процессом или текущим процессом  
в режиме только для чтения или неблокируемом режиме

07 : Ошибка пароля файла

|  |
| --- |
| **Функция 20: Read Sequential (Последовательное чтение)** |
| Входные параметры:  Регистр C: 14H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Физическая ошибка |

Функция *Read Sequential* (Последовательное чтение) читает от одной до шестнадцати 128-байтовых последовательных записей из файла в память, начиная с текущего адреса DMA. Многосекторный счетчик BDOS (см. Функцию 44) определяет число читаемых записей. Значение по умолчанию - одна запись. FCB, адресуемый регистровой парой DE, должен быть ранее активирован с помощью вызова функций открытия или создания файла.

Функция 20 читает каждую запись из байта "cr" в экстенте, затем автоматически увеличивает на единицу поле "cr" для следующей позиции записи. Если поле "cr" переполняется то, функция автоматически открывает следующий логический экстент и сбрасывает поле "cr" к 0 при подготовке к следующей операции чтения. Вызывающий процесс должен установить поле "cr" равным 0 после вызова открытия, если целью является последовательное чтение с начала файла.

По возвращении, функция последовательного чтения устанавливает регистр A равным 0, если операция чтения была успешна. В противном случае, регистр A содержит код ошибки, идентифицирующий ошибку как показано ниже:

01 : Чтение незаписанных данных (конец файла)

09 : Недопустимый FCB

10 : Ошибка контрольной суммы FCB

11 : Ошибка проверки неблокируемого файла

255 : Физическая ошибка. См. регистр H

Код ошибки 01 возвращается, если в следующей позиции записи файла данные не существуют. Как правило, ситуация с отсутствием данных встречается в конце файла. Однако, она может также произойти, если предпринята попытка, считать блок данных, который не был ранее записан или из несуществующего экстента. Эти ситуации обычно ограничиваются файлами, создаваемыми или добавленными с помощью функций BDOS произвольной записи (см. Функции 34 и 40).

Код ошибки 09 возвращается, если FCB был признан недействительным в предыдущем вызове BDOS случайного чтения или записи, вернувшего ошибку. Вызов произвольного чтения (функция 33) для существующей записи в файле, может быть сделан для повторной проверки FCB.

Код ошибки 10 возвращается, если указанный FCB не прошел тест контрольной суммы.

Код ошибки 11 возвращается, если BDOS не удается найти запись каталога FCB, при попытке проверить, что указанный FCB содержит текущую информацию. Эта ошибка возвращается только для файлов открытых в неблокируемом режиме.

Код ошибки 255 возвращается, если произошла физическая ошибка и режим ошибок BDOS является режимом возврата ошибок или режимом возврата и отображения ошибок (см. Функцию 45). Если режим ошибок - режим по умолчанию, сообщение идентифицирующее физическую ошибку выводится в консоль и процесс вызова завершается. Если вызывающему процессу возвращается физическая ошибка, она идентифицируется по четыре младшим битам регистра H, как показано ниже:

01 : Систематическая ошибка

04 : Ошибка выбора диска

Если многосекторный счетчик BDOS больше единицы, функция последовательного чтения устанавливает также четыре старших бита регистра H при возврате после любых ошибок. В этом случае четыре бита содержат целое число успешно прочитанных записей, прежде чем произошла ошибка. Это значение может изменяться от 0 до 15. Четыре бита старшего разряда регистра H всегда обнуляются, если многосекторный счетчик равен единице.

|  |
| --- |
| **Функция 21: Write Sequential (Последовательная запись)** |
| Входные параметры:  Регистр C: 15H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Физическая ошибка |

Функция *Write Sequential* (Последовательная запись) записывает от одной до шестнадцати 128-байтовых записей начиная с текущего адреса DMA в файл с именем в FCB, ссылка на который содержится в регистровой паре DE. Многосекторный счетчик BDOS (см. Функцию 44) определяет количество записываемых записей. Значение по умолчанию - одна запись. Адресуемый FCB, должен быть ранее активирован с помощью вызова функций открытия или создания файла.

Функция 21 помещает запись в файл в позицию обозначенную байтом "cr" в FCB, затем автоматически увеличивает на единицу поле "cr" для следующей позиции записи. Если поле "cr" переполняется, то функция автоматически открывает или создает следующий логический экстент и сбрасывает поле "cr" к 0 при подготовке к следующей операции записи. Если Функция 21 используется, для записи в существующий файл, то вновь записываемые данные накладываются на уже существующие в файле. Вызывающий процесс должен установить поле "cr" равным 0 после вызова открытия или создания, если целью является последовательная запись с начала файла.

По возвращении, функция последовательной записи устанавливает регистр A равным 0, если операция записи была успешна. Иначе, регистр A содержит код ошибки, идентифицирующий ошибку как показано ниже:

01 : Нет места в каталоге

02 : Не доступен блок данных

08 : Запись блокирована другим процессом

09 : Недопустимый FCB

10 : Ошибка контрольной суммы FCB

11 : Ошибка проверки неблокируемого файла

255 : Физическая ошибка. См. регистр H

Код ошибки 01 возвращается, если функция записи пытается создать новый экстент, который требует новую запись в каталоге, и не существует доступных записей в каталоге на выбранном диске.

Код ошибки 02 возвращается, когда команда записи пытается выделить новый блок данных файлу, а свободных блоков данных на выбранном диске не существует.

Код ошибки 08 возвращается, если функция записи пытается записать в запись, заблокированную другим процессом. Эта ошибка возвращается только для файлов, открытых в неблокируемом режиме.

Код ошибки 09 возвращается, если FCB был признан недействительным в предыдущем вызове BDOS случайного чтения или записи, вернувшего ошибку. Для повторной проверки FCB может быть сделан вызов произвольного чтения (функция 33) для существующей записи в файле.

Код ошибки 10 возвращается, если указанный FCB не прошел тест контрольной суммы FCB.

Код ошибки 11 возвращается, если BDOS не удается найти запись каталога FCB, при попытке проверить, что указанный FCB содержит текущую информацию. Эта ошибка возвращается только для файлов открытых в неблокируемом режиме.

Код ошибки 255 возвращается, если произошла физическая ошибка и режим ошибок BDOS является режимом возврата ошибок или режимом возврата и отображения ошибок (см. Функцию 45). Если режим ошибок - режим по умолчанию, сообщение идентифицирующее физическую ошибку выводится в консоль и процесс вызова завершается. Если вызывающему процессу возвращается физическая ошибка, она идентифицируется по четыре младшим битам регистра H, как показано ниже:

01 : Систематическая ошибка

02 : Диск только для чтения

03 : Файл только для чтения, или

Файл открыт в режиме только для чтения, или

Файл защищен паролем в режиме записи

04 : Ошибка выбора диска

Функция последовательной записи также устанавливает четыре старших бита регистра H при возврате после любых ошибок, если многосекторный счетчик BDOS больше единицы. В этом случае четыре бита содержат целое число успешно записанных записей, прежде чем произошла ошибка. Это значение может изменяться от 0 до 15. Четыре бита старшего разряда регистра H всегда обнуляются, если многосекторный счетчик равен единице.

|  |
| --- |
| **Функция 22: Make File (Создание файла)** |
| Входные параметры:  Регистр C: 16H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код каталога  Регистр H: Физическая или расширенная ошибка |

Функция *Make File* создает новую запись каталога для файла c текущим кодом пользователя. Она также создает XFCB для файла, если указанный диск имеет метку каталога, которая вызывает автоматическое создание XFCB. Процесс вызова передает адрес FCB в регистровой паре DE, с байтом 0 в FCB определяющим диск, байтами 1 - 11 определяющими имя и расширение файла и байтом 12 устанавливающим номер экстента. Обычно, байт 12 устанавливается равным 0. Байт 32 из FCB (поле "cr") должны быть установлен равным 0 (до или после вызова *Make File*), если намерение состоит в том, чтобы писать последовательно с начала файла.

Атрибут интерфейса f5' определяет режим открытия файл. Атрибут интерфейса f6' определяет, должен ли пароль быть присвоен создаваемому файлу. Атрибуты интерфейса приведены ниже:

f5' = 0 - Открытие в режиме блокировки (режим по умолчанию)

f5' = 1 - Открытие в неблокируемом режиме

f6' = 0 - Не назначать пароль (по умолчанию)

f5' = 1 - Присвоить пароль создаваемому файлу

Когда атрибут f6' установлен в 1, процесс вызова должен поместить пароль в первые 8 байтов текущего буфера DMA и установить байт 9 из буфера DMA в режим пароля (См. Функцию 102).

Функция *Make* возвращается с ошибкой, если указанный FCB “называет” файл существующий в каталоге под текущим кодом пользователя. Может быть сделана предшествующая операция удаления, если существует возможность дублирования.

Если операция создания выполнена успешно, она активирует указанный FCB для файловых операций (открывает FCB) и инициализирует элемент каталога и пустой файл для указанного FCB. Вычисляется контрольная сумма и назначается FCB. Функции BDOS, которые требуют открытого FCB, например *Write Random* (произвольной записи), проверяют правильность контрольной суммы FCB перед выполнением. Если режим открытия - неблокируемый, байты r0 и r1 устанавливаются в двух байтное значение, называемое идентификатором файла. Идентификатор файла является обязательным параметром для функций BDOS *Lock Record* (блокирование записи) и *Unlock Record* (разблокирование записи). Обратите внимание, что функция *Make* инициализирует все атрибуты файла значениями равными нулю.

Также файловая система BDOS для открытого файла создает элемент в системном списке блокировки, для записи о успешной операции создания файла. Пока существует этот элемент, ни один другой процесс не может удалять, переименовывать или изменять атрибуты файла.

Если указанный диск, содержит метку каталога, которая вызывает автоматическое создание XFCB, функция *Make* создает XFCB и создает метку с датой и временем создания для создаваемого файла. Обратите внимание, что создание метки времени не выполняется (поле XFCB метки о времени создания устанавливается равным нулю), если XFCB файла назначен вызовом BDOS *Write File XFCB* (Запись XFCB файла). Если атрибут интерфейса FCB f6' равняется 1, функция *Make* также назначает для файла пароль, передаваемый в DMA.

По возвращении, функция создания возвращает код каталога в регистре A значение от 0 до 3, если операция создания была успешна, или 0FFH (255 в десятичной системе), если пространство в каталоге не доступно. Регистр H устанавливается равным нулю в обоих случаях. Если возникла физическая или расширенная ошибка, функция создания выполняет различные действия в зависимости от режима ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS является режимом по умолчанию, сообщение идентифицирующее ошибку выводится в консоль, и процесс вызова завершается. В противном случае, функция удаления возвращает процессу вызова в регистре A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

02 : Диск только для чтения

04 : Ошибка выбора диска

08 : Файл уже существует

09 : ? в поле имени или расширения файла

10 : Превышен лимит открытых файлов

11 : Нет места в системном списке блокировок

|  |
| --- |
| **Функция 23: Rename File (Переименование файла)** |
| Входные параметры:  Регистр C: 17H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код возврата  Регистр H: Физическая или расширенная ошибка |

Функция *Rename File* (переименования) использует FCB, адресуемый регистровой парой DE, для изменения всех записей файла в каталоге, с именем файла определенным в первых 16 байтах FCB на имя файла во вторых 16 байтах. Если файл, определенный первым именем файла, защищен паролем, корректный пароль должен быть помещен в первые восемь байтов текущего буфера DMA или предварительно установлен в качестве пароля по умолчанию (см. Функцию 106). Процесс вызова должен также гарантировать, что имена файлов, определенные в FCB, допустимы и однозначны, и что файл с новым именем не существует на диске. Функция 23 использует значение "dr" в байте 0 из FCB для выбора диска. Номер диска в байте 16 из FCB игнорируется.

Процесс может переименовать файл, открытый им, если файл был открыт в заблокированном режиме. Однако, если функция BDOS процесса впоследствии ссылается на файл с требованием открыть FCB, возвращается ошибка контрольной суммы. Файл, открытый в режиме только для чтения или неблокируемом режиме, не может быть переименован никаким процессом.

По возвращении, функция переименования возвращает код каталога в регистре A со значением от 0 до 3, если переименование было успешно, или 0FFH (255 в десятичной системе), если файл с именем первого файла в FCB не был найден. Регистр H устанавливается равным нулю в обоих случаях. Если возникла физическая или расширенная ошибка, функция переименования выполняет различные действия в зависимости от режима ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS является режимом по умолчанию, сообщение идентифицирующее ошибку выводится в консоль, и процесс вызова завершается. В противном случае, функция переименования возвращает процессу вызова в регистре A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

02 : Диск только для чтения

03 : Файл только для чтения

04 : Ошибка выбора диска

05 : Файл открыт другим процессом

07 : Ошибка пароля файла

08 : Файл уже существует

09 : ? в поле имени или расширения файла

|  |
| --- |
| **Функция 24: Returns login vector (Получить вектор регистрации)** |
| Входные параметры:  Регистр C: 18H  Возвращаемые значения:  Регистр HL: Вектор регистрации |

Функция 24 возвращает вектор регистрации в регистровой паре HL. Вектор регистрации является 16-разрядным значением с младшим значимым битом регистра L, соответствующим диску A, и старшим битом в регистре H соответствующим шестнадцатому диску, обозначаемому P. Бит в состоянии "0" указывает, что диск не зарегистрирован, в то время как в "1" означает, что диск активен. Диск становится активным, либо с помощью явного вызова BDOS 14 *Select Disk* (выбор диска), или неявного выбора когда файловая операция BDOS определяет ненулевой байт "dr" в FCB. Функция 24 поддерживает совместимость с более ранними версиями, поскольку регистры A и L при возврате содержат одинаковые значения.

|  |
| --- |
| **Функция 25: Return current disk (Получить номер текущего диска)** |
| Входные параметры:  Регистр C: 19H  Возвращаемые значения:  Регистр A: Номер текущего диска |

Функция 25 возвращает в регистре A номер выбранного по умолчанию диска. Номера дисков от 0 до 15 соответствуют дискам от A до P.

|  |
| --- |
| **Функция 26: Set DMA address (Установить адрес** **DMA)** |
| Входные параметры:  Регистр C: 1AH  Возвращаемые значения:  Регистр DE: Адрес DMA |

"DMA" является сокращением для *прямого доступа к памяти*, которое часто используется в связи с дисковыми контроллерами, которые непосредственно получают доступ к памяти компьютера для передачи данных от и из дисковой подсистемы. В MP/M II, текущий DMA обычно определяется как буфер в памяти, где данные (запись) находятся перед записью на диск и после операции чтения с диска. Если многосекторный счетчик BDOS равен 1 (см. Функцию 44), размер буфера составляет 128 байтов. Однако, если многосекторный счетчик BDOS больше 1, размер буфера должен равняться N \* 128, где N равняется многосекторному счетчику.

Некоторые функции BDOS также используют текущий DMA, для передачи параметров и возвращаемых значений. Например, функции BDOS, которые проверяют и присваивают пароли файлу, требуют, чтобы пароль был помещен в текущий DMA. Как другой пример, Функция 46 (Получить свободное место на диске) возвращает свои результаты в первых 3 байтах текущего DMA. Если текущий DMA используется в данном контексте, размер буфера в памяти определяется конкретными требованиями вызываемой функции.

Когда транзитная программа инициируется CLI, ее адрес DMA установлен в BASE+0080H. Функция BDOS *Reset Disk System* (Функция 13) также устанавливает адрес DMA в BASE+0080H. Функция *Set DMA* (установки DMA) может изменить это значение по умолчанию на другой адрес памяти. Адрес DMA устанавливается равным значению, переданному в регистровой паре DE. Адрес DMA остается в этом значении, пока он не будет изменен другим вызовом *Set DMA* или *Reset Disk System*.

|  |
| --- |
| **Функция 27: Get Address** **Allocation (Получить адрес распределения)** |
| Входные параметры:  Регистр C: 1BH  Возвращаемые значения:  Регистр HL: Адрес распределения |

MP/M II поддерживает "вектор распределения" в основной памяти для каждого активного диска. Многие программы часто используют информацию, предоставленную вектором распределения, чтобы определить размер свободного места данных на диске. Отметим, однако, что информация о распределении может быть неточной, если диск был отмечен только для чтения.

Функция 27 возвращает в регистровой паре HL, начальный адрес вектора распределения для выбранного в настоящее время диска. Если происходит физическая ошибка, в режиме ошибок BDOS являющимся одним из режимов возврата (см. Функцию 45), функция 27 возвращает значение 0FFFFH в регистровой паре HL.

В системе MP/M II с коммутацией банков вектор распределения может быть помещен в нулевой банк. Это является опцией XIOS. В этом случае транзитная программа, загруженная в другой банк, не может получить доступ к вектору распределения. Однако, может использоваться функция BDOS *Get Disk Free Space* (Функция 46), чтобы напрямую вернуть число свободных 128-байтных записей на диске. Фактически, утилиты MP/M II, которые выводят на экран свободное место на диске (STAT, SDIR и SHOW) используют функцию 46 для этой цели.

|  |
| --- |
| **Функция 28: Write Protect Disk (Защита диска от записи)** |
| Входные параметры:  Регистр C: 1CH  Возвращаемые значения:  Регистр A: Код возврата |

Функция *Write Protect Disk* обеспечивает временную защиту выбранного в настоящее время диска от записи, отмечая диск как только для чтения. Никакой процесс не может производить запись на диск, который находится в состоянии только для чтения. Для возврата диска из состояния только для чтения в состояние чтение и запись должна быть успешно выполнена операция сброса диска (см. Функции 13 и 37).

В MP/M II функция *Write Protect Disk* является условной. Если у другого процесса есть открытый файл на диске, эта функция отклоняется и процессу вызова возвращается значение 0FFH. В противном случае, регистр A устанавливается равным 0. Обратите внимание, что диск в состоянии только для чтения не может быть сброшен процессом, если у другого процесса есть открытый файл на диске.

|  |
| --- |
| **Функция 29: Get read/only Vector (Получить вектор только для чтения)** |
| Входные параметры:  Регистр C: 1DH  Возвращаемые значения:  Регистр HL: Значение вектора только для чтения |

Функция 29 возвращает в регистровой паре HL битовый вектор, который показывает, какие диски имеют временное состояние только для чтения. Бит только для чтения, устанавливается с помощью вызова BDOS *Write Protect Disk* или автоматически программным обеспечением MP/M II, при обнаружении смены носителей дисков.

Формат битового вектора походит на формат вектора регистрации, возвращаемого функцией 24. Младший значащий бит соответствует диску A, в то время как старший значащий бит соответствует диску P.

|  |
| --- |
| **Функция 30: Set File Attributes (Установка атрибутов файла)** |
| Входные параметры:  Регистр C: 1EH  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код каталога |

Функция *Set File Attributes* - единственная функция BDOS, которая позволяет программе управлять атрибутами файла. Другие функции BDOS могут опросить эти атрибуты файла, но не могут изменить их. Атрибуты файла, которые могут быть установлены или сброшены функцией 30: от f1' до f4', только для чтения (t1'), системный (t2') и архив (t3'). Регистровая пара DE содержит адрес FCB с именем файла с установленными или сброшенными атрибутами. Процесс вызова должен гарантировать, что не определяет неоднозначное имя файла. Кроме того, если указанный файл защищен паролем, корректный пароль должен быть помещен в первые восемь байтов текущего буфера DMA или предварительно установлен пароль по умолчанию (см. Функцию 106).

Функция 30 ищет в каталоге указанном в FCB записи, принадлежащие текущему номеру пользователя, которые соответствуют указанным в FCB полям имени и расширения. Затем функция обновляет каталог для добавления выбранных признаков. Атрибуты файл t1', t2' и t3' определяются MP/M II. Они описаны в [Разделе 2.2.3](#_Определение_блока_управления). Атрибуты от f1' до f4' в настоящее время не используются, но могут быть полезны для прикладных программ, потому что они не участвуют в процессе сравнения, используемом BDOS во время операций *Open File* (открытия файла) и *Close File* (закрытия файла). Признаки от f5' до f8' зарезервированы для использования в качестве атрибутов интерфейса.

Эта функция не выполняется, если файл, определенный в указанном FCB в настоящее время открыт другим процессом. Однако, она выполняется, если указанный файл открыт вызывающим процессом в режиме блокировки. После успешной установки атрибутов файла, открытого вызывающим процессом, любая последующая ссылка на файл, требующая открытого FCB, возвращает ошибку контрольной суммы. Эта функция не устанавливает атрибуты файла, открытого в настоящее время только для чтения или неблокируемом режиме для любого процесса.

По возвращении, функция 30 возвращает код каталога в регистре A со значением от 0 до 3, если функция была успешна, или 0FFH (255 в десятичной системе), если файл, определенный в указанном FCB, не был найден. Регистр H устанавливается равным нулю в обоих случаях. Если возникла физическая или расширенная ошибка, функция *Set File Attributes* выполняет различные действия в зависимости от режима ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS является режимом по умолчанию, сообщение идентифицирующее ошибку выводится в консоль, и процесс вызова завершается. В противном случае, функция 30 возвращает процессу вызова в регистре A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

02 : Диск только для чтения

04 : Ошибка выбора диска

05 : Файл открыт другим процессом

07 : Ошибка пароля файла

09 : ? в поле имени или расширения файла

|  |
| --- |
| **Функция 31: Get Address DPB (Получить адрес блока параметров диска)** |
| Входные параметры:  Регистр C: 1FH  Возвращаемые значения:  Регистр HL: Адрес DPB |

Функция 31 возвращает в регистровой паре HL, адрес XIOS резидентного блока параметров диска (DPB) для текущего выбранного диска. (См. *Системное руководство MP/M II* для получения информации о формате DPB). Вызывающий процесс может использовать этот адрес для извлечения значений параметров диска для отображения или вычисления места на диске.

Если возникает физическая ошибка, когда режим ошибок BDOS является одним из режимов возврата (см. Функцию 45), Функция 31 возвращает значение 0FFFFH в регистровой паре HL.

|  |
| --- |
| **Функция 32: Set/Get User Code (Установить/Получить номер пользователя)** |
| Входные параметры:  Регистр C: 20H  Регистр E: 0FFH (получить) или номер пользователя (при установке)  Возвращаемые значения:  Регистр A: Текущий номер или (нет значения) |

Процесс может изменить или опросить номер активного в настоящее время пользователя, вызвав функцию 32. Если регистр E = 0FFH, то значение номера текущего пользователя возвращается в регистре A и может находится в диапазоне от 0 до 15. Если регистр E не содержит 0FFH, то номер текущего пользователя изменяется на значение из регистра E (по модулю 16).

|  |
| --- |
| **Функция 33: Read Random (Произвольное чтение)** |
| Входные параметры:  Регистр C: 21H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Физическая ошибка |

Функция *Read Random* (произвольного чтения) подобна функции последовательного чтения за исключением того, что операция чтения происходит по конкретному номеру произвольной записи, выбранному 24-разрядному значению, созданному из поля из трех байт (r0, r1, r2), начинающегося в позиции 33 в FCB. Обратите внимание, что последовательность 24 битов сохранена в первом младшем значащим байте (r0), следующем среднем байте (r1) и последнем старшем байте (r2). Номер произвольной записи может изменяться от 0 до 242 143. Это соответствует максимальному значению 3 в байте r2.

Для чтения файла с помощью функции 33, процесс вызова должен сначала открыть начальный экстент (экстент 0). Это гарантирует, что FCB правильно инициализирован для последующих операций произвольного доступа. (Начальный экстент может содержать или не содержать любые распределенные данные). Функция 33 помещает указанный номер записи в поле произвольной записи, и затем BDOS читает запись по текущему адресу DMA. Функция автоматически устанавливает логический экстент и текущие значения записи, но в отличие от функции последовательного чтения, она не увеличивает номер записи. Таким образом, последующий вызов произвольного чтение перечитывает ту же запись. После операции произвольного чтения к файлу можно получить доступ последовательно, начиная с текущей позиции, произвольного доступа. Однако, последняя запись произвольного доступа перечитывается или перезаписывается при переключении от произвольного к последовательному режиму.

Если многосекторный счетчик BDOS больше, чем 1 (см. Функцию 44), функция произвольного чтения читает несколько последовательных записей в память, начинающуюся с текущего DMA. Поля r0, r1, и r2 в FCB автоматически увеличиваются на 1, чтобы считать каждую запись. Однако, номер произвольной записи FCB восстанавливает значение первой записи при возвращении к процессу вызова. По возвращении, функция произвольного чтения обнуляет регистр A, если операция чтения была успешна. В противном случае, регистр A содержит один из следующих кодов ошибки:

01 : Чтение незаписанных данных

03 : Нет возможности закрыть текущий экстент

04 : Поиск в несуществующем экстенте

06 : Номер произвольной записи вне диапазона

10 : Ошибка контрольной суммы FCB

11 : Ошибка проверки неблокированного файла

255 : Физическая ошибка. См. регистр H

Код ошибки 01 возвращается, если функция произвольного чтения пытается получить доступ к блоку данных, который не был ранее записан.

Код ошибки 03 возвращается, если функция произвольного чтения не может закрыть текущий экстент до перехода к новому экстенту.

Код ошибки 04 возвращается, если операция произвольного чтение получает доступ к несуществующему экстенту.

Код ошибки 06 возвращается, если байт 35 (r2) указанного FCB больше, чем 3.

Код ошибки 10 возвращается, если указанный FCB не прошел тест контрольной суммы.

Код ошибки 11 возвращается, если BDOS не удается найти запись каталога FCB, при попытке проверить, что указанный FCB содержит текущую информацию. Эта ошибка возвращается только для файлов открытых в неблокируемом режиме.

Код ошибки 255 возвращается, если произошла физическая ошибка и режим ошибок BDOS является режимом возврата ошибок или режимом возврата и отображения ошибок (см. Функцию 45). Если режим ошибок - режим по умолчанию, сообщение идентифицирующее физическую ошибку выводится в консоль и процесс вызова завершается. Если вызывающему процессу возвращается физическая ошибка, она идентифицируется по четыре младшим битам регистра H, как показано ниже:

01 : Систематическая ошибка

04 : Ошибка выбора диска

Если многосекторный счетчик BDOS больше единицы, функция произвольного чтения устанавливает также четыре старших бита регистра H при возврате после любых ошибок. В этом случае четыре бита содержат целое число успешно прочитанных записей, прежде чем произошла ошибка. Это значение может изменяться от 0 до 15. Четыре бита старшего разряда регистра H всегда обнуляются, если многосекторный счетчик равен единице.

|  |
| --- |
| **Функция 34: Write Random (Произвольная запись)** |
| Входные параметры:  Регистр C: 22H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Физическая ошибка |

Функция произвольной записи походит на функцию произвольного чтения, за исключением того, что данные записываются на диск из текущего адреса DMA. Если экстент и/или блок данных на диске, куда данные должны быть записаны, еще не выделены, BDOS автоматически выполняет выделение, перед продолжением операции записи.

Чтобы записать в файл, используя функцию произвольной записи, процесс вызова должен сначала открыть начальный экстент (экстент 0). Это гарантирует, что FCB правильно инициализирован для последующих операций произвольного доступа. Начальный экстент может содержать или не содержать любые распределенные данные, но открытие экстента 0 создает запись о файле в каталоге так, что он может отображаться с помощью утилиты DIR. Если процесс не откроет экстент 0 и распределит данные в какой-то другой экстент, то файл будет невидим для утилиты DIR.

Функция произвольной записи устанавливает логический экстент и текущую позицию записи в соответствии с началом произвольной записи, но не изменяет номер произвольной записи. Таким образом операции последовательного чтения или записи могут следовать за произвольной записью, перечитывая или перезаписывая текущую запись, если вызывающий процесс переключается из произвольного в последовательный режим.

Если многосекторный счетчик BDOS больше, чем 1 (см. Функцию 44), функция произвольной записи записывает несколько последовательных записей из памяти, начинающейся с текущего DMA. Поля r0, r1, и r2 в FCB автоматически увеличиваются на 1, чтобы записать каждую запись. Однако, номер произвольной записи FCB восстанавливает значение первой записи при возвращении к процессу вызова. По возвращении, функция произвольной записи обнуляет регистр A, если операция чтения была успешна. В противном случае, регистр A содержит один из следующих кодов ошибки:

02 : Не доступен блок данных

03 : Нет возможности закрыть текущий экстент

05 : Нет места в каталоге

06 : Номер произвольной записи вне диапазона

08 : Запись блокирована другим процессом

10 : Ошибка контрольной суммы FCB

11 : Ошибка проверки неблокированного файла

255 : Физическая ошибка. См. регистр H

Код ошибки 02 возвращается, если команда записи пытается выделить новый блок данных файлу, а на выбранном дисководе отсутствуют свободные блоки данных.

Код ошибки 03 возвращается, если функция произвольного чтения не может закрыть текущий экстент до перехода к новому экстенту.

Код ошибки 05 возвращается, если функция записи пытается создать новый экстент, который требует новой записи каталога, а в каталоге на выбранном дисководе свободные записи отсутствуют.

Код ошибки 06 возвращается, если байт 35 (r2) указанного FCB больше, чем 3.

Код ошибки 08 возвращается, если функция произвольной записи пытается записать в запись, заблокированную другим процессом. Эта ошибка возвращается только для файлов, открытых в неблокируемом режиме.

Код ошибки 10 возвращается, если указанный FCB не прошел тест контрольной суммы.

Код ошибки 11 возвращается, если BDOS не удается найти запись каталога FCB, при попытке проверить, что указанный FCB содержит текущую информацию. Эта ошибка возвращается только для файлов открытых в неблокируемом режиме.

Код ошибки 255 возвращается, если произошла физическая ошибка и режим ошибок BDOS является режимом возврата ошибок или режимом возврата и отображения ошибок (см. Функцию 45). Если режим ошибок - режим по умолчанию, сообщение идентифицирующее физическую ошибку выводится в консоль и процесс вызова завершается. Если вызывающему процессу возвращается физическая ошибка, она идентифицируется по четыре младшим битам регистра H, как показано ниже:

01 : Систематическая ошибка

02 : Диск только для чтения

03 : Файл только для чтения, или

Файл открыт в режиме только для чтения, или

Файл защищен паролем в режиме записи

04 : Ошибка выбора диска

Если многосекторный счетчик BDOS больше единицы, функция произвольного записи устанавливает также четыре старших бита регистра H при возврате после любых ошибок. В этом случае четыре бита содержат целое число успешно записанных записей, прежде чем произошла ошибка. Это значение может изменяться от 0 до 15. Четыре бита старшего разряда регистра H всегда обнуляются, если многосекторный счетчик равен единице.

|  |
| --- |
| **Функция 35: Compute File Size (Вычислить размер файла)** |
| Входные параметры:  Регистр C: 23H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Физическая или расширенная ошибка  Установленное поле произвольной записи |

Функция вычисления размера файла определяет "виртуальный" размер файла, который в действительности является адресом записи расположенной сразу после конца файла. "Виртуальный" размер файла соответствует физическому размеру, если файл записан последовательно. Если файл записан в произвольном режиме, могут существовать разрывы в распределении, и этот файл может содержать меньше записей, чем указанный размер. Например, если единственная запись с номером записи 262 143 (максимальная в MP/M II) записана в файл, используя функцию произвольной записи, то "виртуальный" размер файла составляет 262 144 записей несмотря на то, что выделен фактически только 1 блок данных.

Чтобы вычислить размер файла, процесс вызова передает в регистровой паре DE, адрес FCB в формате произвольного режима (байты r0, r1 и r2 присутствуют). Обратите внимание, что FCB должен содержать однозначное имя и расширение файла. Функция 35 присваивает полю произвольной записи FCB значение номера произвольной записи + 1 последней записи в файле. Если байт r2 установлен в 04, то файл содержит максимальное количество записей 262 144.

Процесс может добавлять данные в конец существующего файла с помощью вызова функции 35, установив позицию произвольной записи в конец файла, а затем выполняя последовательность операций произвольной записи, начиная с адреса заданной записи.

Примечание: BDOS не требует, чтобы файл был открыт, перед использованием Функции 35.

По возвращении, функция 35 возвращает 0 в регистре A, если файл, определенный в указанном FCB, был найден, или 0FFH в регистре A, если файл не был найден. Регистр H обнуляется в обоих случаях. Если возникла физическая или расширенная ошибка, функция 35 выполняет различные действия в зависимости от режима ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS является режим по умолчанию, сообщение, идентифицирующее ошибку, выводится в консоль и процесс завершается. В противном случае, функция 35 возвращает процессу вызова в регистре A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

04 : Ошибка выбора диска

09 : Недопустимый ? в поле имени или расширения FCB

|  |
| --- |
| **Функция 36: Set Random Record (Установить номер произвольной записи)** |
| Входные параметры:  Регистр C: 24H  Регистр DE: Адрес FCB  Возвращаемые значения:  Установленное поле произвольной записи |

Функция *Set Random Record* возвращает номер произвольной записи следующей записи для доступа к файлу, который был последовательно считан или записан в определенной точке. Она возвращает значение в поле произвольной записи (байты r0, r1 и r2) FCB, адресуемые регистровой парой DE. Функцию 36 можно использовать двумя способами.

Во-первых, часто необходимо первоначально считать и отсканировать последовательный файл для извлечения позиций различных "ключевых" полей. Для каждого встречающегося ключа вызывается функция 36 для вычисления позиции произвольной записи для данных, соответствующих этому ключу. Если размер блока данных составляет 128 байтов, полученный номер записи минус 1 помещается в таблицу с ключом для последующего извлечения. После сканирования всего файла и сведения в таблицу ключей и номеров их записей, можно переместиться непосредственно в определенную запись, выполнив произвольное чтение, используя соответствующий номер произвольной записи, который был сохранен ранее. Схема легко обобщается при использовании записей переменной длины, так как программе нужно хранить только относительную позицию байта в буфере вместе с ключом и номером записи, чтобы найти точную начальную позицию данных по ключу в дальнейшем.

Второе использование функции 36 возникает при переключении с последовательного чтения или записи на произвольное чтение или запись. После ряда операций последовательного доступа к файлу в определенной точке, вызывается функция 36, которая определяет номер записи и последующие операции произвольного чтения и записи продолжаются со следующей записи в файле.

|  |
| --- |
| **Функция 37: Reset Drive (Сброс диска)** |
| Входные параметры:  Регистр C: 25H  Регистр DE: Вектор диска  Возвращаемые значения:  Регистр A: Код возврата |

Функция *Reset Drive* используется для восстановления выбранных дисков в сброшенное состояние (сброшенные диски являются незарегистрированными и находятся в состоянии чтения-записи). Передаваемый параметр в регистровой паре DE является 16 битовым вектором сбрасываемых дисков, в котором младший значащий бит соответствует первому диску A, а старший бит, соответствует шестнадцатому диску, обозначенному P. Значения бита "1" означают, что указанный диск должен быть сброшен.

Эта функция в MP/M II является условной. Если у другого процесса есть файл, открытый на диске, который предполагается сбросить, и диск - съемный или только для чтения, функция сброса диска отклоняется и диски не сбрасываются.

По возвращении, если операция сброса успешна, регистр A устанавливается равным 0. В противном случае, регистр A устанавливается в значение 0FFH (255 в десятичной системе). Если режим ошибок BDOS находится не в режиме возврата ошибок (см. Функцию 45), то выводится в консоль сообщение, идентифицирующее процесс владеющий открытым файлом.

|  |
| --- |
| **Функция 38: Access Drive (Доступ к диску)** |
| Входные параметры:  Регистр C: 26H  Регистр DE: Вектор диска  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Расширенная ошибка |

Функция *Access Drive* вставляет специальный элемент открытый файл в системный список блокировки для каждого указанного диска. Пока существует элемент в списке блокировки, диск не может быть сброшен другим процессом. Как и в функции 37, вызывающий процесс передает вектор диска в регистровой паре DE. Формат вектора диска является таким же, как используется функции 37.

Функция *Access Drive* не вставляет элементы, если недостаточно свободного места в списке блокировки для поддержки всех новых элементов, или если число вставляемых элементов в список блокировки превышает максимально возможное число открытых файлов. Этот максимум является параметром MP/M II GENSYS. Если режим ошибок BDOS является режимом по умолчанию (см. Функцию 45), сообщение об ошибке выводится в консоль, и процесс вызова завершается. В противном случае, функция *Access Drive* возвращает процессу вызова в регистра A значение в 0FFH, и регистр H устанавливается в одно из следующих значений.

10 : Превышен лимит открытых файлов

11 : Нет места в системном списке блокировок

Регистр A устанавливается в ноль, если функция *Access Drive* успешна.

|  |
| --- |
| **Функция 38: Free Drive (Освобождение диска)** |
| Входные параметры:  Регистр C: 27H  Регистр DE: Вектор диска |

Функция *Free Drive* производит чистку элементов списка блокировки всех открытых файлов на указанных дисках и заблокированных записей принадлежащих процессу вызова. Как и в функция 38, процесс вызова передает вектор диска в регистровой паре DE.

Функция 39 не закрывает файлы, связанные с очищенными элементами открытых файлов из списка блокировок. Кроме того, если процесс осуществляет вызов функции BDOS ссылающейся на «очищенные» файлы, требующие открытого FCB, возвращается ошибка контрольной суммы. Файлы с выполненными операциями записи должны быть закрыты перед вызовом *Free Drive* для файлов на диске. В противном случае данные могут быть потеряны.

|  |
| --- |
| **Функция 40: Write Random With Zero**  **(Произвольная запись с заполнением нулями)** |
| Входные параметры:  Регистр C: 28H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Физическая ошибка |

Функция *Write Random With Zero*, подобна функции *Write Random* (Функция 34) за исключением того, что блок ранее нераспределенных данных заполняется нулями до записи. Если эта функция использовалась для создания файла, записи заполненные нулями, к которым получают доступ с помощью операции произвольного чтения, определяют незаписанные номера произвольной записи. Незаписанные номера произвольной записи в выделенных блоках данных файлов созданных с использованием функции произвольной записи содержат неинициализированные данные.

|  |
| --- |
| **Функция 41: Test and Write Record (Тестирование и запись)** |
| Входные параметры:  Регистр C: 29H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Физическая ошибка |

Функция *Test and Write Record* предоставляет средство проверки текущего содержимого записи на диске перед его обновлением. Вызывающий процесс должен установить байты номера тестируемой произвольной записи R0, R1 и R2 в FCB адресуемом регистровой парой DE. Оригинальная версия записи (т.е. тестируемая запись) должна находиться по текущему адресу DMA, и следующей за ней новой версией записи. Размер записи может изменяться от 128 байт до шестнадцати раз этого значения в зависимости от многосекторного счетчика BDOS (см Функцию 44).

Функция 41 проверяет, что первая запись идентична записи на диске прежде, чем заменить ее новой версией записи. Если запись на диске не соответствует, запись на диске не изменяется, и вызывающему процессу возвращается код ошибки.

Функция тестирования и записи предназначена для использования в ситуациях, когда больше чем у одного процесса есть доступ для чтения-записи к общему файлу. Эта ситуация поддерживается в МР/М II, если несколько процессов открывают один и тот же файл в неблокированном режиме. Функция 41 является логической заменой для последовательности операций блокирования/разблокирования записи, поскольку она предотвращает одновременное обновление той же записи двумя процессами. Обратите внимание, что эта функция также поддерживается для файлов, открытых в блокируемом режиме, чтобы обеспечить совместимость между MP/M II и CP/M.

По возвращении, если функция *Test and Write Random* успешна, регистр A устанавливается равным 0. В противном случае, регистр A содержит один из следующих кодов ошибки:

01 : Чтение незаписанных данных

03 : Нет возможности закрыть текущий экстент

04 : Поиск в несуществующем экстенте

06 : Номер произвольной записи вне диапазона

07 : Несоответствие записи

08 : Запись блокирована другим процессом

10 : Ошибка контрольной суммы FCB

11 : Ошибка проверки неблокированного файла

255 : Физическая ошибка. См. регистр H

Код ошибки 01 возвращается, если функция тестирования и записи получает доступ к блоку данных, который не был ранее записан.

Код ошибки 03 возвращается, если функция тестирования и записи не может закрыть текущий экстент до перехода к новому экстенту.

Код ошибки 04 возвращается, если операция чтения получает доступ к несуществующему экстенту.

Код ошибки 06 возвращается, если байт 35 (r2) указанного FCB больше, чем 3.

Код ошибки 07 возвращается, если тестирование и запись не пройдено.

Код ошибки 08 возвращается, если указанная запись заблокирована другим процессом. Эта ошибка возвращается только для файлов, открытых в неблокируемом режиме.

Код ошибки 10 возвращается, если указанный FCB не прошел проверку контрольной суммы.

Код ошибки 11 возвращается, если BDOS не удается найти запись каталога FCB, при попытке проверить, что указанный FCB содержит текущую информацию. Эта ошибка возвращается только для файлов открытых в неблокируемом режиме.

Код ошибки 255 возвращается, если произошла физическая ошибка и режим ошибок BDOS является режимом возврата ошибок или режимом возврата и отображения ошибок (см. Функцию 45). Если режим ошибок - режим по умолчанию, сообщение идентифицирующее физическую ошибку выводится в консоль и процесс вызова завершается. Если вызывающему процессу возвращается физическая ошибка, она идентифицируется по четырем младшим битам регистра H, как показано ниже:

01 : Систематическая ошибка

02 : Диск только для чтения

03 : Файл только для чтения, или

Файл открыт в режиме только для чтения, или

Файл защищен паролем в режиме записи

04 : Ошибка выбора диска

Если многосекторный счетчик BDOS больше единицы, функция тестирования и записи устанавливает также четыре старших бита регистра H при возврате после любых ошибок. В этом случае четыре бита содержат целое число успешно записанных записей, прежде чем произошла ошибка. Это значение может изменяться от 0 до 15. Четыре бита старшего разряда регистра H всегда обнуляются, если многосекторный счетчик равен единице.

|  |
| --- |
| **Функция 42: Lock Record (Блокировка записи)** |
| Входные параметры:  Регистр C: 2AH  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Физическая ошибка |

Функция *Lock Record* блокирует одну или несколько последовательных записей так, чтобы никакая другая программа имеющая доступ к записям не могла одновременно заблокировать или обновить их. Эта функция поддерживается только для файлов, открытых в неблокируемом режиме. Если она вызывают для файла, открытого в режиме блокировки или режиме только для чтения, действие блокировки не выполняется и возвращается успешный результат. Это сделано для обеспечения совместимости между MP/M II и CP/M.

Процесс вызова передает в регистровой паре DE, адрес FCB в котором поле произвольной записи заполнено номером произвольной записи первой записи, которая будет заблокирована. Число записей, которые будут заблокированы, определяется многосекторным счетчиком BDOS (см. Функцию 44). Текущий DMA должен содержать 2-байтовый идентификатор файла, возвращенный функцией *Open File*, когда указанный FCB был открыт. Обратите внимание, что идентификатор файла возвращается функцией открытия, только при неблокируемом режиме открытия.

Функция *Lock Record* требует, чтобы каждый номер записи, который будет заблокирован, находился в блоке выделенном файлу. Кроме того, функция 42 проверяет, что ни одна из записей, которые будут заблокированы, в настоящее время не заблокирована другим процессом. Эти две проверки выполняются до блокировки любых записей.

Параметр генерации системы MP/M II определяет максимальное количество записей, которые могут быть заблокированы одним процессом. Каждая заблокированная запись использует запись в системной таблице блокировки BDOS, которая совместно используется заблокированными записями и записями открытых файлов. Другой параметр генерации системы MP/M II устанавливает размер этой таблицы. Если в системной таблице блокировки не хватает места, чтобы заблокировать все указанные записи, или процессом превышен лимит блокировок записей, то функция *Lock Record* не блокирует записи и возвращает код ошибки вызывающему процессу.

По возвращении, функция *Lock Record* устанавливает регистр A равным 0, если операция блокировки была успешной. В противном случае, регистр A содержит один из следующих кодов ошибки:

01 : Чтение незаписанных данных

03 : Нет возможности закрыть текущий экстент

04 : Поиск в несуществующем экстенте

06 : Номер произвольной записи вне диапазона

08 : Запись блокирована другим процессом

10 : Ошибка контрольной суммы FCB

11 : Ошибка проверки неблокированного файла

12 : Процесс превысил лимит блокировок записей

13 : Недопустимый идентификатор файла

14 : Нет места в системном списке блокировок

255 : Физическая ошибка. См. регистр H

Код ошибки 01 возвращается, если функция *Lock Record* получает доступ к блоку данных, который не был ранее записан.

Код ошибки 03 возвращается, если функция *Lock Record* не может закрыть текущий экстент до перехода к новому экстенту.

Код ошибки 04 возвращается, если функция *Lock Record* получает доступ к несуществующему экстенту.

Код ошибки 06 возвращается, если байт 35 (r2) указанного FCB больше, чем 3.

Код ошибки 08 возвращается, если указанная запись заблокирована другим процессом.

Код ошибки 10 возвращается, если указанный FCB не прошел проверку контрольной суммы.

Код ошибки 11 возвращается, если BDOS не удается найти запись каталога FCB, при попытке проверить, что указанный FCB содержит текущую информацию.

Код ошибки 12 возвращается, если сумма количества записей, заблокированных в настоящее время процессом вызова и количества записей, заблокированных вызовом *Lock Record*, превышает максимальное допустимое значение. Это значение является параметром GENSYS MP/M II.

Код ошибки 13 возвращается, если в текущий DMA помещен недопустимый идентификатор файла.

Код ошибки 255 возвращается, если произошла физическая ошибка и режим ошибок BDOS является один из режимов возврата (см. Функцию 45). Если режим ошибок - режим по умолчанию, сообщение идентифицирующее физическую ошибку выводится в консоль и процесс вызова завершается. Если вызывающему процессу возвращается физическая ошибка, она идентифицируется по четырем младшим битам регистра H, как показано ниже:

01 : Систематическая ошибка

04 : Ошибка выбора диска

Если многосекторный счетчик BDOS больше единицы, функция *Lock Record* устанавливает также четыре старших бита регистра H при возврате после любых ошибок. В этом случае четыре бита содержат целое число успешно заблокированных записей, прежде чем произошла ошибка. Это значение может изменяться от 0 до 15. Четыре бита старшего разряда регистра H всегда обнуляются, если многосекторный счетчик равен единице.

|  |
| --- |
| **Функция 43: Unlock Record (Разблокирование записи)** |
| Входные параметры:  Регистр C: 2BH  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код ошибки  Регистр H: Физическая ошибка |

Функция *Unlock Record* разблокирует одну или несколько последовательных записей, ранее заблокированных функцией *Lock Record*. Эта функция поддерживается только для файлов, открытых в неблокируемом режиме. Если она вызывается для файла, открытого в режиме блокировок или режиме только для чтения действие блокировки не выполняется и возвращается успешный результат. Это сделано для обеспечения совместимости между MP/M II и CP/M.

Процесс вызова передает в регистровой паре DE, адрес FCB, в котором поле произвольной записи заполнено номером произвольной записи первой записи, которая будет разблокирована. Число записей, которые будут разблокированы, определяется многосекторным счетчиком BDOS (см. Функцию 44). Текущий DMA должен содержать 2-байтовый идентификатор файла, возвращенный функцией *Open File*, когда указанный FCB был открыт. Обратите внимание, что идентификатор файла возвращается функцией открытия, только при неблокируемом режиме открытия.

Функция *Unlock Record* не разблокирует запись, которая в настоящее время заблокирована другим процессом. Однако, если процесс пытается сделать это ошибка не возвращается. Таким образом, если многосекторный счетчик больше 1, функция *Unlock Record* разблокирует все записи, заблокированные процессом вызова пропуская записи заблокированные другими процессами.

По возвращении, функция *Unlock Record* устанавливает регистр A равным 0, если операция разблокирования была успешной. В противном случае, регистр A содержит один из следующих кодов ошибки:

01 : Чтение незаписанных данных

03 : Нет возможности закрыть текущий экстент

04 : Поиск в несуществующем экстенте

06 : Номер произвольной записи вне диапазона

10 : Ошибка контрольной суммы FCB

11 : Ошибка проверки неблокированного файла

13 : Недопустимый идентификатор файла

255 : Физическая ошибка. См. регистр H

Код ошибки 01 возвращается, если функция *Unlock Record* получает доступ к блоку данных, который не был ранее записан.

Код ошибки 03 возвращается, если функция *Unlock Record* не может закрыть текущий экстент до перехода к новому экстенту.

Код ошибки 04 возвращается, если функция *Unlock Record* получает доступ к несуществующему экстенту.

Код ошибки 06 возвращается, если байт 35 (r2) указанного FCB больше, чем 3.

Код ошибки 10 возвращается, если указанный FCB не прошел проверку контрольной суммы.

Код ошибки 11 возвращается, если BDOS не удается найти запись каталога FCB, при попытке проверить, что указанный FCB содержит текущую информацию.

Код ошибки 13 возвращается, если в текущий DMA помещен недопустимый идентификатор файла.

Код ошибки 255 возвращается, если произошла физическая ошибка и режим ошибок BDOS является один из режимов возврата (см. Функцию 45). Если режим ошибок - режим по умолчанию, сообщение идентифицирующее физическую ошибку выводится в консоль и процесс вызова завершается. Если вызывающему процессу возвращается физическая ошибка, она идентифицируется по четырем младшим битам регистра H, как показано ниже:

01 : Систематическая ошибка

04 : Ошибка выбора диска

Если многосекторный счетчик BDOS больше единицы, функция *Unlock Record* устанавливает также четыре старших бита регистра H при возврате после любых ошибок. В этом случае четыре бита содержат целое число успешно разблокированных записей, прежде чем произошла ошибка. Это значение может изменяться от 0 до 15. Четыре бита старшего разряда регистра H всегда обнуляются, если многосекторный счетчик равен единице.

|  |
| --- |
| **Функция 44: Set Multi-Sector Count (Установка многосекторного счетчика**) |
| Входные параметры:  Регистр C: 2CH  Регистр E: Число секторов  Возвращаемые значения:  Регистр A: Код ошибки |

Функция *Set Multi-Sector Count* обеспечивает в MP/M II блокирование логических записей. Она позволяет процессу читать и записать от 1 до 16 "физических" 128-байтных записей за один раз во время последующих функций BDOS чтения и записи. Она также определяет количество блокируемых и разблокируемых 128-байтовых записей функциями BDOS блокировки и разблокировки.

Функция 44 устанавливает значение многосекторного счетчика для процесса вызова равным значению передаваемому в регистре E. После установки, указанный многосекторный счетчик остается в силе, пока вызывающий процесс не делает другой вызов функции *Set Multi-Sector Count* и изменяет значение. Обратите внимание, что интерпретатор командной строки (CLI) устанавливает многосекторный счетчик в 1, когда он инициирует транзитную программу.

Многосекторный счетчик оказывает влияние на сообщение об ошибке BDOS для функций BDOS чтения, записи, блокирования и разблокирования. Если ошибка прерывает эти функции, когда многосекторный счетчик больше 1, они возвращают количество успешно обработанных записей в четырех старших битах регистра H.

По возвращении, регистр A устанавливается равным 0, если указанное значение находится в диапазоне от 1 до 16. В противном случае, регистр A устанавливается равным 0FFH.

|  |
| --- |
| **Функция 45: Set BDOS Error Mode (Установка режима ошибок)** |
| Входные параметры:  Регистр C: 2DH  Регистр E: Режим ошибокBDOS  Возвращаемые значения:  Нет |

Функция *Set BDOS Error Mode* определяет, как физические и расширенные ошибки (см. Раздел 2.2.13) обрабатываются процессом. Режим ошибок может находиться в трех режимах: режиме по умолчанию, режиме возврата ошибок и режиме возврата и отображения ошибок. В режиме по умолчанию BDOS выводит системное сообщение, идентифицирующей ошибку на консоль и завершает процесс вызова. В режимах возврата, BDOS устанавливает в регистре A значение 0FFH (255 в десятичной системе), помещает код ошибки, идентифицирующий физическую или расширенную ошибку в четырех младших битах регистра H, и возвращается к процессу вызова. В режиме возврата и отображения, BDOS выводит на экран системное сообщение прежде, чем возвращается к процессу вызова. Однако, системные сообщения не выводятся на экран, когда BDOS находится в режиме возврата ошибок.

Функция 45 устанавливает режим ошибок BDOS для вызывающего процесса в режим, указанный в регистре E. Если регистр E установлен в 0FFH (255 в десятичной системе), режим ошибок устанавливается в режим возврата ошибок. Если регистр E установлен в 0FEH (254 в десятичной системе) режим ошибок устанавливается в режим возврата и отображения. Если регистр E установлен в какое-либо другое значение, режим ошибок устанавливается в режим по умолчанию.

|  |
| --- |
| **Функция 46: Get Disk Free Space (Получить свободное место на диске)** |
| Входные параметры:  Регистр C: 2EH  Регистр E: Диск  Возвращаемые значения:  Первые 3 байта текущего буфера DMA  Регистр A: Флаг ошибки  Регистр H: Физическая ошибка |

Функция *Get Disk Free Space* определяет число свободных секторов (128-байтовые записей) на указанном диске. Процесс вызова передает номер диска в регистре E, 0 для диска A, 1 для B, и т.д., до 15 для диска P в системах с 16 дисками. Функция 46 возвращает двоичное число в первых 3 байтах текущего буфера DMA.

Это число возвращается в следующем формате:

|  |  |  |
| --- | --- | --- |
| fs0 | fs1 | fs2 |

Формат свободного места на диске

fs0 = младший байт

fs1 = средний байт

fs2 = старший байт

По возвращении, регистр A устанавливается равным 0, если режим ошибок BDOS является режимом по умолчанию. Однако, если режим ошибок BDOS является одним из режимов возврата (см. Функцию 45), и возникает физическая ошибка, регистр A устанавливается равным 0FFH (255 в десятичной системе), и регистр H устанавливается в одно из следующих значений:

01 : Систематическая ошибка

04 : Ошибка выбора диска

|  |
| --- |
| **Функция 47: Chain To Program (Цепочка к программе)** |
| Входные параметры:  Регистр C: 2FH |

Функции *Chain To Program* обеспечивает средство объединения в цепочку от одной программы до следующей без вмешательства оператора. Хотя, для этого вызова нет переданного параметра, процесс вызова должен поместить командную строку заканчивающуюся нулевым байтом в буфер DMA по умолчанию.

Функция 47 не возвращает значений вызывающему процессу, потому что любые возникающие ошибки обрабатываются интерпретатором командной строки (CLI).

Примечание: Функция 47 делает условный вызов XDOS *Attach Console* (присоединение консоли) к процессу вызова. Если процесс вызова отсоединен от его консоли, сцепляемая программа не выполняется и функция 47 возвращается к процессу вызова.

|  |
| --- |
| **Функция 46: Flush Buffers (Очистка буферов)** |
| Входные параметры:  Регистр C: 30H  Возвращаемые значения:  Регистр A: Флаг ошибки  Регистр H: Систематическая ошибка |

Функция *Flush Buffers* вызывает запись любых записей, ожидающей записи, содержащихся во внутренних буферах блокирования/разблокирования. Эта функция влияет только на те системы, в которых реализован алгоритм блокирования/разблокирования отложенной записи в XIOS (см. [Раздел 2.2.12](#_Блокирование_и_разблокирование)).

По возвращении, регистр A устанавливается равным 0, если работа очистки была успешна. Если возникла физическая ошибка, функция *Flush Buffers* выполняет различные действия в зависимости от режима ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS находится в режиме по умолчанию, сообщение, идентифицирующее ошибку, выводится на консоль и процесс вызова завершается. В противном случае, функция *Flush Buffers* возвращает процессу вызова в регистре A значение 0FFH и регистру H присваивает следующий код физической ошибки:

01 : Систематическая ошибка

|  |
| --- |
| **Функция 100: Set Directory Label (Установка метки каталога)** |
| Входные параметры:  Регистр C: 64H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код каталога  Регистр H: Физическая или расширенная ошибка |

Функция *Set Directory Label* создает метку каталога или обновляет существующую метку каталога для указанного диска. Вызывающий процесс передает в регистровой паре DE адрес FCB, содержащий поля имени, расширения и экстента, которые будут присвоены метке каталога. Поля имени и расширения указанного FCB не используются для определения местоположения метки каталога в каталоге. Они просто копируются в обновляемую или создаваемую метку каталога. Поле экстента FCB (байт 12) содержит указанный пользователем байт данных метки каталога. Определение байта данных метки каталога следующее:

бит 7 - Требуется пароль для файлов защищенных паролем

6 - Установить метку даты и времени доступа

5 - Установить метку даты и времени обновления

4 - Функция Make создает XFCB

0 – Назначить новый пароль метке каталога

Если метка текущего каталога защищена паролем, корректный пароль должен быть помещен в первые 8 байтов текущего DMA или ранее установлен в качестве пароля по умолчанию (см. функцию 106). Если бит 0 (бит младшего разряда) байта 12 FCB установлен в 1, это указывает, что пароль для метки каталога был помещен во вторые восемь байтов текущего DMA.

Функция 100 возвращает код каталога в регистре A со значением от 0 до 3, если создание или обновление метки каталога прошло успешно, или 0FFH (255 в десятичной системе), если в указанном каталоге нет места для создания метки каталога. Регистр H обнуляется в обоих случаях. Если возникла физическая или расширенная ошибка, функция 100 выполняет различные действия в зависимости от режима ошибок BDOS (см. Функцию 45). Если режим ошибок BDOS является режим по умолчанию, сообщение, идентифицирующее ошибку, выводится в консоль и процесс завершается. В противном случае, функция 100 возвращает процессу вызова в регистре A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

02 : Диск только для чтения

04 : Ошибка выбора диска

07 : Ошибка пароля файла

|  |
| --- |
| **Функция 101: Return Directory Label Data (Получить данные метки каталога)** |
| Входные параметры:  Регистр C: 65H  Регистр E: Диск  Возвращаемые значения:  Регистр A: Байт данных метки каталога  Регистр H: Физическая ошибка |

Функция *Return Directory Label Data* возвращает байт данных метки каталога для указанного диска. Процесс вызова передает номер диска в регистре E: 0 для диска A, 1 для диска B, и т.д. до 15 для диска P в системах с шестнадцатью дисками. Формат байта данных метки каталога показан ниже:

бит 7 - Требуется пароль для файлов защищенных паролем

6 - Установить метку даты и времени доступа

5 - Установить метку даты и времени обновления

4 - Функция Make создает XFCB

0 – Метка каталога существует на диске

Функция 101 возвращает байт данных метки каталога вызывающему процессу в регистре A. Регистр A равный нулю указывает, что метка каталога отсутствует на указанном диске. Если возникает физическая ошибка в функции 101, когда режим ошибок BDOS находится в одном из режимов возврата (см. функцию 45), эта функция возвращает с регистре A значение 0FFH (255 в десятичной системе) и регистр H устанавливается к одному из следующих значений:

01 : Систематическая ошибка

04 : Ошибка выбора диска

|  |
| --- |
| **Функция 102: Read File XFCB (Чтение XFCB файла)** |
| Входные параметры:  Регистр C: 66H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код каталога  Регистр H: Физическая ошибка |

Функция *Read File XFCB* считывает информацию каталога XFCB для указанного файла в байты 20 - 32 из указанного FCB. Процесс вызова передает в регистровой паре DE адрес FCB, в котором были определены поля диска, имени и расширения файла.

Если функция 102 успешна, она устанавливает следующие поля в FCB, на который ссылается:

|  |  |
| --- | --- |
| байт 12 : | Поле режима пароля XFCB  бит 7 – режим чтения  бит 6 - режим записи  бит 5 - режим удаления  Байт 12 равный нулю указывает, что файлу не был присвоен пароль. |
| байты 13-23: | Поле пароля XFCB (в зашифрованном виде) |
| байты 24-27: | Поле временной метки создания или доступа XFCB |
| байты 28-31: | Поле временной метки обновления XFCB |

По возвращении, функция 102 возвращает код каталога в регистре A со значением от 0 до 3, если операция чтения XFCB была успешна, или 0FFH (255 в десятичной системе), если XFCB не был найден. Регистр H обнуляется в обоих случаях. Если возникает физическая или расширенная ошибка, функция 102 выполняет различные действия в зависимости от режима ошибок BDOS (см. функцию 45). Если режим ошибок BDOS является режим по умолчанию, сообщение, идентифицирующее ошибку, выводится в консоль и процесс завершается. В противном случае, функция 102 возвращает процессу вызова в регистре A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

04 : Ошибка выбора диска

|  |
| --- |
| **Функция 103: Write File XFCB (Запись XFCB файла)** |
| Входные параметры:  Регистр C: 67H  Регистр DE: Адрес FCB  Возвращаемые значения:  Регистр A: Код каталога  Регистр H: Физическая или расширенная ошибка |

Функция *Write File XFCB* создает новый XFCB или обновляет существующий XFCB для указанного файла. Вызывающий процесс передает в регистровой паре DE адрес FCB, в котором были определены поля диска, имени, расширения и экстента. Поле "ex", если установлено, определяет режим пароля и должен ли новый пароль быть присвоен файлу. Формат байта экстента показан ниже:

FCB байт 12 (ex): Режим пароля XFCB

бит 7 - Режим чтения

бит 6 - Режим записи

бит 5 - Режим удаления

бит 0 - Назначить новый пароль файлу

Если бит 0 установлен в 1, новый пароль должен находиться во вторых 8 байтах текущего DMA. Если FCB в настоящее время защищен паролем, корректный пароль ранее установленный в качестве пароля по умолчанию (см. Функцию 106) должен находиться в первых 8 байтах текущего DMA.

По возвращении, функция 100 возвращает код каталога в регистре A со значением от 0 до 3, если создание или обновление XFCB было успешно, или 0FFH (255 в десятичной системе), если метка каталога не существует на указанном диске, или файл, названный в FCB, не был найден, или нет места в каталоге для создания XFCB. Регистр H обнуляется в обоих случаях. Если возникает физическая или расширенная ошибка, функция 103 выполняет различные действия в зависимости от режима ошибок BDOS (см. функцию 45). Если режим ошибок BDOS является режимом по умолчанию, сообщение, идентифицирующее ошибку, выводится в консоль и процесс завершается. В противном случае, функция 103 возвращает процессу вызова в регистре A значение 0FFH и регистру H присваивается один из следующих кодов физических или расширенных ошибок:

01 : Систематическая ошибка

02 : Диск только для чтения

04 : Ошибка выбора диска

07 : Ошибка пароля файла

|  |
| --- |
| **Функция 104: Set Date and Time (Установить дату и время)** |
| Входные параметры:  Регистр C: 68H  Регистр DE: Адрес TOD  Возвращаемые значения:  Нет |

Функция *Set Date and Time* устанавливает внутрисистемные дату и время. Процесс вызова передает в регистровой паре DE адрес четырехбайтовой структуры, содержащей спецификацию даты и времени. Формат структуры данных даты и времени следующий:

байты 0 – 1 : Поле даты

байт 2: Поле часов

байт 3: Поле минут

Дата представляется в виде 16-разрядного целого числа с 1 днем, соответствующим 1 января 1978 года, Время представляется в виде двух байтов: часы и минуты, хранящиеся как две цифры BCD.

В МР/М II эта функция также устанавливает поле секунд системной даты и времени к нулю.

|  |
| --- |
| **Функция 105: Get Date and Time (Получить дату и время)** |
| Входные параметры:  Регистр C: 69H  Регистр DE: Адрес TOD  Возвращаемые значения:  TOD |

Функция *Get Date and Time* получает внутрисистемные дату и время. Процесс вызова передает в регистровой паре DE адрес четырехбайтовой структуры данных, которая получает значения даты и времени. Формат структуры данных совпадает с форматом, описанным в функции 104. Эта функция эквивалентна функций 155 MP/M II за исключением того, что она не возвращает поле секунд внутреннего времени.

|  |
| --- |
| **Функция 106: Set Default Password (Установка пароля по умолчанию)** |
| Входные параметры:  Регистр C: 6AH  Регистр DE: Адрес пароля  Возвращаемые значения:  Нет |

Функция *Set Default Password* (установка пароля по умолчанию) позволяет процессу определять значение пароля до получения доступа к файлу защищенному паролем. Когда файловая система получает доступ к файлу защищенному паролем, она проверяет текущий DMA и пароль по умолчанию на соответствие правильному значению. Ошибка пароля не возвращается, если любой пароль корректен. Пароль по умолчанию сохраняется BDOS во внутренней таблице, индексированной номером консоли вызывающего процесса. После назначения, он сохраняется до тех пор, пока процесс с тем же номером консоли не выполнит другой вызов установки пароля по умолчанию.

Для выполнения вызова функции 106, вызывающий процесс устанавливает в регистровой паре DE адрес 8-байтового поля, содержащего пароль.

|  |
| --- |
| **Функция 107: Return Serial Number (Получить серийный номер)** |
| Входные параметры:  Регистр C: 6BH  Возвращаемые значения:  Регистр DE: Поле серийного номера |

Функция 107 возвращает в регистровой паре DE адрес 6-байтового поля серийного номера MP/M II.

# Интерфейс XDOS

## Введение

Этот раздел содержит информацию о структурах данных, используемых в модуле XDOS. XDOS использует эти структуры данных для:

* управления ресурсами памяти;
* передачи сообщений между процессами;
* синхронизации выполнения процессов.

Также присутствуют описания функций XDOS, включая входные параметры и возвращаемые значения и обсуждение обработки ошибок XDOS. Читатель должен быть ознакомлен с материалом, рассмотренным в [Разделе 1](#_Введение_в_MP/M) перед продолжением.

## Структура данных дескриптора процесса

Каждый процесс, работающий под MP/M II связан с *дескриптором процесса*, который определяет все характеристики процесса. XDOS использует дескриптор процесса для сохранения и восстановления состояния процесса. Структура данных дескриптора процесса показана ниже на языках PL/M и ассемблере.

PL/M: Ассемблер:

DECLARE CNS$HNDLR STRUCTURE ( /\* CNSHND: \*/

PL ADDRESS, /\* DW 0 ; PL \*/

STATUS BYTE, /\* DB 0 ; Состояние процесса \*/

PRIORITY BYTE, /\* DB 200 ; Приоритет процесса \*/

STKPTR ADDRESS, /\* DW CNSTK+38 ; Указатель стека \*/

NAME (8) BYTE, /\* DB 'CNS ' ; Имя процесса (8 символов) \*/

CONSOLE$LIST BYTE, /\* DB 0 ; Назначенная консоль \*/

MEMSEG BYTE, /\* DB 0FFH ; Сегмент памяти(FF=resident) \*/

DPARAM ADDRESS, /\* DS 2 ; DPARAM (область буфера) \*/

THREAD ADDRESS, /\* DS 2 ; THREAD (список процессов) \*/

DISK$SET$DMA ADDRESS, /\* DS 2 ; Адрес DMA диска \*/

DISK$SLCT BYTE, /\* DS 1 ; Код диска/пользователя \*/

DCNT ADDRESS, /\* DS 2 ; DCNT системный счетчик \*/

SEARCHL BYTE, /\* DS 1 ; SEARCHL \*/

SEARCHA ADDRESS, /\* DS 2 ; SEARCHA \*/

PD EXTENT, /\* DS 2 ; PD EXTENT \*/

REGISTERS (10) ADDRESS, /\* DS 20 ;hl',de',bc',af',iy,ix,hl,de,bc,af \*/

EXTENSION ADDRESS) /\* DS 2 ; EXTENSION \*/

INITIAL (0,0,200,

.CNS$STK(19),'CNS ',

1,0FFH);

DECLARE CNS$STK (20) ADDRESS /\* CNSTK: \*/

INITIAL (

0C7C7H,0C7C7H,0C7C7H,0C7C7H, /\* DW 0C7C7H,0C7C7H,0C7C7H,0C7C7H \*/

0C7C7H,0C7C7H,0C7C7H,0C7C7H, /\* DW 0C7C7H,0C7C7H,0C7C7H,0C7C7H \*/

0C7C7H,0C7C7H,0C7C7H,0C7C7H, /\* DW 0C7C7H,0C7C7H,0C7C7H,0C7C7H \*/

0C7C7H,0C7C7H,0C7C7H,0C7C7H, /\* DW 0C7C7H,0C7C7H,0C7C7H,0C7C7H \*/

0C7C7H,0C7C7H,0C7C7H, /\* DW 0C7C7H,0C7C7H,0C7C7H \*/

STRT$CNS); /\* DW CNSPR ; CNSTK+38 = PROCEDURE ADR \*/

Элементы структуры данных дескриптора процесса, показанной выше, определены в *Таблице 3-1*.

Таблица ‑ Элементы дескриптора процесса

| **Элемент** | **Определение** |
| --- | --- |
| PL | 2-байтовое поле ссылки, первоначально установленное пользователем при создании процесса к адресу следующего дескриптора процесса или ноль, если больше ничего не существует. |
| STATUS | 1 байт, состояние процесса, устанавливается системой.  Диспетчер считывает байт состояния для определения операции, которую необходимо выполнить процессу. Значения байта состояния приведены ниже:  00 - процесс готов к запуску  01 - исключение процесса из очереди  02 - постановка процесса в очередь  03 - опрос процесса  04 - ожидание процессом флага  05 - процесс находится в списке задержки  06 - не реализовано в МР/М II  07 - завершить процесс  08 - установить приоритет процесса  09 - диспетчерезировать  10 - присоединить консоль  11 - отсоединить консоль  12 - назначить консоль  13 - присоединить устройство печати  14 - отсоединить устройство печати |
| PRIORITY | 1 байт, приоритет процесса, устанавливается пользователем. |
| STKPTR | 2 байта, указатель стека, изначально задается пользователем. |
| NAME | 8 байтов, имя процесса ASCII, устанавливается пользователем.  Старший бит каждого байта имени процесса зарезервирован для использования системой. Назначение каждого старшего бита в зависимости от NAME(n), описаны ниже:  NAME(0)' Старший бит NAME(0) в состоянии "on" указывает, что процесс выполняет прямые вызовы BIOS консоли и что MP/M II следует игнорировать все управляющие символы. Он также подавляет нормальную проверку состояния консоли, когда вызываются функции BDOS. Пользователь может задать этот бит.  NAME(1)' Старший бит NAME(1) в состоянии "on" указывает, что этот процесс в настоящее время выполняет повторно используемый код BDOS. MP/M II не позволяет прекратить процесс, пока он находится в BDOS. Любая попытка прервать процесс установит NAME(6)' в состояние "on". Этот бит устанавливается системой. Он не должен быть задан пользователем.  NAME(2)' Старший бит NAME(2) в состоянии "on" указывает, что при входе в BDOS для этого процесса не делается замена стека. Этот бит имеет приоритет над системной логической переменной указывающей, были ли выделены пользовательские системные стеки. Это требуется, когда больше чем один процесс совместно использует тот же сегмент памяти и делает вызовы функции BDOS. Пользователь может установить этот бит.  NAME(3)' Старший бит NAME(3) в состоянии "on" указывает, что прямая имитация клавиатуры должна быть подавлена. Прямая имитация клавиатуры делается путем выполнения вызова состояния консоли при каждом вызове функции ввода-вывода BDOS. Этот бит устанавливается пользователем.  NAME(4)' Старший бит NAME(4) в состоянии "on" указывает, что расширенные ошибки, возникшие в вызовах BDOS возвращаются в вызывающую программу. Обычно сообщение об ошибке выводится в консоль, и вызывающая программа завершается. Этот бит устанавливается пользователем.  NAME(5)' Старший бит NAME(5) в состоянии "on" указывает, что расширенные ошибки, возникшие в вызовах BDOS должны быть возвращены в вызывающую программу и сообщение об ошибке будет отображаться на консоли. Этот бит устанавливается пользователем.  NAME(6)' Старший бит NAME(6) в состоянии "on" указывает, что была предпринята попытка прервать процесс, в то время как или NAME(1)' или NAME(7)' были "включены". Этот бит устанавливается системой.  NAME(7)' Старший бит NAME(7) в состоянии "on" указывает, что процесс не должен быть прерван каким-либо образом. Попытка прервать этот процесс приводит к установке NAME(6)' в состояние "on". Этот бит устанавливается пользователем. |
| CONSOLE/LIST | 1 байт, младшие четыре бита содержат номер устройства консоли, а старшие четыре бита содержат номер устройства печати, используемые процессом, устанавливаются пользователем. |
| MEMSEG | 1 байт, индекс таблицы сегмента памяти. |
| DPARAM | 2 байта, зарезервированы для MP/M II. |
| THREAD | 2 байта, список потоков процессов, задается системой. |
| DISK$SET$DMA | 2 байта, адрес DMA по умолчанию, установленный системой вызовом BDOS установка DMA, могут быть установлены пользователем. |
| DISK$SLCT | 1 байт, код диска/пользователя по умолчанию, установленный системой вызовами BDOS установка пользователя и выбора диска, может быть установлен пользователем. |
| DCNT | 2 байта, зарезервированы для MP/M II. |
| SEARCHL | 1 байт, зарезервирован для MP/M II. |
| SEARCHA | 2 байта, зарезервированы для MP/M II. |
| PD EXTENT | 2 байта, зарезервированы для MP/M II. |
| REGISTERS | 20 байтов, область хранения регистров 8080/Z80, могут быть установлены пользователем до создания процесса для передачи параметров созданного процесса. Следующие элементы показывают распределение хранения регистров. Байты сохраняются обычным для 8080/Z80 способом, младший байт регистра предшествует старшему байту.  **Элемент Определение**  Байты 0 - 1 HL', Альтернативный Z80  Байты 2 - 3 DE', Альтернативный Z80  Байты 4 - 5 BC', Альтернативный Z80  Байты 6 - 7 AF', Альтернативный Z80  Байты 8 - 9 IY  Байты 10 - 11 IX  Байты 12 - 13 HL  Байты 14 - 15 DE  Байты 16 - 17 BC  Байты 18 - 19 AF |
| EXTENSION | 2 байта, зарезервированы для MP/M II. |

Следующие соглашения должны использоваться в именовании процессов, которые должны работать под MP/M II: процессы которые ожидают в очереди, которые получают командные хвосты от TMP, должны иметь то же имя что и очередь, которую они читают. Если процесс должен быть защищен от прерывания пользователем с помощью команды ABORT, в его имени должен быть по крайней мере один символ в нижнем регистре.

## Структуры данных очереди

Механизм очереди "первым прибыл - первым обслужен" (FIFO), реализованный в MP/M II, обеспечивает несколько важных функций в среде мультипрограммирования. Очереди могут быть использованы для обмена сообщениями между процессами, синхронизации процессов и обеспечения взаимного исключения.

MP/M II предназначена для упрощения управления очередью для пользователя и системных процессов. Очереди обрабатываются как файлы на диске и могут быть созданы, открыты, записаны в, прочтены из и удалены.

Структуры данных очереди, используемые MP/M II, включают блок управления очередью (QCB) и блок управления очередью пользователя (UQCB). Существует два типа блоков управления очередью: *циклические* или *связные*. Тип используемого QCB зависит от размера сообщения, которое содержит очередь. Сообщение размером от 0 до 2 байт используют циклические очереди, в то время как сообщения размером из 3 или более байтов используют связные очереди.

### Циклические очереди

Следующий пример показывает инициализацию QCB для циклической очереди, содержащей 80 сообщений, каждое из которых имеет длину один байт. Пример показан на языке PL/M и ассемблере.

PL/M: Ассемблер:

DECLARE CIRCULAR$QUEUE STRUCTURE ( /\* CRCQUE: \*/

QL ADDRESS, /\* DW 0 ; QL \*/

NAME(8) BYTE, /\* DB 'CIRCQUE ' ; NAME \*/

MSGLEN ADDRESS, /\* DW 1 ; MSGLEN \*/

NMBMSGS ADDRESS, /\* DW 80 ; NMBMSGS \*/

DQPH ADDRESS, /\* DS 2 ; DQPH \*/

NQPH ADDRESS, /\* DS 2 ; NQPH \*/

MSG$IN ADDRESS, /\* DS 2 ; MSGIN \*/

MSG$OUT ADDRESS, /\* DS 2 ; MSGOUT \*/

MSG$CNT ADDRESS, /\* DS 2 ; MSGCNT \*/

BUFFER (80) BYTE ) /\* BUFFER: \*/

INITIAL (0,'CIRCQUE ',1,80); /\* DS 80 ; BUFFER \*/

Элементы циклической очереди, показанной выше, определены в *Таблице 3-2*. Суммарные накладные расходы для очереди составляют 24 байта.

Таблица ‑ Элементы циклической очереди

| **Элемент** | **Определение** |
| --- | --- |
| QL | 2-байтовая ссылка, устанавливается системой. |
| NAME | 8 символов ASCII имени очереди, определяются пользователем. |
| MSGLEN | 2 байта, длина сообщения, устанавливается пользователем. |
| NMBMSGS | 2 байта, количество сообщений, устанавливается пользователем. |
| DQPH | 2 байта, заголовок списка удаления из очереди, устанавливается системой. |
| NQPH | 2 байта, заголовок списка постановки в очередь, устанавливается системой. |
| MSG$IN | 2 байта, указатель к следующему входному сообщению, устанавливается системой. |
| MSG$OUT | 2 байта, указатель к следующему выходному сообщению, устанавливается системой. |
| MSG$CNT | 2 байта, число сообщений в очереди, устанавливается системой. |
| BUFFER | n байт, где n равно длине сообщения умноженному на число сообщений. Область выделяется пользователем, устанавливается системой. Примечание: очереди взаимного исключения требуют 2-байтовый буфер для адреса владельца дескриптора процесса . |

### Связные очереди

В следующем примере показано, как инициализировать QCB для связной очереди, содержащей 4 сообщения, каждое длиной 33 байта.

PL/M: Ассемблер:

DECLARE LINKED$QUEUE STRUCTURE ( /\* LNKQUE: \*/

QL ADDRESS, /\* DS 2 ; QL \*/

NAME (8) BYTE, /\* DB 'LINKQUE ' ; NAME \*/

MSGLEN ADDRESS, /\* DW 33 ; MSGLEN \*/

NMBMSGS ADDRESS, /\* DW 4 ; NMBMSGS \*/

DQPH ADDRESS, /\* DS 2 ; DQPH \*/

NQPH ADDRESS, /\* DS 2 ; NQPH \*/

MH ADDRESS, /\* DS 2 ; MH \*/

MT ADDRESS, /\* DS 2 ; MT \*/

BH ADDRESS, /\* DS 2 ; BH \*/

BUFFER (140) BYTE ) /\* BUFFER: \*/

INITIAL (0,'LNKQUE ',33,4); /\* DS 2 ; MSG #1 LINK \*/

/\* DS 33 ; MSG #1 DATA \*/

/\* DS 2 ; MSG #2 LINK \*/

/\* DS 33 ; MSG #2 DATA \*/

/\* DS 2 ; MSG #3 LINK \*/

/\* DS 33 ; MSG #3 DATA \*/

/\* DS 2 ; MSG #4 LINK \*/

/\* DS 33 ; MSG #4 DATA \*/

Элементы связной очереди показанные выше, определены в *Таблице 3-3*. Суммарные накладные расходы для очереди составляют 24 байта.

Таблица ‑ Элементы связной очереди

| **Элемент** | **Определение** |
| --- | --- |
| QL | 2-байтовая ссылка, устанавливается системой. |
| NAME | 8 символов ASCII имени очереди, определяются пользователем. |
| MSGLEN | 2 байта, длина сообщения, устанавливается пользователем. |
| NMBMSGS | 2 байта, количество сообщений, устанавливается пользователем. |
| DQPH | 2 байта, заголовок списка удаления из очереди, устанавливается системой. |
| NQPH | 2 байта, заголовок списка постановки в очередь, устанавливается системой. |
| MH | 2 байта, заголовок сообщения, устанавливается системой. |
| BH | 2 байта, хвост сообщения, устанавливается системой. |
| BUFFER | n байт, где n равно длине сообщения плюс два умноженному на число сообщений. Область выделяется пользователем, устанавливается системой. |

### Блок управления очередью пользователя

Структура данных блока управления очередью пользователя (UQCB) обеспечивает доступ для чтения/записи к очередям таким же способом, как и FCB обеспечивает доступ к дисковому файлу. Как и файлы, очереди "открываются" операцией, которая заполняет фактический адрес QCB, после чего могут читаться или записываться.

Если известен фактический адрес очереди, он может использоваться в поле указателя UQCB, в этом случае 8-байтовое поле имени может быть опущено, и для получения доступа к очереди операция открытия не требуется. Если адрес не известен, то должна быть выполнена операция открытия (см. Функцию 135).

Следующий пример иллюстрирует, как инициализировать UQCB на языке PL/M и ассемблере.

PL/M: Ассемблер:

DECLARE USER$QUEUE$CONTROL$BLOCK STRUCTURE( /\* UQCB: \*/

POINTER ADDRESS, /\* DS 2 ; POINTER \*/

MSGADR ADDRESS, /\* DW BUFFER ; SGADR \*/

NAME (8) BYTE ) /\* DB 'SPOOL ' ; NAME \*/

INITIAL (0,.BUFFER,'SPOOL ');

DECLARE BUFFER (33) BYTE; /\* BUFFER: \*/

/\* DS 33 ; BUFFER \*/

Элементы UQCB, показанные выше, определены в *Таблице 3-4*.

Таблица ‑ Элементы UQCB

| **Элемент** | **Определение** |
| --- | --- |
| POINTER | 2 байта, фактический адрес очереди устанавливается системой во время операции открытия очереди или устанавливается пользователем, если фактический адрес очереди известен. |
| MSGADR | 2 байта, адрес буфера пользователя, устанавливается пользователем. |
| NAME | 8 байтов, имя очереди ASCII, определяется пользователем, может быть опущено, если пользователем установлено поле указателя. |

### Соглашения о присвоении имен очереди

В MP/M II должны использоваться следующие соглашения в именовании очередей: если процессор терминальных сообщений (TMP) должен записать непосредственно в очередь, то имя очереди должно быть в верхнем регистре ASCII. Таким образом, когда пользователь в системной консоли вводит имя очереди, а затем хвост команды, CLI пишет хвост команды непосредственно в очередь (см. [Раздел 1.5](#_Интерпретатор_командной_строки)).

Чтобы сделать очередь недоступной пользователям в системной консоли, имя очереди должно содержать по крайней мере один строчный символ ASCII. Очереди взаимного исключения нужно назвать прописным 'MX', сопровождаемым 1 - 6 дополнительными символами ASCII. У этих очередей должен быть двухбайтовый буфер, в который XDOS помещает адрес дескриптора процесса, владеющего сообщением взаимного исключения.

## Структура данных дескриптора памяти

Каждый процесс, работающий в MP/M II, связан с дескриптором процесса, который содержит индекс сегмента памяти. Этот индекс определяет конкретный дескриптор памяти в таблице сегментов памяти MP/M II. В MP/M II индекс сегмент памяти может иметь значения от 0 до 7, соответствующий таблице сегментов памяти из 8 записей или FFH, указывая, что процесс находится в общей памяти и не использует таблицу сегментов памяти. XDOS использует структуру данных дескриптора памяти для выделения и управления ресурсом памяти. Дескриптор памяти содержит четыре байта: адрес страницы начала смещения памяти, размер страницы сегмента памяти, атрибуты сегмента памяти и банк. Структура данных дескриптора памяти показана ниже на языках PL/M и ассемблере.

PL/M: Ассемблер:

Declare memory$descriptor structure ( /\* MEMDES: \*/

base byte, /\* DS 1 ; base \*/

size byte, /\* DS 1 ; size \*/

attrib byte, /\* DS 1 ; attributes \*/

bank byte ); /\* DS 1 ; bank \*/

Элементы дескриптора памяти, показанного выше, определены в *Таблице 3-5*.

Таблица ‑ Элементы дескриптора памяти

| **Элемент** | **Определение** |
| --- | --- |
| BASE | 1 байт, адрес базовой страницы сегмента памяти, устанавливается пользователем. |
| SIZE | 1 байт, размер сегмента памяти в страницах, устанавливается пользователем. |
| ATTRIBUTE | 1 байт, старший бит в состоянии "on" указывает, что сегмент памяти выделен, другие биты зарезервированы для MP/M II, обычно устанавливаются системой, но пользователь может предварительно выделить сегмент памяти, установив старший бит в состояние "on". |
| BANK | 1 байт, номер банка в диапазоне от 0 до 7, где банк 0 является банком, который активизируется при загрузке и инициализации MP/M II, устанавливается пользователем. |

## Системная страница данных

Системная страница данных является верхними 256 байтами операционной системы MP/M II. Она содержит статическую информацию о конфигурации системы, которую пользователь вводит при выполнении GENSYS для генерации системы. Она также содержит динамическую информацию, которая используется MP/M II во время выполнения.

*Таблица 3-6* описывает назначения отдельных байтов в системной странице данных.

Таблица ‑ Назначения байтов системной страницы данных

| **Байт** | **Содержание** |
| --- | --- |
| 000-000 | Mem$top, вершина страницы памяти |
| 001-001 | Nmb$cns, число системных консолей (TMP) |
| 002-002 | Brkpt$RST, номер рестарта точки останова |
| 003-003 | Добавление системного вызова пользовательских стеков, логическая |
| 004-004 | Коммутация банков, логическая |
| 005-005 | Версия Z80, логическая |
| 006-006 | Переключаемый BDOS, логическая |
| 007-007 | Страница таблицы переходов XIOS |
| 008-008 | Начальная страница RESBDOS |
| 009-010 | Адрес главной таблицы конфигурации CP/NET |
| 011-011 | Начальная страница XDOS |
| 012-012 | Начальная страница RSP (BNKXIOS top+1) |
| 013-013 | Начальная страница BNKXIOS |
| 014-014 | Начальная страница BNKBDOS |
| 015-015 | Max$mem$seg, максимальный номер сегмента памяти |
| 016-047 | Первоначальная таблица сегментов памяти |
| 048-063 | Таблица векторов точек останова, заполненная отладчиками |
| 064-079 | Зарезервированы для MP/M II |
| 080-095 | Системный вызов пользовательской таблицы указателя стека |
| 096-119 | Зарезервированы для MP/M II |
| 120-121 | Количество записей в файле MPM.SYS |
| 122-122 | Число тактов/сек |
| 123-123 | Системный диск |
| 124-124 | Начальная страница общей памяти |
| 125-125 | Число резидентных системных процессов (RSP) |
| 126-127 | Массив адресов Listcp |
| 128-143 | Subflg, массив флагов submit |
| 144-186 | Зарезервированы для MP/M II |
| 187-187 | Максимальное количество заблокированных записей/процессов |
| 188-188 | Максимальное количество открытых файлов/ процессов |
| 189-190 | Число устройств печати |
| 191-192 | Указатель на начало таблицы блокировок свободного места |
| 193-193 | Общая число записей заблокированных системой |
| 194-194 | Общее число файлов открытых в системе |
| 195-195 | Журнал регистрации, логическая |
| 196-196 | Диск для временных файлов |
| 197-197 | Число принтеров |
| 197-241 | Зарезервированы для MP/M II |
| 242-242 | Начальная страница переключаемой XDOS |
| 243-243 | Начало дескриптора процессов TMP |
| 244-244 | Начало CONSOLE.DAT |
| 245-246 | Точка входа BDOS/XDOS |
| 247-247 | Начало TMP.SPR |
| 248-248 | Nmbrsps, Число переключаемых RSP |
| 249-249 | Начальный адрес BRSP |
| 250-251 | Brspl, на не резидентные процессы RSP |
| 252-253 | Sysdatadr, Адрес сегмента внутренних данных XDOS |
| 254-255 | Rspl, ссылка на резидентный системный процесс |

## Сегмент внутренний данных XDOS

Этот раздел содержит информацию относительно расположения критически важных переменных, содержащихся в сегменте внутренних данных XDOS. Информация может быть полезной в некоторых прикладных программах. Однако, к ним нужно получать доступ с осторожностью. Информация может также быть полезной в отладке системы, разрешив доступ к списку готовых процессов (*Ready List*) через *Ready List Root* (RLR), как во время выполнения, а также в аварийном дампе.

Следующий пример, написанный на языке ассемблера, иллюстрирует метод для доступа к *Ready List Root* (корню списка готовых процессов).

; Смещения сегмента внутренних данных MP/M

0 equ 0000h ; Время дня

osrlr equ 0005h ; Корень списка готовых процессов

osdlr equ 0007h ; delay list root

osdrl equ 0009h ; Список процессов готовых к диспетчеризации

osplr equ 000Bh ; poll list root

osslr equ 000Dh ; swap list root (not used)

osqlr equ 000Fh ; queue list root

osthrdrt equ 0011h ; Поток (thread) root

osnmbcns equ 0013h ; Число консолей

oscnsatt equ 0014h ; Таблица присоединенных консолей

oscnsque equ 0034h ; Очередь консоли

osnmbflags equ 0054h ; Число флагов

ossysfla equ 0055h ; Системные флаги

osnmbsegs equ 0095h ; Число сегментов памяти

osmsegtbl equ 0096h ; Таблица сегментов памяти

ospdtbl equ 00B6h ; Таблица дескрипторов процесса

osnmblst equ 0256h ; Количество устройств печати

oslstatt equ 0257h ; Таблица присоединенных устройств печати

oslstque equ 0277h ; Список очереди

sysdatadr equ 154 ; Получить адрес страницы системных данных

. . .

mvi c,sysdatadr

call xdos ; HL = Системная страница данных

lxi d,00fch ; DE = Смещение к указателю

dad d

mov e,m

inx h

mov d,m ; DE = Начало внутреннего сегмента данных XDOS

lxi h,osrlr ; HL = Смещение к Ready List Root

dad d ; HL = Адрес Ready List Root

## Обработка ошибок XDOS

XDOS не требует возможности обработки ошибок, аналогичной BDOS, потому что функции XDOS выполняют "логические" или внутренние, а не "физические" или внешние операции. Т.е. функции XDOS реализованы полностью в резидентных структурах данных, и возникновение любой физической или расширенной "ошибки", по определению являются катастрофой для системы. Поэтому, те функции XDOS, которые возвращают значение в регистре A возвращают "логическую значение", которое является кодом, указывающим только, успешна ли функция. Если по некоторым причинам функция не успешна, процесс вызова должен быть в состоянии обработать это состояние ошибки. Коды возврата для функций XDOS определены в *Таблице 3-7*.

Таблица ‑ Коды возврата XDOS

|  |  |
| --- | --- |
| **Содержимое регистра A** | **Значение** |
| 0 | Успешная операция |
| FFH | Неуспешная операция |

## Вызовы функции XDOS

Для функций расширенной дисковой операционной системы (XDOS) включенных в этот раздел, описаны входные параметры и возвращаемые значения для каждой функции XDOS. Соглашения о вызовах XDOS идентичны BDOS, описанным в [Разделе 1.8](#_Соглашения_о_вызовах).

|  |
| --- |
| **Функция 128: Absolute Memory Request (Запрос абсолютной памяти)** |
| Входные параметры:  Регистр C: 80H  Регистр DE: Адрес дескриптора памяти  Возвращаемые значения:  Регистр A: Код возврата  Заполненный дескриптор памяти |

Функция запроса абсолютной памяти выделяет вызывающему процессу сегмент памяти, определенный параметром дескриптора памяти. Эта функция позволяет интерпретатору командной строки (CLI) загружать неперемещаемые программы, такие как CP/M файлы \*.COM, начинающиеся с абсолютного адреса TPA 0100H. Процесс вызова передает адрес дескриптора памяти в регистровой паре DE, устанавливая байт BASE. При возврате XDOS устанавливает другие байты. Структура данных дескриптора памяти описана в [Разделе 3.4](#_Структура_данных_дескриптора).

Функция 128 возвращает "логическое значение", указывающее, было ли выделение успешно. Возвращаемое значение FFH означает отказ выделить требуемую память, а значение 0 указывает успех. Если запрос абсолютной памяти имеет успех, индекс сегмента памяти процесса вызова устанавливается с учетом выделенной памяти. Таким образом чрезвычайно важно, чтобы эта функция была вызвана только из процесса, находящегося в общей памяти. Обратите внимание, что BASE и SIZE определяют начальный адрес базовой страницы и размер страницы, где страница составляет 256 байтов.

|  |
| --- |
| **Функция 129: Relocatable Memory Request (Запрос перемещаемой памяти)** |
| Входные параметры:  Регистр C: 81H  Регистр DE: Адрес дескриптора памяти  Возвращаемые значения:  Регистр A: Код возврата  Заполненный дескриптор памяти |

Функция запроса перемещаемой памяти выделяет требуемую непрерывную память вызывающему процессу. Процесс вызова передает адрес дескриптора памяти в регистровой паре DE, устанавливая байт SIZE. При возврате XDOS устанавливает другие байты.

Функция 129 возвращает в регистре A "логическое значение", указывающее, было ли выделение успешно. Возвращаемое значение FFH означает отказ выделить требуемую память, а значение 0 указывает успех. Если запрос перемещаемой памяти имеет успех, индекс сегмента памяти процесса вызова устанавливается с учетом выделенной памяти. Таким образом чрезвычайно важно, чтобы эта функция была вызвана только из процесса, находящегося в общей памяти. Обратите внимание, что BASE и SIZE определяют начальный адрес базовой страницы и размер страницы, где страница составляет 256 байтов.

|  |
| --- |
| **Функция 130: Memory Free (Освобождение памяти)** |
| Входные параметры:  Регистр C: 82H  Регистр DE: Адрес дескриптора памяти |

Функция *Memory Free* возвращает указанный сегмент памяти, принадлежащий вызывающему процессу обратно в операционную систему. Вызывающий процесс передает адрес дескриптора памяти в регистровой паре DE. Функция 130 не возвращает значение в регистре А.

|  |
| --- |
| **Функция 131: Poll (Опрос устройства)** |
| Входные параметры:  Регистр C: 83H  Регистр E: Номер устройства |

Функция *Poll* опрашивает указанное устройство, пока состояние готовности не получено. Вызывающий процесс освобождает процессор, пока выполняется опрос, позволяя выполняться другим процессам.

Функция 131 предназначена для использования в пользовательских XIOS, потому что XIOS связывает номер устройства с фактическим кодом, выполняемым для операции опроса. Это не исключает другое использование функции *Poll*, но это действительно означает, что прикладная программа, сделавшая вызов опроса, должен быть согласован с конкретным XIOS.

|  |
| --- |
| **Функция 132: Flag Wait (Ожидание флага)** |
| Входные параметры:  Регистр C: 84H  Регистр E: Номер флага  Возвращаемые значения:  Регистр A: Код возврата |

Функция *Flag Wait* (ожидания флага), заставляет процесс освободить ЦП, пока флаг, определенный в вызове, не будет установлен. Операция ожидания флага используется в системе на основе прерываний, чтобы заставить процесс вызова "ожидать" возникновение условия определенного прерывания.

Функция 132 возвращает в регистре A "логическое значение", указывающее, было ли выполнено успешное ожидание флага. Возвращаемое значение FFH означает, что ожидание флага не произошло, потому что другой процесс уже ожидал указанный флаг. Возвращаемое значение 0 указывает на успех.

Обратите внимание, что флаги не ставится в очередь, что означает, что доступ к флагам должен тщательно контролироваться. Как правило, физические обработчики прерываний устанавливают флаги, в то время как единственный процесс ожидает состояние "on" каждого флага.

|  |
| --- |
| **Функция 133: Flag Set (Установка флага)** |
| Входные параметры:  Регистр C: 85H  Регистр E: Номер флага  Возвращаемые значения:  Регистр A: Код возврата |

Функция установки флага "будит" ожидающий процесс. Функцию установки флага обычно вызывает процедура обработки прерывания после обслуживания прерывания и определения, какой флаг должен быть установлен.

Функция 133 возвращает в регистре A "логическое значение", указывающее, была ли выполнена успешная установка флага. Возвращаемое значение FFH указывает, что произошло переполнение флага, т.е. флаг уже был установлен, когда вызвали функцию установки флага. Возвращаемое значение 0 означает успех.

|  |
| --- |
| **Функция 134: Make Queue (Создание очереди)** |
| Входные параметры:  Регистр C: 86H  Регистр DE: Адрес блока управления очередью (QCB) |

Функция создания очереди устанавливает блок управления очередью (QCB). Очередь настраивается как циклическая или связная в зависимости от размера сообщения. Сообщения размером от 0 до 2 байтов используют циклические очереди, в то время как сообщения размером 3 или более байтов используют связные очереди.

Процесс вызова передает адрес блока управления очередью (QCB) в регистровой паре DE. QCB должен содержать имя очереди, длину сообщения, число сообщений, достаточно места для размещения сообщения и ссылки, если очередь связная.

Структуры данных QCB циклических и связных очередей описаны в [Разделе 3.3](#_Структуры_данных_очереди).

Очереди могут быть созданы только в общей памяти или пользовательскими программами в системах без коммутации банков, потому что все очереди сохраняются в связном списке, который должен быть доступен в любое время. Т.е. очередь не может находиться в сегменте памяти переключаемого банка. Однако. к очереди, создаваемой в общей памяти, могут получить доступ вся система и пользовательские программы.

|  |
| --- |
| **Функция 135: Open Queue (Открытие очереди)** |
| Входные параметры:  Регистр C: 87H  Регистр DE: Адрес блок управления очередью пользователя (UQCB)  Возвращаемые значения:  Регистр A: Код возврата |

Функция открытия очереди помещает фактический адрес QCB в блок управления очередью пользователя (UQCB). Функция 135 позволяет программе пользователя получать доступ к очереди, определяя только имя очереди. Процесс получает фактический адрес, вызывая функцию 135, и затем читает из или пишет в очередь, используя функции XDOS чтения и записи очереди.

Функция 135 возвращает "логическое значение" в регистре A указывающее, была ли операция открытия очереди успешна. Возвращаемое значение 0FFH означает отказ, в то время как 0 означает успех.

Структура данных блока управления очередью пользователя описана в [Разделе 3.3.3](#_Блок_управления_очередью).

|  |
| --- |
| **Функция 136: Delete Queue (Удаление очереди)** |
| Входные параметры:  Регистр C: 88H  Регистр DE: Адрес QCB (блока управления очередью)  Возвращаемые значения:  Регистр A: Код возврата |

Функция *Delete Queue* удаляет указанную очередь из списка очередей. Процесс вызова передает адрес QCB для указанной очереди в регистровой паре DE.

Функция 136 возвращает "логическое значение" в регистре A указывающее, была ли операция удаления очереди успешна. Возвращаемое значение 0FFH означает отказ, обычно потому что некоторый процесс не отсоединен от очереди. Возвращаемое значение 0 означает успех.

|  |
| --- |
| **Функция 137: Read Queue (Чтение из очереди)** |
| Входные параметры:  Регистр C: 89H  Регистр DE: Адрес UQCB (блока управления очередью пользователя)  Возвращаемые значения:  Прочтенное сообщение |

Функция чтения из очереди читает сообщение из очереди, определенной UQCB. Если сообщение не доступно в очереди, процесс вызова освобождает ЦП, пока другой процесс не запишет сообщение в очередь. Процесс вызова передает адрес UQCB в регистровой паре DE, и когда сообщение становится доступным в очереди, функция 137 копирует его в буфер, адресуемый полем MSGADR из UQCB.

|  |
| --- |
| **Функция 138: Conditional Read Queue (Условное чтение из очереди)** |
| Входные параметры:  Регистр C: 8AH  Регистр DE: Адрес UQCB (блока управления очередью пользователя)  Возвращаемые значения:  Регистр A: Код возврата  Прочтенное сообщение если доступно |

Функция условного чтения из очереди читает сообщение из очереди, определенной UQCB только, когда очередь содержит сообщение. Эта функция может использоваться, чтобы препятствовать приостановке выполнения вызывающего процесса, если сообщения не существуют. Процесс вызова передает адрес UQCB в регистровой паре DE, и если сообщение доступно в очереди, функция 138 копирует его в буфер, адресуемый полем MSGADR из UQCB.

Функция 138 возвращает "логическое значение" в регистре A указывающее, было ли сообщение доступно в очереди. Возвращаемое значение 0FFH указывает на отсутствие сообщения, в то время как 0 означает, что сообщение было доступно и было скопировано в буфер пользователя.

|  |
| --- |
| **Функция 139: Write Queue (Запись в очередь)** |
| Входные параметры:  Регистр C: 8BH  Регистр DE: Адрес UQCB (блока управления очередью пользователя)  Сообщение, которое будет отправлено |

Функция *Write Queue* записывает сообщение в очередь, заданную UQCB. Если буферы не доступны в очереди, вызывающий процесс освобождает процессор, до тех пор, пока один не станет доступным. Вызывающий процесс передает адрес UQCB в регистровой паре DE, и когда буфер доступен в очереди, функция копирует буфер, адресуемый полем MSGADR из UQCB в очередь. Функция 139 не возвращает значение в регистре A.

|  |
| --- |
| **Функция 140: Conditional Write Queue (Условная запись в очередь)** |
| Входные параметры:  Регистр C: 8CH  Регистр DE: Адрес UQCB (блока управления очередью пользователя)  Сообщение, которое будет отправлено  Возвращаемые значения:  Регистр A: Код возврата |

Функция условной записи в очередь записывает сообщение в очередь, заданную UQCB только тогда, когда буфер доступен. Эта функция может препятствовать приостановке выполнения процесса вызова, если буферы очереди переполнены. Вызывающий процесс передает адрес UQCB в регистровой паре DE, и если буфер доступен в очереди, функция копирует буфер, адресуемый полем MSGADR из UQCB в существующую очередь.

Функция 140 возвращает "логическое значение" в регистре A указывающее, был ли буфер доступен в очереди. Возвращенное значение 0FFH указывает на недоступность буфера, в то время как 0 означает, что буфер был доступен и пользовательский буфер был скопирован в него.

|  |
| --- |
| **Функция 141: Delay (Задержка)** |
| Входные параметры:  Регистр C: 8DH  Регистр DE: Количество тактов |

Функция *Delay* задерживает выполнение вызывающего процесса на заданное количество системных единиц времени, тем самым позволяя другим процессам использовать ЦП пока указанный срок не истечет. Использование функции 141 позволяет избежать типичных программных циклов задержки, которых следует избегать в МП/М II, потому что они используют процессор.

Единица системного времени обычно - 60 Гц (16.67 миллисекунд), в зависимости от приложения. Например, вполне вероятно, что в Европе она будет 50 Гц (20 миллисекунд).

Вызывающий процесс передает 16-разрядное целое число в регистровой паре DE, которое определяет число тактов, в течении которых процесс должен быть отложен. Начиная с вызова процедуры задержки, обычно асинхронной по отношению к реальному времени, существует неопределенность до одной такта в точном количестве времени задержки. Таким образом задержка 10 тактов гарантирует задержку по крайней мере 10 тактов, но она может составлять почти 11 тактов.

|  |
| --- |
| **Функция 142: Dispatch (Диспетчеризация)** |
| Входные параметры:  Регистр C: 8EH |

Функция *Dispatch* (диспетчеризации) заставляет MP/M II определять готовый процесс с самым высоким приоритетом, и затем предоставить этому процессу ЦП. Функция 142 предназначена для систем не основанных на прерываниях, в которых желательно позволить чисто вычислительным процессам периодически освобождать ЦП. Поскольку все пользовательские процессы, как правило, работают с тем же приоритетом, вызовы *Dispatch* в различных точках программы, предоставляет другой доступ процессам к ЦП циклическим (round-robin) способом. *Dispatch* может также безопасно включить прерывания после выполнения инструкции запрета прерываний (DI).

В регистровой паре DE никакие параметры не передаются, и в регистре A никакие значения не возвращаются. Процесс вызывает функцию 142, передавая номер функции 8EH в регистре C. Примечание: Вызов *Dispatch* не удаляет вызывающий процесс из списка готовых процессов.

|  |
| --- |
| **Функция 143: Terminate Process (Завершение процесса)** |
| Входные параметры:  Регистр C: 8FH  Регистр D: Условие освобождения памяти  Регистр E: Код завершения |

Функция *Terminate Process* завершает процесс вызова, который передает параметры в регистрах D и E, указывающие, должен ли процесс быть завершен, если он является системным процессом, и если сегмент памяти, принадлежащий вызывающему процессу должен быть освобожден. Значение 0FFH в регистре E указывает, что процесс должен быть завершен безоговорочно. Значение 0 указывает, что только пользовательский процесс должен быть удален. Если процесс вызова является пользовательским процессом, и регистр D содержит 0FFH, сегмент памяти, принадлежавший процессу не освобождается. Таким образом, процесс, который является потомком родительского процесса, оба из которых выполняется в том же сегменте памяти, можно завершить, не освобождая сегмент памяти, который также занят родителем.

Функция 143 не возвращает значение в регистре A. Процесс вызова просто прекращает существование с точки зрения MP/M II.

|  |
| --- |
| **Функция 144: Create Process (Создание процесса)** |
| Входные параметры:  Регистр C: 90H  Регистр DE: Адрес дескриптора процесса  Возвращаемые значения:  Заполненный дескриптор процесса |

Функция *Create Process* создает один или несколько процессов, помещая переданные дескрипторы процессов в список готовых процессов MP/M II.

Процесс вызова передает адрес дескриптора процесса в регистровой паре DE. Первое поле дескриптора процесса является полем ссылки, которая может указывать на другой дескриптор процесса.

Процессы могут только быть созданы или в общей памяти, или пользовательскими программами в системах без переключения банков, потому что все дескрипторы процессов сохраняются в связных списках, которые должны быть доступны в любой момент времени.

Структура данных дескриптора процесса описана в [Разделе 3.2](#_Структура_данных_дескриптора_2).

|  |
| --- |
| **Функция 145: Set Priority (Установить приоритет)** |
| Входные параметры:  Регистр C: 91H  Регистр E: Приоритет |

Функция *Set Priority* устанавливает или изменяет приоритет процесса вызова к переданному параметром. Процесс вызова передает приоритет в регистре E. Функция 145 не возвращает значение в регистре A.

Эта функция полезна, когда процесс должен иметь высокий приоритет во время фазы инициализации, но после этого должен работать с более низким приоритетом.

|  |
| --- |
| **Функция 146: Attach Console (Присоединение консоли)** |
| Входные параметры:  Регистр C: 92H |

Функция *Attach Console* присоединяет консоль к процессу вызова, определенную в поле CONSOLE дескриптора процесса. Если консоль уже присоединена к некоторому другому процессу, процесс вызова освобождает ЦП до освобождения консоли другим процессом. Когда консоль становится свободной, и процесс вызова является самый приоритетным процессом ожидающим консоль, выполняется операция присоединения консоли.

Нет никаких параметров, передаваемых в регистрах D и E, и никакие значения не возвращаются в регистре A. Процесс вызывает функцию 146, передавая номер функции 92H в регистре C.

|  |
| --- |
| **Функция 147: Detach Console (Отсоединение консоли)** |
| Входные параметры:  Регистр C: 93H |

Функция *Detach Console* отсоединяет от вызывающего процесса консоль указанную в поле CONSOLE дескриптора процесса. Если консоль в настоящее время не присоединена, никакие действия не выполняются.

Нет никаких параметров, передаваемых в регистрах D и E, и никакие значения не возвращаются в регистре A. Процесс вызывает функцию 147, передавая номер функции 93H в регистре C.

|  |
| --- |
| **Функция 148: Set Console (Установка консоли)** |
| Входные параметры:  Регистр C: 94H  Регистр E: Консоль |

Функция *Set Console* отсоединяет в настоящее время присоединенную консоль и затем присоединяет указанную консоль. Если консоль уже присоединена к некоторому другому процессу, процесс вызова освобождает ЦП до освобождения консоли другим процессом. Когда консоль становится свободной, и процесс вызова является самый приоритетным процессом ожидающим консоль, выполняется операция присоединения консоли.

Процесс вызова передает номер присоединяемой консоли в регистре E. Функция не возвращает значение в регистре A.

|  |
| --- |
| **Функция 149: Assign Console (Назначение консоли)** |
| Входные параметры:  Регистр C: 95H  Регистр DE: Адрес APB (Блока параметров назначения)  Возвращаемые значения:  Регистр A: Код возврата |

Функция *Assign Console* безоговорочно назначает консоль указанному процессу. Т.е. назначение выполняется независимо от того, ожидает ли какой-либо другой процесс в настоящее время присоединение консоли. Вызывающий процесс передает адрес структуры данных названой *Блоком параметров назначения* (APB). Эта структура данных содержит номер назначаемой консоли, 8 символов ASCII имени процесса и "логическую переменную", указывающую требуется ли соответствие с полем CONSOLE дескриптора процесса (True или 0FFH указывает, что оно требуется).

Чрезвычайно важно отметить, что вызывающий процесс должен быть владельцем консоли или консоль должна быть в настоящее время не присоединенной для надлежащего выполнения этой функции.

Функция 149 возвращает "логическое значение" в регистре A указывающее, было ли выполнено назначение. Возвращаемое значение 0FFH означает отказ в назначении консоли, либо потому, что дескриптор процесса с указанным именем не найден, или потому что требовалось соответствие, а поле CONSOLE дескриптора процесса не соответствовало указанной консоли. Возвращаемое значение 0 означает успешное назначение.

|  |
| --- |
| **Функция 150: Send CLI Command (Отправка команды CLI)** |
| Входные параметры:  Регистр C: 96H  Регистр DE: Адрес CLICMD (команды CLI) |

Функция *Send CLI Command* позволяет запущенному процессу отправить командную строку к интерпретатору командной строки (см. Раздел 1.5). Процесс вызова передает адрес структуры данных с названием *Команда CLI* (CLICMD) в регистровой паре DE. Эта структура данных содержит: номер диска/пользователя по умолчанию, номер консоли и командную строку. Инициализация структуры данных CLICMD показана ниже на языках PL/M и ассемблере.

PL/M: Ассемблер:

Declare CLI$command structure ( /\* CLICMD:

disk$user byte, /\* DS 1 ; Номер диска/пользователя \*/

/\* ; по умолчанию \*/

console byte, /\* DS 1 ; Номер консоли \*/

command$line (129) byte); /\* DS 129 ; Командная строка \*/

Первый байт структуры данных - номер диска/пользователя по умолчанию. Старшее четыре бита содержат диск по умолчанию и младшие четыре бита, содержат номер пользователя. Второй байт структуры данных содержит номер консоли для выполняемого процесса. Командная строка ASCII начинается с третьего байта и завершается нулевым байтом.

Чрезвычайно важно отметить, что CLI должен владеть консолью, указанной в параметре функции *Send CLI Command*. Это присвоение консоли к CLI может быть выполнено с помощью Функции 149 *Assign Console* (назначить консоль).

Функция 150 не возвращает значение в регистре A.

|  |
| --- |
| **Функция 151: Call Resident System Procedure**  **(Вызов системной резидентной процедуры)** |
| Входные параметры:  Регистр C: 97H  Регистр DE: Адрес CPB (Блока параметров вызова)  Возвращаемые значения:  Регистр HL: Код возврата |

Функция *Call Resident System Procedure* позволяет процессу вызывать необязательную Резидентную системную процедуру (RSP). Процессу вызова в регистровой паре DE передается адрес структуры данных названную *блоком параметров вызова* (CPB). Данные CPB содержат адрес строки с именем RSP из 8 символов ASCII, сопровождаемое двухбайтовым параметром, который вызывающий процесс передает к RSP.

Инициализация структуры данных CPB показана ниже на языках PL/M и ассемблере.

PL/M: Ассемблер:

Declare CALL$PB structure ( /\* CALLPB: \*/

Name$adr address, /\* DW NAME \*/

Param address ) /\* DW 0 ; Параметр \*/

initial (.name, 0);

Declare name (8) byte /\* NAME: \*/

initial ('Proc1 '); /\* DB 'Proc1 ' \*/

Функция 151 возвращает 0001H в регистровой паре HL, если вызванный RSP отсутствует. В противном случае, код, передаваемый назад от RSP. Как правило, возвращенное значение FFH означает отказ, в то время как 0 означает успех.

|  |
| --- |
| **Функция 152: Parse Filename (Разбор имени файла)** |
| Входные параметры:  Регистр C: 98H  Регистр DE: Адрес PFCB (Блок управления разбором имени файла)  Возвращаемые значения:  Регистр HL: Код возврата  Разобранный блок управления файлом (FCB) |

Функция *Parse Filename* анализирует спецификацию файла ASCII (FILENAME) и готовит блок управления файлом (FCB). Вызывающий процесс передает адрес структуры данных названной *Блок управления разбором имени файла* (PFCB) в регистровой паре DE. PFCB содержит адрес ASCII строки с именем файла с последующим адресом FCB назначения.

Структура данных PFCB показана ниже на языках PL/M и ассемблере.

PL/M: Ассемблер:

Declare Parse$FN$CB structure ( /\* PFNCB: \*/

File$name$adr address, /\* DW FLNAME \*/

FCB$adr address ) initial ( /\* DW FCB \*/

.file$name,.fcb );

Declare file$name (128) byte; /\* FLNAME: DS 128 \*/

Declare fcb (36) byte; /\* FCB: DS 36 \*/

Функция 152 предполагает спецификацию файла в следующем виде:

{D:}{FILENAME}{.TYP}{;PASSWORD}

в которой элементы, заключенные в фигурные скобки, являются необязательными.

Функция *Parse Filename* анализирует первую спецификацию файла, которую она находит в строке ввода. Функция сначала удаляет начальные пробелы и знаки табуляции. Затем функция предполагает, что спецификация файла заканчивается на первом встреченном разделителе, который находится вне контекста с конкретным полем, которое она анализирует. Например, если она находит двоеточие (:) и оно не является вторым символом спецификации файла, двоеточие ограничивает всю спецификацию файла. Функция распознает следующие символы в качестве разделителей:

пробел

табулятор

возврат каретки

null

; (точка с запятой) - за исключением перед полем пароля

= (знак равенства)

< (меньше чем)

> (больше чем)

. (точка) - за исключением после имени и до расширения файла

: (двоеточие) - за исключением перед именем файла и после диска

, (запятая)

[ (левая квадратная скобка)

] (правая квадратная скобка)

/ (наклонная черта)

$ (знак доллара)

Если функция достигает неграфического символа (в диапазоне 1 - 31), не упомянутого выше, она рассматривает его как ошибку.

Функция *Parse Filename* инициализирует указанный FCB следующим образом:

|  |  |
| --- | --- |
| Байт 0 | Поле диска устанавливает указанный диск. Если диск не определен используется значение по умолчанию. 0=диск по умолчанию, 1=A, 2=B и т.д. |
| Байты 1-8 | Имени присваивается указанное имя файла. Все буквы преобразуются в верхний регистр. Если имя меньше восьми символов в длину, остающиеся байты в поле имени файла дополняются пробелами.  Если в имени файла присутствует звездочка (\*), все остающиеся байты в поле имени файла заполняются вопросительными знаками (?). Если имя файла больше восьми байтов длиной, происходит ошибка. |
| Байты 9-11 | Расширение устанавливается в указанное расширение файла. Если расширение не определено, поле расширения заполняется пробелами. Все буквы преобразуются в верхний регистр. Если расширение меньше трех символов в длину, остающиеся байты в поле расширения файла дополняются пробелами. Если встречается звездочка (\*), все остающиеся байты заполняются вопросительными знаками (?). Если поле расширения больше 3 байтов в длину, возникает ошибка. |
| Байты 12-15 | Заполняются нулями. |
| Байты 16-23 | Поле пароля устанавливается в указанный пароль. Если пароль не определен, оно заполняется пробелами. Если пароль меньше восьми символов в длину, остающиеся байты дополняются пробелами. Все буквы преобразуются в верхний регистр. Если поле пароля больше восьми байтов в длину, происходит ошибка. |
| Байты 24-25 | Смещение начала пароля в строке FILENAME помещается сюда. Если пароль не определен, это поле обнуляется. Следует отметить, что пароль, обозначенный этим полем, находится в строке FILENAME, которая не изменяется функцией *Parse Filename*. Если в пароле присутствуют какие-либо символы в нижнем регистре, они должны быть преобразованы в верхний регистр, чтобы сделать его эквивалентным полю пароля в FCB. |
| Байт 26 | Число символов в указанном пароле помещается сюда. Если пароль не определен, это поле обнуляется. |

Если происходит ошибка, все поля, которые не были проанализированы, устанавливаются в значения по умолчанию, и функция возвращает в регистровой паре HL значение 0FFFFh, указывающее ошибку.

При успешном разборе, функция *Parse Filename* проверяет следующий элемент в строке FILENAME. Она пропускает конечные пробелы и табуляции и смотрит на следующий символ. Если символ - null (ноль) или возврат каретки, она возвращает 0, означающий конец строки FILENAME. Если следующий символ - разделитель, она возвращает адрес разделителя. Если следующий символ не разделитель, она возвращает адрес разделителя пробела или табуляции.

Если первым символом не являющимся пробелом или символом табуляции в строке FILENAME является нулем (0) или возвратом каретки, функция *Parse Filename* возвращает ноль, указывающий на конец строки, и FCB инициализируется значениями по умолчанию.

Если функция *Parse Filename* должна использоваться, чтобы проанализировать последующее имя файла в строке FILENAME, возвращенный адрес должен быть продвинут на разделитель прежде, чем поместить его в PFCB.

|  |
| --- |
| **Функция 153: Get Console Number (Получить номер консоли)** |
| Входные параметры:  Регистр C: 99H  Возвращаемые значения:  Регистр A: Номер консоли |

Функция *Get Console Number* получает значение поля CONSOLE из дескриптора процесса вызывающего процесса. Процесс вызова помещает номер функции 99H в регистр C, и функция возвращает номер консоли в регистре A.

|  |
| --- |
| **Функция 154: System Data Address (Адрес системных данных)** |
| Входные параметры:  Регистр C: 9AH  Возвращаемые значения:  Регистр HL: Адрес системной страницы данных |

Функция *System Data Address* возвращает начальный адрес страницы системных данных. Страница системных данных находится в верхних 256 байтах операционной системы MP/M II. Она содержит сведения о конфигурации, введенные программой MP/M II GENSYS, а также данные времени выполнения, включая submit флаги. Содержание системной страницы данных описано в [Разделе 3.5](#_Системная_страница_данных_1).

Вызывающий процесс передает номер функции 9AH в регистре C, и функция возвращает начальный адрес системной страницы данных в регистровой паре HL.

|  |
| --- |
| **Функция 155: Get Date and Time (Получить дату и время)** |
| Входные параметры:  Регистр C: 9BH  Регистр DE: Адрес TOD  Возвращаемые значения:  Дата и время |

Функция *Get Date and Time* возвращает текущую закодированную дату и время. Вызывающий процесс передает адрес структуры данных названной TOD в регистровой паре DE. Структура данных TOD представляет дату как 16-разрядное целое число с 1 днем соответствующим 1 января 1978 г. Она представляет время как три байта: часы, минуты и секунды, сохраненные как две цифры в формате BCD.

Инициализация структуры данных TOD показана ниже на языках PL/M и ассемблере.

PL/M: Ассемблер:

Declare TOD structure ( /\* TOD: \*/

date address, /\* DS 2 ; Дата \*/

hour byte, /\* DS 1 ; Часы \*/

min byte, /\* DS 1 ; Минуты \*/

sec byte ); /\* DS 1 ; Секунды \*/

|  |
| --- |
| **Функция 156: Return Process Descriptor Address**  **(Возврат адреса дескриптора процесса)** |
| Входные параметры:  Регистр C: 9CH  Возвращаемые значения:  Регистр HL: Адрес дескриптора процесса |

Функции *Return Process Descriptor Address* возвращает адрес дескриптора процесса вызывающего процесса. По определению это - заголовок списка готовых процессов.

|  |
| --- |
| **Функция 157: Abort Specified Process (Завершить указанный процесс)** |
| Входные параметры:  Регистр C: 9DH  Регистр DE: Адрес ABTPB (Блока параметров завершения процесса)  Возвращаемые значения:  Регистр A: Код возврата |

Функция *Abort Specified Process* позволяет процессу завершать другой указанный процесс. Процесс вызова передает адрес структуры данных названной *блоком параметров завершения процесса* (ABTPB) в регистровой паре DE. Инициализация ABTPB показана ниже на языках PL/M и ассемблере.

PL/M: Ассемблер:

Declare Abort$paramter$block structure( /\* APB: \*/

pdadr address, /\* DS 2 ; Адрес дескриптора процесса \*/

termination$code address, /\* DS 2 ; Код завершения \*/

name (8) byte, /\* DS 8 ; Имя процесса \*/

console byte ); /\* DS 1 ; Консоль используемая процессом \*/

Если известен адрес дескриптора процесса, он может быть заполнен и можно опустить поля имени процесса и консоли. В противном случае, поле адрес дескриптора процесса должно быть равно 0, и должны быть указаны имя процесса и консоли. Код завершения, который является параметром, передаваемым функции 143 *Terminate Process* должен быть предоставлен в любом случае.

|  |
| --- |
| **Функция 158: Attach List (Присоединение устройства печати)** |
| Входные параметры:  Регистр C: 9EH |

Функция *Attach List* присоединяет устройство печати, определенное в поле CONSOLE/LIST дескриптора процесса к процессу вызова. Если устройство печати уже присоединено к некоторому другому процессу, вызывающий процесс освобождает ЦП до момента отсоединения другого процесса от устройства печати. Когда устройство печати становится свободным, и процесс вызова с самым высоким приоритетом ожидает устройство печати, выполняется операция присоединения.

Процесс вызывает функцию 158, передавая номер функции 9EH в регистре C. Функция не возвращает значение в регистре A.

|  |
| --- |
| **Функция 159: Detach List (Отсоединение устройства печати)** |
| Входные параметры:  Регистр C: 9FH |

Функция *Detach List* отсоединяет устройство печати, определенное в поле CONSOLE/LIST дескриптора процесса от процесса вызова. Если устройство печати в настоящее время не присоединено, никакие действия не выполняются.

Процесс вызывает функцию 159, передавая номер функции 9FH в регистре C. Функция не возвращает значение в регистре A.

|  |
| --- |
| **Функция 160: Set List (Установить устройство печати**) |
| Входные параметры:  Регистр C: A0H  Регистр E: Устройство печати |

Функция *Set List* отсоединяет устройство печати, в настоящее время присоединенное к процессу вызова, и затем присоединяет указанное устройство печати. Если устройство печати уже присоединено к некоторому другому процессу, вызывающий процесс освобождает ЦП до момента отсоединения другого процесса от устройства печати. Когда устройство печати становится свободным, и процесс вызова с самым высоким приоритетом ожидает устройство печати, выполняется операция присоединения.

Вызывающий процесс передает номер присоединяемого устройства печати в регистре E. Функция не возвращает значение в регистре A.

|  |
| --- |
| **Функция 161: Conditional Attach List**  **(Условное присоединение устройства печати)** |
| Входные параметры:  Регистр C: A1H  Возвращаемые значения:  Регистр A: Код возврата |

Функция *Conditional Attach List* присоединяет устройство печати, определенное в поле CONSOLE/LIST дескриптора процесса к процессу вызова, только если устройство печати в настоящее время свободно.

Если устройство печати в настоящее время присоединено к другому процессу функция возвращает в регистре A значение 0FFH, указывая, что устройство печати не могло быть присоединено. Функция возвращает значение 0, чтобы указать, что устройство печати уже присоединено к процессу или, что оно было свободно, и операция присоединения выполнена успешно.

|  |
| --- |
| **Функция 162: Conditional Attach Console (Условное присоединение консоли)** |
| Входные параметры:  Регистр C: A2H  Возвращаемые значения:  Регистр A: Код возврата |

Функция *Conditional Attach Console* присоединяет консоль, определенную в поле CONSOLE/LIST дескриптора процесса к процессу вызова, только если консоль в настоящее время свободна.

Если консоль в настоящее время присоединена к другому процессу функция возвращает в регистре A значение 0FFH, указывая, что консоль не могла быть присоединена. Функция возвращает значение 0, чтобы указать, что консоль уже присоединена к процессу, или что она было свободна, и успешно выполнена операция присоединения.

|  |
| --- |
| **Функция 163: Return MP/M Version Number (Возврат номера версии MP/M)** |
| Входные параметры:  Регистр C: A3H  Возвращаемые значения:  Регистр HL: Номер версии |

Функция *Return MP/M Version Number* предоставляет информацию, которая позволяет программирование независящее от версий. Функция возвращает двухбайтовое значение в H = 01 для MP/M II и в L уровень редакции MP/M II.

|  |
| --- |
| **Функция 164: Get List Number (Получить номер устройства печати)** |
| Входные параметры:  Регистр C: A4H  Возвращаемые значения:  Регистр A: Номер устройства печати |

Функция *Get List Number* возвращает значение устройства печати из дескриптора процесса вызывающего процесса. Процесс вызывает функцию 164, передавая номер функции A4H в регистре C. Функция возвращает номер устройства печати в регистре A.

# Ассемблер 8080 ASM

## Обзор

Программа ASM читает исходные файлы на языке ассемблера с диска и производит машинный язык 8080 в формате Intel Hex. Вызов программы ASM, осуществляется вводом команды ASM в любой из следующих форм:

ASM filename

ASM filename.parms

В обоих случаях, программа ассемблера предполагает, что на диске существует файл с названием:

filename.ASM

содержащий исходный файл на языке ассемблера 8080. Первая и вторая формы, показанные выше, отличаются только тем, что вторая форма передает параметры в ассемблер для управления доступом к файлу источника и файлам назначения HEX и печати.

В любом случае, MP/M II загружает программу ASM, которая выводит сообщение:

MP/M ASSEMBLER VER 2.0

где 2.0 - номер текущей версии. В ответ на команду ассемблер читает исходный файл с предполагаемым расширением файла "ASM" и создает два выходных файла:

filename.HEX

filename.PRN

Файл HEX содержит машинный код, соответствующий исходной программе в формате Intel hex, и файл PRN содержит аннотированный листинг, показывающий сгенерированный машинный код, флаги ошибок и исходные строки. Если во время трансляции возникают ошибки, они перечисляются в файле PRN, а также в консоли.

Вторая форма команды может перенаправить файлы ввода и вывода от их значений по умолчанию. Часть команды "parms" является группой из трех букв, которые определяют источник исходного файла, место назначения HEX файла и место назначения файла печати. Форма команды следующая:

filename.p1p2p3

где p1, p2 и p3 заменяются отдельными буквами, значения которых определены в *Таблице 4-1*.

Таблица ‑ Параметры ASM

|  |  |  |
| --- | --- | --- |
| **Символ** | **Допустимая буква** | **Значение** |
| p1 | A, B, . . .,P | Определяет диск, который содержит исходный файл |
| p2 | A, B, . . .,P | Обозначает диск, который получает HEX файл. |
|  | z | Пропустить генерацию HEX файла |
| p3 | A, B, . . .,P | Определяет диск, который получает файл печати |
|  | X | Выводит листинг на консоль |
|  | Z | Пропустить генерацию файла печати |

Таким образом, команда:

ASM PROG.AAA

указывает, что ассемблер берет исходный файл (PROG.ASM) на диске A, и также создает файлы HEX (PROG.HEX) и печати (PROG.PRN) на диске A. Эта команда используется по умолчанию, если ассемблер выполнен с диска A без дополнительных параметров, как показано ниже:

0A>ASM PROG

Команда:

0A>ASM PROG.ABX

указывает, что ассемблер берет исходный файл на диске A, помещает HEX файл на диск B, и отправляет файл листинга на консоль. Команда:

0A>ASM PROG.BZZ

берет исходный файл с диска B и пропускает генерацию файлов печати и HEX. Используйте эту команду для быстрого выполнения ассемблера с целью проверки синтаксиса программы.

Исходный формат программы совместим с ассемблером Intel 8080, несмотря на то, что макросы не поддерживаются. Однако некоторые расширения ассемблера MP/M II упрощают его использование. Эти расширения описаны ниже.

## Формат программы

Программы на языке ассемблера, приемлемом в качестве ввода для ассемблера ASM состоит из последовательности операторов вида:

номер\_строки метка операция операнд ;комментарий

в котором любое или все поля могут присутствовать в конкретном случае. Каждый оператор языка ассемблера должен быть завершен символами возврата каретки и перевод строки (ED автоматически вставляет перевод строки), или символом восклицательного знака "!", который рассматривает как конец строки ассемблера. Таким образом, несколько операторов ассемблера могут быть записаны на одной физической строке, если разделены восклицательными знаками.

номер\_строки - необязательное десятичное целочисленное значение, представляющее собой исходный номер строки программы, который допустим в любой исходной строке. Поскольку эти номера строк вставляются автоматически строчно-ориентированными редакторами, ASM игнорирует это поле, если оно существует.

Поле метки принимает любую из форм ниже:

идентификатор

идентификатор:

Метки являются необязательными, за исключением отдельно оговоренных случаев, в частности, операторов типа. Идентификатор представляет собой последовательность из любых буквенно-цифровых символов, но первым символом должна быть буква. Вы можете использовать идентификаторы свободно, чтобы помечать элементы, такие как шаги программы и директивы ассемблера. В идентификаторе значимыми являются только первые 16 символов, за исключением встроенного символа доллара "$", который может использоваться для улучшения удобочитаемость имени. Все буквенные символы в нижнем регистре обрабатываются, как если бы они были в верхнем регистре. При необходимости за идентификатором может следовать двоеточие. Таким образом все метки перечисленные ниже являются допустимыми:

X xy long$name

X: yxl: longer$named$data:

XlY2 Xlx2 x234$5678$9012$3456:

Поле операции содержит или директиву ассемблера, псевдо оператор или код операции 8080 машины. Псевдо операторы и коды операций машины описаны в [Разделе 4.5](#_Operation_Codes). [Раздел 4.4](#_Assembler_Directives) описывает директивы ассемблера.

Поле операнда в операторе обычно содержит выражение, сформированное из констант и меток, вместе с арифметическими и логическими операциями с этими элементами. Все детали корректно сформированных выражений приведены в [Разделе 4.3](#_Формирование_операнда).

Поле комментария может содержать любые символы после точки с запятой ";" до следующего реального или логического конца строки. Эти символы читаются и распечатываются, но игнорируются ассемблером. Ассемблер MP/M также рассматривает операторы, которые начинаются со звездочки "\*" в первом столбце в качестве операторов комментариев. Они распечатываются и игнорируются в процессе ассемблирования.

Исходная программа ассемблера является последовательность операторов, определенных выше, необязательно завершенных оператором END. ASM игнорирует операторы после END.

## Формирование операнда

Чтобы полностью описать коды операций и псевдо-операций, сначала необходимо представить форму поля операнда, потому что оно появляется почти во всех операторах. Выражения в поле операнда состоят из простых операндов (меток, констант и зарезервированных слов), объединенных в правильно сформированные подвыражения с помощью арифметических и логических операторов. ASM вычисляет каждое выражение в процессе ассемблирования. Каждое выражение должно иметь 16-разрядное значение. Кроме того, количество значащих цифр в результате не должна превышать предполагаемого использования. То есть, если выражение должно использоваться непосредственно в инструкции перемещения байта, то старшие значащие 8 битов выражения должны быть равны нулю. Ограничения на значения выражений приводятся в индивидуальных инструкциях.

### Метки

Как описано выше, метка является идентификатором, который появляется в составе конкретного оператора. В общем, метке присваивается значение, определенное типом оператора, которому она предшествует. Если метка появляется в операторе, который генерирует машинный код или резервирует область памяти (например, инструкция MOV или псевдо операция DS), то метке присваивается значение адреса программы, помечаемого им. Если метка предшествует EQU или SET, то ASM дает метке значение, которое следует из вычисления поля операнда. За исключением оператора SET, идентификатор может помечать только один оператор.

Когда метка появится в поле операнда, ASM подставляет ее значение во время ассемблирования. Затем это значение может быть объединено с другими операндами и операторами, чтобы сформировать поле операнда для определенной инструкции.

### Числовые константы

Числовая константа представляет собой 16-битное значение в одном из нескольких оснований. Основание, называемое системой счисления константы, обозначается завершающим показателем системы счисления. Показатели системы счисления распознаваемые ASM определены в *Таблице 4-2* ниже.

Таблица ‑ Показатели системы счисления

|  |  |
| --- | --- |
| **Индикатор** | **Основание** |
| B | двоичная константа (основание 2) |
| O | восьмеричная константа (основание 8) |
| Q | восьмеричная константа (основание 8) |
| D | десятичная константа (основание 10) |
| H | шестнадцатеричная константа (основание 16) |

Q принимается в качестве альтернативного показателя системы счисления для восьмеричных чисел, чтобы минимизировать путаницу между буквой O и цифра 0. Любая числовая константа, которая не завершается показателем основания, считается десятичной константой.

Константа таким образом состоит из последовательность цифр, сопровождаемых дополнительным показателем основания, в которой цифры находятся в соответствующем диапазоне для основания. Т.е. двоичные константы должны быть составлены из цифр 0 и 1, восьмеричные константы могут содержать цифры в диапазоне 0-7, в то время как десятичные константы содержат десятичные цифры. Шестнадцатеричные константы содержат десятичные цифры, а также шестнадцатеричные цифры A (10D), B (11D), C (12D), D (13D), E (14D) и F (15D).

Обратите внимание, что ведущая цифра шестнадцатеричной константы должна быть десятичной цифрой, чтобы ASM не мог перепутать шестнадцатеричную константу с идентификатором (предшествующего 0 будет всегда достаточно). Константа, составленная таким способом, должна вычисляться к двоичному числу, которое может содержаться в 16-разрядном счетчике. Иначе, оно усекается до наименее значимых 16 бит. По аналогии с идентификаторами, встроенные символы $ допускаются в константах для улучшения удобочитаемости. Наконец, ASM переводит показатель основания в верхний регистр, если встречается буква в нижнем регистре. Все приведенные ниже константы являются допустимыми:

1234 1234D 1100B 1111$0000$1111$0000B

1234H 0FFEH 33770 33$77$22Q

3377o 0fe3h 1234d 0ffffh

### Зарезервированные слова

Несколько зарезервированных символьных последовательностей символов имеют предопределенные значения в поле операнда оператора. Имена регистров процессора 8080, когда встречаются ассемблером, переводятся в значения, показанные в *Таблице 4-3*.

Таблица ‑ Регистры 8080

|  |  |
| --- | --- |
| **Буквы регистров** | **Значение** |
| A | 7 |
| B | 0 |
| C | 1 |
| D | 2 |
| E | 3 |
| H | 4 |
| L | 5 |
| M | 6 |
| SP | 6 |
| PSW | 6 |

Опять же, строчные имена в нижнем регистре имеют те же значения, что и их эквиваленты в верхнем регистре. Машинные команды также могут появиться в поле операнда. Если это так, он вычисляет их внутренние коды. Для инструкции, которой требуется операнд, в которой конкретный операнд становится частью двоичного битового шаблона команды (например, MOV A, B), значение инструкции (в этом случае MOV) является комбинацией двоичных разрядов инструкции с нулями в дополнительных полях (например, MOV производит 40H).

Если символ $ появляется в поле операнда, но не в составе идентификатора или числовой константы, его значение становится адресом начала следующей инструкции не включая инструкцию содержащуюся в текущей логической строке.

### Строковые константы

Строковые константы представляют собой последовательность символов ASCII окруженных символами апострофа ('). Все строки должны полностью находиться в текущей физической строке, таким образом позволяя символы "!" в строках, и не должны превышать 64 символов в длину. Символ апострофа может быть включен в строку, введя его дважды (''), который становится одним апострофом при чтении ассемблером. За исключением псевдо оператора DB, длина строки ограничена одним или двумя символами, которые становятся 8-разрядным или 16-разрядным значением, соответственно. Двух-символьные строки становятся 16-разрядной константой со вторым символом в младшем байте и первым символом в старшем байте.

Значение символа соответствует его коду ASCII (см. Приложение I). Преобразования символов в строке не производятся, и таким образом, могут быть представлены символы в верхнем и нижнем регистре. Заметьте однако, что в строках допускаются только графические (печатные) символы ASCII. Некоторые примеры допустимых строк

'A' 'AB' 'ab' 'c'

'''' 'a''' '''''' ''”'

'Walla Walla Wash.'

'She said ''Hello'' to me.'

'I said ''Hello'' to her.'

### Арифметические и логические операторы

Операнды, описанные выше, могут быть объединены в нормальной алгебраической нотации, используя любую комбинацию из правильно сформированных операндов, операторов и заключенных в скобки выражений. Операторы, распознаваемые в поле операнда, приведены в *Таблице 4-4*.

Таблица ‑ Арифметические и логические операторы

|  |  |
| --- | --- |
| **Операция** | **Результат** |
| a + b | арифметическая сумма без знака a и b |
| a - b | беззнаковая арифметическая разность между a и b |
| + b | унарный плюс (производит b) |
| - b | унарный минус (идентичен 0 - b) |
| a \* b | умножение без знака a и b |
| a / b | беззнаковое деление a на b |
| a MOD b | остаток после a / b |
| NOT b | логическая инверсия b: все 0 становятся 1, 1 становятся 0, b 16-разрядное значение |
| a AND b | поразрядное логическое И a и b |
| a OR b | поразрядное логическое ИЛИ a и b |
| a XOR b | побитовое логическое исключающее ИЛИ a и b |
| a SHL b | значение, полученное сдвигом влево a на величину b с заполнением нулями |
| a SHR b | значение, полученное сдвигом вправо a на величину b с заполнением нулями |

В *Таблице 4-4*, a и b представляют собой простые операнды, такие как метки, числовые константы, зарезервированные слова, одно или двух символьные строки, или полностью заключенные в скобки подвыражения, такие как в примерах ниже.

10+20 10h+37Q L1/3 (L2+4) SHR 3

('a' and 5fh) + '0' ('B'+B) OR (PSW+M)

(1+(2+c)) shr (A-(B+1))

Обратите внимание, что все вычисления, выполненные во время ассемблирования, являются 16 битовыми операциями без знака. Таким образом, -1 вычисляется как 0-1, который приводит к значению 0FFFFh (т.е., все биты равны 1). Полученное выражение должно соответствовать коду операции, в котором оно используется. Если, например, выражение используется в инструкции ADI (непосредственное сложение), то старшие восемь битов выражения должны быть равными нулю. Например, операция "ADI -1" приводит к сообщению об ошибке, потому что -1 становится 0FFFFh, которое не может быть представлено как 8-разрядное значение. Выражение "ADI (-1) AND 0FFH" приемлемо, потому что операция "AND" обнуляет старшие биты выражения.

### Приоритет операторов

Для удобства программиста, ASM предполагает, что операторы имеют относительный приоритет выполнения. Это позволяет писать выражения без вложенных уровней скобок. Выражения в скобках обладают относительным приоритетом. Порядок выполнения операторов без скобок в выражении приведен ниже. Операторы перечисленные первыми имеют наивысший приоритет. В выражении без скобок они выполняются первыми. Операторы перечисленные последними имеют самый низкий приоритет. Операторы перечисленные в той же строке, имеют одинаковый приоритет и выполняются слева направо, как они встречаются в выражении.

1) / MOD SHL SHR

2) - +

3) NOT

4) AND

5) OR XOR

Благодаря этой иерархии, выражения, показанные ниже слева интерпретируются ассемблером как выражения со скобками показанными справа:

|  |  |
| --- | --- |
| a \* b + c  a + b \* c  a MOD b \* c SHL d  a OR b AND NOT c + d SHL e | (a \* b) + c  a + (b \* c)  ((a MOD b) \* c) SHL d  a OR (b AND (NOT (c + (d SHL e)))) |

Сбалансированные скобки включающие в себя подвыражения могут всегда переопределять подразумеваемые скобки, и таким образом, последнее выражение, приведенное выше, может быть переписано, чтобы вызвать выполнение операторов в другом порядке, например:

(a OR b) AND (NOT c) + d SHL e

Это выражение имеет предполагаемые скобки:

(a OR b) AND HNOT c) + (d SHL e) )

Обратите внимание, что выражение без скобок правильно построено, только если результирующее выражение, получаемое после добавления подразумеваемых скобок, построено правильно.

## Директивы ассемблера

Директивы ассемблера присваивают меткам определенные значения во время ассемблирования, выполняют условное ассемблирование, определяют области хранения и определяют начальные адреса в программе. Каждая ассемблерная директива обозначается "псевдо оператором", которая появляется в поле оператора строки. Допустимые псевдо операторы приведены в *Таблице 4-5* и описаны индивидуально в следующих разделах.

Таблица ‑ Директивы ASM

| **Символ** | **Функция** |
| --- | --- |
| ORG | Устанавливает адрес программы или данных |
| END | Обозначает конец программы, дополнительный начальный адрес |
| EQU | Числовое "приравнивание" |
| SET | Числовая "установка" |
| IF | Начало условного ассемблирования |
| ENDIF | Конец условного ассемблирования |
| DB | Определение байта данных |
| DW | Определение слова данных |
| DS | Определение области хранения данных |

### Директива ORG

Оператор ORG принимает форму:

метка ORG выражение

где "метка" - дополнительная метка программы, и "выражение" - 16-разрядное выражение, состоящее из операндов, которые определены перед оператором ORG. Ассемблер начинает генерацию машинного кода с места, определенного в выражении. Может быть любое число операторов ORG в конкретной программе. Однако, не выполняется никаких проверок, гарантирующих, что программист не определяет перекрывающиеся области памяти. Обратите внимание, что большинство программ, написанных для системы MP/M II, начинается со следующего оператора ORG:

ORG 100H

Он запускает генерацию машинного кода с начала переходных области транзитных программ MP/M II. Чтобы подготовить странично-перемещаемую программу для выполнения в MP/M II, соберите исходную программу дважды, добавив 100H к каждому оператору ORG во время второго ассемблирования. Соедините два шестнадцатеричных файла, сгенерированные ассемблером, используя PIP, затем представьте составной файл утилите GENMOD, которая производит файл с расширением PRL.

Если метка определена в операторах ORG, то метке присваивается значение выражения. Эта метка может затем использоваться в поле операнда других операторов для представления этого выражения.

### Директива END

Оператор END необязательный в программе на языке ассемблера, но если он присутствует, он должен быть последним оператором, потому что все последующие операторы игнорируются. Две формы директивы END следующие:

метка END

метка END выражение

в которых метка не является обязательной. Если используется первая форма, процесс ассемблирования останавливается и начальный адрес начала программы по умолчанию принимается равным 0000. Иначе, вычисляется выражение и его значение становится начальным адресом программы. Этот начальный адрес включается в последнюю запись файла с машинным кодом в формате Intel hex, который выдается при ассемблировании. Таким образом большинство программ на языке ассемблера CP/M заканчивается оператором:

END 100H

в результате чего начальный адрес по умолчанию 100H.

### Директива EQU

Оператор EQU (приравнивания) устанавливает синонимы для конкретных числовых значений. Имеет форму:

метка EQU выражение

где метка должна присутствовать и не должна помечать никакие другие операторы. Ассемблер вычисляет выражение и присваивает его значение идентификатору, заданному в поле метки. Идентификатор обычно является именем, которое описывает значение более понятным человеку способом. Затем, это имя может использоваться всюду в программе, чтобы "параметризовать" некоторые функции. Например, предположим, что данные, полученные от телетайпа появляется в определенном порту ввода, и данные передаются в телетайп через следующий номер порта вывода. Несколько операторов приравнивания может определить эти порты для конкретной аппаратной среды, как показано ниже:

TTYBASE EQU 10H ; Начальный номер портов TTY

TTYIN EQU TTYBASE ; Входные данные TTY

TTYOUT EQU TTYBASE+1 ; Выходные данные TTY

Позже в программе, операторы, которые получают доступ к телетайпу, могут выглядеть как показано ниже:

IN TTYIN ; Чтение данных из TTY в регистр A

. . .

OUT TTYOUT ; Запись данных в TTY из регистра A

Они делают программу более читаемой, чем если бы использовались абсолютные порты ввода-вывода. Кроме того, если вы переопределяете аппаратную среду, в которой коммуникационные порты телетайпа начинаются в 7FH вместо 10H, вы должны изменить только первый оператор:

TTYBASE EQU 7FH ; Начальный номер порта TTY

и программа может быть повторно ассемблирована, без изменения других операторов.

### Директива SET

Оператор SET подобен EQU, принимая форму:

метка SET выражение

Он отличается от SET, в котором метка может появляться в других операторах SET в программе. Выражение вычисляется и становится текущим значением, связанным с меткой. Таким образом оператор EQU определяет метку с помощью единственного значения, в то время как оператор SET определяет значение, которое допустимо от текущего оператора SET до следующего оператора SET, в котором появляется метка. чаще всего SET используется для управления условным ассемблированием.

### Директивы IF и ENDIF

Операторы IF и ENDIF определяют область операторов ассемблера, которые будут включены или исключены во время процесса ассемблирования. Форма оператора:

IF выражение

оператор1

оператор2

. . .

операторn

ENDIF

После обнаружения оператора IF ассемблер вычисляет выражение после IF. Все операнды в выражении должны быть определены до оператора IF. Если младший значащий бит вычисленного выражения равен 1, то оператор1 ... операторn ассемблируются. Если младший значащий бит вычисленного выражения равен 0, то операторы распечатываются, но не ассемблируются. Условное ассемблирование часто используется для написания одной "универсальной" программы, которая включает в себя целый ряд возможных сред времени выполнения только с несколькими определенными частями программы, выбранными для конкретного ассемблирования. Например, следующие сегменты программы могут быть частью программы, которая взаимодействует с телетайпом или консолью CRT (но не обоими), выбрав конкретное значение TTY перед началом ассемблирования.

TRUE EQU 0FFFFH ; Определение значения TRUE

FALSE EQU NOT TRUE ; Определение значения FALSE

TTY EQU TRUE ; TRUE если TTY, FALSE если CRT

TTYBASE EQU 10H ; Начальный номер портов ввода-вывода TTY

CRTBASE EQU 20H ; Начальный номер портов ввода-вывода CRT

IF TTY ; Ассемблировать относящееся к TTYBASE

CONIN EQU TTYBASE ; Ввод с телетайпа

CONOUT EQU TTYBASE+1 ; Вывод на телетайп

ENDIF

IF NOT TTY ; Ассемблировать относящееся к CRTBASE

CONIN EQU CRTBASE ; Ввод с консоли

CONOUT EQU CRTBASE+1 ; Вывод на консоль

ENDIF

. . .

IN CONIN ; Прочитать данные из порта

. . .

OUT CONOUT ; Записать данные в порт

В этом случае программа собирается для среды, где телетайп соединяется, портом начинающимся в 10H. Оператор, определяющий TTY, может быть изменен на:

TTY EQU FALSE

и в этом случае программа собирается для CRT, начинающегося в порту 20H.

### Директива DB

Директива DB позволяет программисту определять инициализированные области хранения в формате одинарной точности (байт). Форма оператора следующая:

метка DB e1, e2, ..., en

где e1 до en - выражения, результатом вычисления которых являются 8-битные значения (старшие восемь битов должно быть равно нулю), или строки ASCII длиной не более 64 символов. Не существует никакого практического ограничения на число выражений, включенных в одну исходную строку. Выражения вычисляются и последовательно помещаются в файл машинного кода, начиная с текущего адреса программы, сгенерированного ассемблером. Символы строки аналогично помещаются в память, начиная с первого символа и заканчивая последним символом. Строки длиной более двух символов, не могут использоваться в качестве операндов в более сложных выражениях. То есть они должны стоять отдельно между запятыми. Обратите внимание, что символы ASCII всегда помещаются в память со сброшенным битом четности (0), и отсутствует преобразование строк их нижнего в верхний регистр. Необязательная метка позволяет ссылаться на область данных, на протяжении оставшейся части программы. Примеры допустимых операторов DB:

data: DB 0,1,2,3,4,5

DB data and 0ffh,5,377Q,1+2+3+4

signon: DB 'please type your name',cr,lf,0

DB 'AB' SHR 8, 'C', 'DE' AND 7FH

### Директива DW

Оператор DW похож на оператор DB за исключением того, что он инициализирует области хранения двухбайтовые слова вместо одиночных байтов. Его форма следующая:

метка DW e1, e2, ..., en

где e1 до en - выражения, результатом вычисления которых являются 16-разрядные значения. Обратите внимание, что строки ASCII из одного или двух символов допустимы, но строки содержащие более двух символов не приемлемы. Во всех случаях область хранения данных согласуется с процессорами 8080: первым в памяти сохраняется младший значимый байт выражения, затем старший значимый байт. Вот некоторые примеры:

doub: DW 0ffefh,doub+4,signon-$,255+255

DW 'a',5, 'ab','CD',6 shl 8 or 11b

### Директива DS

Оператор DS резервирует область неинициализированной памяти и принимает форму:

метка DS выражение

где метка необязательная. Ассемблер начинает последующую генерацию кода после области, зарезервированной DS. Таким образом оператор DS, приведенный выше, оказывает точно такой же эффект как следующие операторы:

метка: EQU $ ; Значением метки является код

; текущего местоположения

ORG $+выражение ; Пропустить зарезервированную область

## Коды операций

Коды операций ассемблера являются основной частью программ на языке ассемблера и формируют поле операций команд. В общем ASM принимает все стандартные мнемоники для микрокомпьютера Intel 8080, которые подробно описаны в руководстве Intel "Руководство по программированию на языке ассемблера 8080". Если в каждой входной строке присутствуют необязательные метки, они принимают значение адреса команды непосредственно перед выдачей команды. *Таблица 4-7* кратко перечисляет отдельные операторы, но Вы должны обратиться к руководству Intel для получения подробных описаний.

В *Таблице 4-7* перечислены каждый код операции в самом общем виде с конкретным примером, затем дается краткое объяснение с любыми особыми ограничениями. В столбце формы "en" символизируют выражение. *Таблица 4-6*, ниже, определяет символы "en".

Таблица ‑ Символы выражений

|  |  |
| --- | --- |
| **Символ** | **Представление** |
| e3 | 3-разрядное значение в диапазоне 0-7, который может быть одним из предопределенных регистров A, B, C, D, E, H, L, M, SP или PSW. |
| e8 | 8-разрядное значение в диапазоне 0-255. |
| e16 | 16-разрядное значение в диапазоне 0-65535. |

Выражения могут быть сформированы из произвольной комбинации операндов и операторов. В некоторых случаях операнды ограничены конкретными значениями в пределах допустимого диапазона, такие как инструкция PUSH. *Таблица 4-7* отмечает такие случаи, если они имеются.

Коды операций, кратко изложены в *Таблице 4-7* делятся на шесть категорий. *переходы, вызовы и инструкции возврата* могут протестировать состояние флагов установленных в ЦП и передать управление в другое место. Инструкции *непосредственной загрузки операнда* одинарной или двойной точности регистров или ячеек памяти одиночной точности постоянными значениями. Они также включают в себя инструкции, которые выполняют непосредственные *арифметические или логические операции* в аккумуляторе (регистре A).

Инструкции *увеличения и уменьшения на единицу* обеспечены для регистров одинарной и двойной точности. Инструкции *перемещения данных* передают данные из памяти в ЦП и от ЦП в память. Инструкции арифметического логического устройства выполняют арифметические и логические операции одинарной точности в аккумуляторе. *Инструкции управления* включают и отключают прерывания, выполняют остановку программы и функцию без операции.

Таблица ‑ Коды операций ASM

| **Форма** | **Пример** | **Объяснение** |
| --- | --- | --- |
| ***Переходы, вызовы и инструкции возврата*** | | |
| JMP e16 | JMP L1 | Безусловный переход к метке |
| JNZ e16 | JNZ L2 | Переход при ненулевом условии к метке |
| JZ e16 | JZ 100H | Переход при нулевом условии к метке |
| JNC e16 | JNC L1+4 | Переход к метке если нет переноса |
| JC e16 | JC L3 | Переход к метке если есть перенос |
| JPO e16 | JPO $+8 | Переход к метке если паритет нечётный |
| JPE e16 | JPE L4 | Переход к метке если паритет чётный |
| JP e16 | JP GAMMA | Переход к метке если результат положительный |
| JMP e16 | JM a1 | Переход к метке если результат отрицательный |
| CALL e16 | CALL S1 | Вызов подпрограммы безусловный |
| CNZ e16 | CNZ S2 | Вызов подпрограммы если флаг не ноль |
| CZ e16 | CZ 100H | Вызов подпрограммы если флаг ноль |
| CNC e16 | CNC S1+4 | Вызов подпрограммы если не установлен перенос |
| CC e16 | CC S3 | Вызов подпрограммы если установлен перенос |
| CPO e16 | CPO S+8 | Вызов подпрограммы если паритет нечётный |
| CPE e16 | CPE S4 | Вызов подпрограммы если паритет чётный |
| CP e16 | CP GAMMA | Вызов подпрограммы если флаг положительный |
| CM e16 | CM b1$c2 | Вызов подпрограммы если флаг отрицательный |
| RST e3 | RST 0 | Программный "перезапуск", эквивалентный CALL 8\*e3, за исключением однобайтной команды |
| RET |  | Возврат из подпрограммы |
| RNZ |  | Возврат если флаг не ноль |
| RZ |  | Возврат если флаг ноль |
| RNC |  | Возврат если не установлен перенос |
| RC |  | Возврат если установлен перенос |
| RPO |  | Возврат если паритет нечётный |
| RPE |  | Возврат если паритет чётный |
| RP |  | Возврат если результат положительный |
| RM |  | Возврат если результат отрицательный |
| ***Инструкции непосредственной загрузки*** | | |
| MVI e3, e8 | MVI B,255 | Непосредственная пересылка данных в регистры A, B, C, D, E, H, L или M (память) |
| ADI e8 | ADI 1 | Непосредственное сложение операнда с аккумулятором без переноса |
| ACI e8 | ACI OFFH | Непосредственное сложение операнда с аккумулятором с переносом |
| SUI e8 | SUIL + 3 | Вычитание из регистра A без заёма (перенос) |
| SBI e8 | SBI L AND 11B | Вычитание из регистра A с заёмом (перенос) |
| ANI e8 | ANI $ AND 7FH | Логическое "И" непосредственных данных с A |
| XRI e8 | XRI 1111$0000B | "Исключающее ИЛИ" регистра A с непосредственными данными |
| ORI e8 | ORI L AND 1+1 | Логическое "ИЛИ" регистра A с непосредственными данными |
| CPI e8 | CPI 'a' | Сравнение регистра A с непосредственными данными (аналогично SUI, но без изменения содержимого регистра A) |
| LXI e3, e16 | LXI B,100H | Непосредственная загрузка в регистровую пару (e3 должен быть эквивалентным B, D, H или SP) |
| ***Инструкции инкремента (увеличения на 1) и декремента (уменьшения на 1)*** | | |
| INR e3 | INR E | Инкремент регистра одинарной точности (e3 означает один из регистров A, B, C, D, E, H, L, M) |
| DCR e3 | DCR A | Декремент регистра одинарной точности (e3 означает один из регистров A, B, C, D, E, H, L, M) |
| INX e3 | INX SP | Инкремент регистровой пары двойной точности (e3 должен быть эквивалентным B, D, H или SP) |
| DCX e3 | DCX B | Декремент регистровой пары двойной точности (e3 должен быть эквивалентным B, D, H или SP) |
| ***Инструкции перемещения данных*** | | |
| MOV e3, e3 | MOV A,B | Перемещение данных в левый элемент из правого элемента (e3 означает один из регистров A, B, C, D, E, H, L или M). Инструкция MOV M, M запрещена |
| LDAX e3 | LDAX B | Загрузка регистра A из памяти по вычисленному адресу (e3 должен быть B или D) |
| STAX e3 | STAX D | Запись регистра A в память по вычисленному адресу (e3 должен быть B или D) |
| LHLD e16 | LHLD L1 | Прямая загрузка HL из памяти по адресу e16 (загрузка двойной точности в H и L) |
| SHLD e16 | SHLD L5+x | Прямая запись HL в память по адресу e16 (запись двойной точности из H и L в память) |
| LDA e16 | LDA Gamma | Загрузка регистра A из адреса e16 |
| STA e16 | STA X3-5 | Запись регистра A в память по адресу e16 |
| POP e3 | POP PSW | Извлечение из стека 2 байт данных и помещение их в регистровую пару или в РSW (e3 должен быть одним из B, D, Н или PSW) |
| PUSH e3 | PUSH B | Запись регистровой пары в стек, установка SP (e3 должен быть одним из B, D, Н или PSW) |
| IN e8 | IN 0 | Прием данных в регистр A из порта e8 |
| OUT e8 | OUT 255 | Посылка данных из регистра A в порт e8 |
| XTHL |  | Обмен содержимым между вершиной стека и регистровой парой HL |
| PCHL |  | Пересылка регистровой пары HL в программный счетчик |
| SPHL |  | Пересылка регистровой пары HL в указатель стека |
| XCHG |  | Обмен содержимого регистровых пар DE и HL |
| ***Арифметические или логические операции*** | | |
| ADD e3 | ADD B | Сложение содержимого регистра A с содержимым регистра обусловленного e3 без учета переноса (e3 означает один из регистров A, B, C, D, E, H или L) |
| ADC e3 | ADC L | Сложение A с содержимым регистра обусловленного e3 с учетом переноса (e3 означает один из регистров A, B, C, D, E, H или L) |
| SUB e3 | SUB H | Вычитание из A содержимого регистра обусловленного e3 без учета переноса (e3 означает один из регистров A, B, C, D, E, H или L) |
| SBB e3 | SBB 2 | Вычитание из A содержимого регистра обусловленного e3 с учетом переноса (e3 означает один из регистров A, B, C, D, E, H или L) |
| ANA e3 | ANA 1+1 | Логическое "И" регистра обусловленного e3 с A (e3 означает один из регистров A, B, C, D, E, H или L) |
| XRA e3 | XRA A | "Исключающее ИЛИ" регистра обусловленного e3 с A (e3 означает один из регистров A, B, C, D, E, H или L) |
| ORA e3 | ORA B | Логическое "ИЛИ" регистра обусловленного e3 с A (e3 означает один из регистров A, B, C, D, E, H или L) |
| CMP e3 | CMP | Сравнение A с регистром обусловленным e3 (e3 означает один из регистров A, B, C, D, E, H или L) |
| DAA |  | Десятичная коррекция регистра A после последней арифметической операции |
| CMA |  | Инверсия содержимого аккумулятора |
| STC |  | Установка флага переноса в 1 |
| CMC |  | Инверсия флага переноса |
| RLC |  | Циклический сдвиг содержимого аккумулятора влево, побочным эффектом является возможное изменение флага переноса (старший разряд регистра A становится флагом переносом) |
| RRC |  | Циклический сдвиг содержимого аккумулятора вправо, побочным эффектом является возможное изменение флага переноса (младший разряд регистра A становится флагом переносом) |
| RAL |  | Арифметический сдвиг содержимого аккумулятора влево (сдвиг через перенос) |
| RAR |  | Арифметический сдвиг содержимого аккумулятора вправо (сдвиг через перенос) |
| DAD e3 | DAD B | Сложение двойной точности регистровой пары e3 с HL (e3 должен быть B, D, H или SP) |
| ***Инструкции управления*** | | |
| HLT |  | Остановка процессора 8080 |
| DI |  | Отключение системы прерываний |
| EI |  | Включение системы прерываний |
| NOP |  | Пустая операция (нет операций) |

## Сообщения об ошибках

Когда ASM находит ошибки в программе на языке ассемблера, он перечисляет их как одно-символьные коды в крайней левой позиции исходного листинга. Строка с ошибкой также отображается в консоли так, что исходный листинг не нужно изучать для определения присутствия ошибок. *Таблица 4-8* определяет коды ошибок.

Таблица ‑ Коды ошибок ассемблера

|  |  |
| --- | --- |
| **Код** | **Значение** |
| D | Ошибка данных: элемент данных оператора не может быть размещен в указанной области данных |
| E | Ошибка выражения: выражение плохо сформировано и не может быть вычислено |
| L | Ошибка метки: метка не может появиться в этом контексте, может быть двойная метка |
| N | Не реализовано: возможности, которые появятся в будущих версиях ASM (например, макросы) распознаны и отмечены, но не поддерживаются в этой версии |
| O | Переполнение: выражение слишком сложное, имеет слишком много незаконченных операторов, чтобы быть вычисленным. Упростите его |
| P | Ошибка фазы: метки не имеют то же значение на двух последовательных проходах через программу |
| R | Ошибка регистра: значение, определенное в качестве регистра, не совместимо с кодом операции |
| S | Синтаксическая ошибка: поля этого оператора плохо сформированы и не могут быть обработаны правильно. Может произойти из-за недопустимых символов или неуместных разделителей |
| U | Неопределенный символ: метка операнда в этом операторе не появилась в другом месте с левой стороны оператора, который генерирует машинный код или резервирует пространство памяти, как инструкция MOV, псевдо операция DS или директивы EQU или SET |
| V | Ошибка значения: операнд, встреченный в выражении, неправильно сформирован |

Некоторые сообщения об ошибке могут быть напечатаны в терминале, если возникает условие ошибки диска. *Таблица 4-9* перечисляет эти сообщения об ошибках.

Таблица ‑ Сообщения терминала ASM

|  |  |
| --- | --- |
| **Сообщение** | **Значение** |
| NO SOURCE FILE PRESENT | Файл, указанный в команде ASM не существует на диске. |
| NO DIRECTORY SPACE | Каталог диск заполнен. Удалите ненужные файлы, затем повторите попытку. |
| SOURCE FILE NAME ERROR | Неправильно сформировано имя файла ASM. Подстановочные символы ? и \* недопустимы. |
| SOURCE FILE READ ERROR | Исходный файл не может быть прочитан должным образом ассемблером. Введите файл в консоли для определения места ошибки. |
| OUTPUT FILE WRITE ERROR | Выходные файлы не могут быть записаны должным образом. Самая вероятная причина - полный диск. Удалите лишнее и повторите попытку. |
| CANNOT CLOSE FILE | Выходной файл не может быть закрыт. Проверьте, не защищен ли диск от записи. |

# RDT

## Обзор RDT

Перемещаемый инструмент отладки (RDT) позволяет пользователю тестировать и отлаживать программы в среде MP/M II. Несколько RDT может быть перемещено для выполнения в системе без коммутации банков или назначены абсолютные ячейки памяти для выполнения в системе с коммутацией банков. Команды RDT - надмножество CP/M отладчика DDT (см. список команд в *Таблице 5-1*). Дополнительные команды позволяют RDT отлаживать перемещаемый код и сохранять исправленные программы. Однако есть одно важное различие между RDT и DDT. RDT является файлом PRL, а DDT - файлом COM. Таким образом RDT может отлаживать файлы COM и PRL, в то время как DDT может отлаживать только файлы COM. Примечание: RDT не может прочитать файл, защищенный паролем.

## Вызов RDT

Вызовите RDT, введя одну из следующих команд:

RDT

RDT filespec

Первая команда просто загружает и выполняет RDT. После отображения его сообщения входа в систему и символа приглашения, RDT готов принять команды оператора. Вторая команда подобна первой, за исключением того, что после загрузки, RDT загружает файл, определенный спецификацией файла.

Вторая команда эквивалентна первому вызову RDT с последующим использовании команды "I" (Input), чтобы вставить имя файла в FCB по умолчанию по адресу BASE+005CH, как показано в следующей последовательности.

0A>**RDT**

00:22:55 A:RDT .PRL (USER n)

[MP/M] DDT VERS 2.0

NEXT PC

0100 0100

-**I filespec**

-**R**

-

В этой точке программа, названная filespec, загружена и готова к отладке.

## Соглашения о записи команд RDT

Когда RDT готов принять команду, он выводит пользователю приглашение в виде дефис "-". В ответ пользователь может ввести командную строку или Ctrl-C (представленный как ^C), чтобы закончить сеанс отладки (см. [Раздел 5.4](#_Завершение_RDT)). Командная строка может иметь до 32 символов и должна быть завершена возвратом каретки. При вводе команд используйте стандартные MP/M II функции редактирования строки (^X, ^H, и т.д.) для исправления опечаток. RDT не обрабатывает командную строку, пока не введен возврат каретки.

Первый символ каждой командной строки определяет действие команды. *Таблица 5-1* перечисляет команды RDT. Команды RDT описаны индивидуально в [Разделе 5.5](#_Команды_RDT).

Таблица ‑ Список команд RDT

|  |  |
| --- | --- |
| **Команда** | **Действие** |
| A | Ввод операторов ассемблера |
| \*B | Установка или сброс битов битового массива |
| D | Вывод на консоль содержимого памяти в шестнадцатеричном коде и ASCII |
| F | Заполнение блока памяти константой |
| G | Начать выполнение с необязательными контрольными точками |
| I | Задание имени файла для загрузки и хвоста команды (настройка блока управления файлом) |
| L | Вывод содержимого памяти в мнемокоде ассемблера |
| M | Перемещение блока памяти |
| \*N | Нормализация и перемещение программы в сегмент памяти RDT |
| R | Чтение программы с диска для последующего выполнения |
| S | Установка памяти нового значения |
| T | Трассировка выполнения программы |
| U | Неотслеженный мониторинг программы |
| \*V | вычисление значения параметра для команды W |
| \*W | Запись содержимого блока памяти на диск |
| X | Исследование и изменение состояния ЦП |
| \*Только в RDT | |

Символ команды может сопровождаться одним или более параметрами, которые могут быть шестнадцатеричными значениями, именами файлов или другой информацией, в зависимости от команды. RDT предполагает, что все значения, вводимые пользователем, шестнадцатеричные. Если пользователь вводит больше чем четыре цифры, RDT усекает их слева, т.е. RDT использует только последние четыре. Параметры должны быть разделены друг от друга запятыми или пробелами. Примечание: пробелы между символом команды и первым параметром не допускаются.

## Завершение RDT

Пользователь завершает RDT, вводя ^C в ответ на приглашение в форме дефиса "-". RDT отвечает запросом:

Abort (Y/N) ?

Примечание: в MP/M II нет средства SAVE имеющегося в CP/M поэтому, если RDT используется для исправления файла, пользователь должен записать файл на диск, используя команду W прежде, чем завершить RDT.

## Команды RDT

Этот раздел описывает команды RDT и их параметры. Команды RDT дают возможность пользователю управлять выполнением программы и позволяют пользователю отображать и изменять системную память и состояние ЦП.

### Команда A (ассемблирование)

Команда A ассемблирует мнемоники 8080 непосредственно в память. Форма команды:

Aa

где a - шестнадцатеричный адрес с которого начинается ассемблирование. RDT отвечает на команду A, выводя на экран адрес ячейки памяти, где ассемблирование должно начаться. В этот момент, пользователь вводит операторы ассемблера. Когда оператор введен, RDT преобразует его в машинный код, помещает значение(я) в память и выводит на экран адрес следующей доступной ячейки памяти. Этот процесс продолжается, пока пользователь не вводит пустую строку или строку, содержащую только точку.

RDT реагирует на недопустимые операторы, отображая вопросительный знак "?" и повторно отображает текущий адрес ассемблирования.

### Команда B (Установка/сброс битов битового массива)

Команда B позволяет пользователю обновить битовый массив страницы перемещаемого файла. Пользователь читает файл, вносит изменения в код, и затем определяет байты, которые необходимо переместить (т.е. старшие адресные байты команд перехода). Затем пользователь обновляет битовый массив с помощью команды B. Есть два определяемых параметра: адрес, который будет изменен с последующим 0, чтобы сбросить байт, ранее отмеченный для перемещения или 1, чтобы отметить байт для перемещения. Команда имеет форму:

Ba,n

где a - шестнадцатеричный адрес и n - 0 или 1.

### Команда D (Отображение)

Команда D выводит на экран содержание памяти как 8-разрядные шестнадцатеричные значения и в символах ASCII. Формы команды следующие:

D

Ds

Ds,f

где s - шестнадцатеричный адрес, с которого начинается отображение, и f - адрес, где отображение должно закончиться. В ответ на первую форму, показанную выше, RDT отображает память с текущего адреса отображения и выводит на экран 16 строк. Ответ на вторую форму аналогичен первой, за исключением того, что первому адрес отображения сначала присваивается адрес s. Третья форма выводит на экран блок памяти между расположениями s и f.

Память отображается на экран на одной или более строках дисплея. Каждая строка дисплея показывает значения до 16 ячеек памяти.

Для первых трех форм отображаемая строка появляется в следующем виде:

aaaa bb bb . . . bb cc . . . c

где aaaa - адрес отображения в шестнадцатеричной форме. bb представляет содержание ячеек памяти в шестнадцатеричной форме, и c представляет содержание памяти в символах ASCII. Любые невизуализуемые символы ASCII представлены в виде точек.

Во время длинного отображения введите любой символ в консоли, чтобы прервать команду D.

### Команда F (Заполнение)

Команда F заполняет область памяти постоянным байтом. Форма команды:

Fs,f,b

где s является шестнадцатеричный начальным адресом заполняемого блока, f - конечный адрес, и b является шестнадцатеричной байтовой константой. RDT сохраняет 8-разрядное значение b в ячейках памяти от s до f с первым местом хранения b по адресу s, а затем увеличивает значение s и сравнивает его с f. Процесс повторяется до тех пор, пока s не превысит f.

### Команда G (Перейти к выполнению)

Команда G передает управление тестируемой программе и дополнительно устанавливает одну или две контрольных точки. Формы команды:

G

G,b1

G,b1,b2

Gs

Gs,b1

Gs,b1,b2

где s - шестнадцатеричный адрес начала выполняемой программы, и b1 и b2 - шестнадцатеричные адреса контрольных точек (точек останова).

В первых трех формах начальный адрес не указан, поэтому RDT начинается выполнение программы в режиме тестирования с текущего значения программного счетчика. (Команда X служит для определения текущего значения счетчика). Первая форма передает управление программе пользователя без установки каких-либо точек останова. Следующие две формы соответственно устанавливают одну и две точки останова перед передачей управления программе пользователя. Следующие три формы аналогичны первым трем, за исключением того, что сначала устанавливается программный счетчик пользователя s.

Как только управление передано тестируемой программе, она выполняется в режиме реального времени, пока не встречаются с контрольной точкой. В этот момент RDT восстанавливает управление, очищает все контрольные точки и показывает адрес, в котором выполнение тестируемой программы было прервано следующим образом:

\*aaaa

где aaaa - шестнадцатеричный адрес, где произошло прерывание. Когда контрольная точка возвращает управление RDT, инструкция по адресу контрольной точки еще не была выполнена.

### Команда I (Ввод хвоста команды)

Команда I вставляет имя файла в FCB по умолчанию в Base+005CH, относительно начала сегмента, в который загружен RDT. Форма команды:

I<command tail>

где <command tail> является символьной строкой, которая обычно содержит одно или более имен файлов. Первое имя файла разбирается в FCB по умолчанию в Base+005CH. Необязательное второе имя файла (если определено) разбирается во вторую часть FCB по умолчанию, начинающуюся в Base+006CH.

### Команда L (Список)

Команда L выводит содержимое памяти на языке ассемблера. Формы команды:

L

Ls

Ls,f

где s - шестнадцатеричный адрес, с которого должен начаться вывод, и f - шестнадцатеричный адрес, где вывод должен закончиться.

Первая форма выдает 11 строк дизассемблированного машинного кода начиная с текущего адреса. Вторая форма устанавливает адрес вывода в s и затем выводит 11 строк кода. Последняя форма выводит дизассемблированный кода с адреса s по адрес f. Во всех трех случаях адрес вывода устанавливается равным следующему не выведенному адресу в подготовке к последующей команде L. Когда RDT восстанавливает управление из протестированной программы (см. команды G, T и U), адресу вывода присваивается текущее значение программного счетчика.

Прервать длинный вывод можно нажав любую клавишу во время процесса вывода. Или, введя ^S, чтобы временно остановить вывод.

### Команда M (Перемещение)

Команда M перемещает значения блока данных от одной области памяти в другую. Форма команды:

Ms,f,d

где s - начальный шестнадцатеричный адрес пересылаемого блока. f - адрес заключительного байта, который будет перемещен, и d - адрес назначения, который получит данные первого байта. Примечание: если d между s и f, часть перемещаемого блока будет перезаписана, прежде чем он будет перемещен, потому что данные передаются, начиная с местоположения s.

### Команда N (Нормализовать)

Команда N корректирует перемещаемые адреса страницы перемещаемого файла, который RDT читает в память. Пользователь читает файл в память с помощью команды R, и затем использует команду N, чтобы подготовить файл к выполнению в сегменте памяти, в котором выполняется RDT. Форма команды:

N

### Команда R (Чтение)

Команда R используется в сочетании с командой I, чтобы считать файлы из диска в TPA в подготовке к отладке. Формы команды:

R

Rb

где b - необаятельный адрес смещения, который добавляется к каждому адресу программы или данных по мере его загрузки. Операция загрузки не должна перезаписывать ни один из системных параметров в ячейках от 000H до 0FFH (т.е., базовую страницу TPA, в которую загружен RDT). Если b опущен, RDT принимает b=0000H. Команда R требует предварительного выполнения команды I, которая определяет допустимое имя файла. Адрес загрузки для каждой записи из HEX файла получается из каждой отдельной HEX записи. RDT предполагает, что любой файл, указанный с расширением HEX содержит машинный код в формате Intel hex. Другие файлы рассматриваются в чисто двоичном формате.

Пользователь может выдать любое число команд R после команды I, чтобы перечитать отлаживаемую программу, если программа не уничтожает FCB по умолчанию в Base+005CH.

Напомним, что последовательность:

0A>**RDT**

-**Ifilespec**

-**R**

эквивалентна:

0A>**RDT filespec**

Когда пользователь дает команду R, RDT отвечает сообщением загрузки в форме:

NEXT PC

nnnn pppp

где nnnn - следующий адрес после загруженной программы, и pppp - предполагаемый программный счетчик, взятый из последней записи, если определен HEX файл. Для других файлов предполагается, что это начало TPA.

### Команда S (Установка)

Команда S позволяет исследовать или изменить содержание отдельного байта в памяти. Форма команды:

Sa

где a означает исследуемый или изменяемый шестнадцатеричный адрес.

RDT отображает адрес памяти и его текущее содержание на следующей строке в следующей форме:

aaaa bb

где aaaa - шестнадцатеричный адрес, и bb - содержание байта памяти в шестнадцатеричном формате. Пользователь может принять решение изменить ячейку памяти или оставить ее неизменной. Если пользователь вводит допустимое шестнадцатеричное значение, RDT заменяет содержание байта в памяти на новое значение. Если значение не введено, содержимое памяти не изменяются, и отображается содержание следующего адреса. В любом случае RDT продолжает отображать последовательные адреса памяти и значения, пока не введена точка или недопустимое значение.

### Команда T (Трассировка)

Команда T прослеживает выполнение от 1 до 0FFFFH шагов программы. Формы команды:

T

Tn

где n число выполненных инструкций, прежде чем вернуть управление консоли.

В ответ на первую форму RDT выводит на экран состояние ЦП и выполняет один шаг программы. Программа сразу завершается с адресом завершения, выведенным на экран в форме:

\*hhhh

где hhhh является адресом следующей команды. Пользователь задает отображаемый адрес (используемый в команде D) равным значению регистров HL, и присваивает адресу вывода (используемый в команде L) значение hhhh. Пользователь может затем исследовать состояние ЦП при завершении программы с помощью команды X.

Вторая форма аналогична первой, за исключением того, что RDT прослеживает выполнение программы для n шагов, прежде чем контрольная точка произойдет. Пользователь может вызвать контрольную точку в режиме трассировки, введя символ Rubout (возврат с удалением). RDT снова выводит на экран состояние ЦП перед каждым шагом программы в том же формате, как описано в команде X.

В любом случае RDT передает управление тестируемой программе по адресу, обозначенному программным счетчиком. Если пользователь не определяет n, RDT выполняет только одну инструкцию. Пользователь может прервать длинную трассировку, прежде чем n шагов будут выполнены, вводя любой символ в консоли.

Примечание: Трассировка программы останавливается на границе с MP/M II и возобновляется после возвращения из MP/M II к тестируемой программе. Таким образом, функции MP/M II, осуществляющие доступ к устройствам ввода-вывода, таким как дисководы, выполняются в режиме реального времени и избегают проблем синхронизации ввода-вывода. Программы, запущенные в режиме трассировки выполняются примерно в 500 раз медленнее, чем в реальном времени, так как RDT получает управление после выполнения каждой команды.

### Команда U (Без трассировки)

Команда U аналогична команде T, за исключением того, что состояние ЦП отображаются только до выполнения первой инструкции, а не перед каждым шагом. Формы команды:

U

Un

где n - число команд, которые необходимо выполнить перед возвратом управления к консоли. Команда U прерывается, нажатием любой клавиши в консоли.

### Команда V (Значение)

Команда V упрощают использование команды W, вычисляя параметр для следующей команды "W". Один параметр следует сразу за "V", который является NEXT (следующим) местоположением после последнего байта для записи на диск.

Как правило, пользователь читает из файла, редактирует его, а затем записывает его на диск. Команда чтения R, производит значение для NEXT. Это значение может быть введено в качестве параметра после команды V, и RDT вычислит и отобразит количество секторов для записи с помощью команды W. Форма команды:

V

### Команда W (Запись)

Команда W записывает содержание непрерывного блока памяти на диск. Форма команды:

Wn

где n - значение параметра, полученное в команде V, и является числом секторов, которые будут записаны в диск. Пользователь вводит значение n в шестнадцатеричном формате.

### Команда X (Изучение состояния ЦП)

Команда Х позволяет пользователю просматривать и изменять состояние процессора тестируемой программы. Формы команды:

X

Xr

Xf

где r - имя одного из регистров ЦП 8080, и f - сокращение одного из флагов ЦП. Первая форма выводит на экран состояние ЦП в формате:

CfZfMfEfIf A=bb B=dddd D=dddd H=dddd S=dddd P=dddd inst

где f - значение флагов, 0 или 1. bb - значение байта, и dddd - два байта, соответствующие регистровой паре. Поле "inst" содержит дизассемблированную инструкцию, которая возникает в расположении, адресуемом программным счетчиком состояния ЦП.

Вторая форма выводит на экран и позволяет пользователю изменять значения регистра, где r - один из регистров, перечисленных в *Таблице 5-2*.

Третья форма выводит на экран и позволяет пользователю изменять значения флагов, перечисленных в *Таблице 5-2*.

Таблица ‑ Флаги и регистры ЦП 8080

|  |  |  |
| --- | --- | --- |
| **Флаг** | **Определение** | **Значение** |
| C | Флаг переноса | (0/1) |
| Z | Флаг нуля | (0/1) |
| M | Флаг минус | (0/1) |
| E | Флаг четности | (0/1) |
| I | Флаг внутреннего переноса | (0/1) |
| **Регистр** | **Определение** | **Значение** |
| A | Аккумулятор | (0-FF) |
| B | Регистровая пара BC | (0-FFFF) |
| D | Регистровая пара DE | (0-FFFF) |
| H | Регистровая пара HL | (0-FFFF) |
| S | Указатель стека | (0-FFFF) |
| P | Программный счетчик | (0-FFFF) |

В каждом случае RDT сначала выводит на экран флаг или значение регистра, и затем принимает ввод в консоли. Если пользователь вводит значение в надлежащий диапазон, RDT изменяет флаг или регистр. Ввод возврата каретки не изменяет значение. Примечание: RDT выводит на экран регистры BC, DE и HL как регистровые пары. Таким образом ввод B изменяет регистровую пару BC, D изменяет регистровую пару DE и т.д.

# Другие программные утилиты

## GENHEX

Синтаксис:

genhex{d:}filename{.typ}xxx

Утилита GENHEX принимает в качестве входных данных файл COM и преобразует его в файл HEX. Эта утилита является полезной для генерации HEX файлов в качестве входных данных для утилиты GENMOD.

Если расширение файла не определено, утилита GENHEX предполагает расширение COM. В строке синтаксиса выше, xxx - смещение, определенное для HEX файла. GENHEX неразрушающая. Т.е. она не изменяет исходный файл COM. Ниже приведен пример команды GENHEX:

0A>**GENHEX B:PROGRAM.COM 100**

## GENMOD

Синтаксис:

genmod {d:}filename.hex{d:}filename.prl$nnnn

Утилита GENMOD производит файл PRL из HEX файла. Пользователь сначала связывает два HEX файла, сгенерированные из одного исходного файла. HEX файлы смещены друг от друга на 100H байт. Утилита GENMOD принимает составной файл в качестве входных данных, и затем готовит файл PRL, генерируя запись заголовка, сегмент кода и данных и битовую карту.

В приведенной выше строке синтаксиса nnnn - это необязательный параметр, который может использоваться для указания дополнительного количества памяти, требуемого программе за пределами области кода. Форма параметра – 11$11 за которыми следуют четыре HEX цифры. Например, если программа написана с использованием всей доступной памяти для буферов, указание необязательного параметра обеспечивает выделение минимального буфера. Утилита GENMOD является неразрушительной. То есть она не меняет исходный файл HEX. Ниже приведен пример команды GENMOD:

0A>**GENMOD B:PROGRAM.HEX A:PROGRAM.PRL.$1000**

## PRLCOM

Синтаксис:

prlcom{d:}filename.prl{d:}filename.com

Утилита PRLCOM принимает исходный файл PRL и создает файл COM, удаляя запись заголовка и битовую карту. Утилита PRLCOM не является разрушительной. То есть, она не меняет исходный файл PRL. Конечный файл может быть на том же или другом диске, но если он уже существует, PRLCOM запрашивает пользователя:

Destination File Exists, delete (Y/N)?

Ответ “N” прерывает PRLCOM. Ниже приведен пример команды PRLCOM:

0A>**PRLCOM B:PROGRAM.PRL A:PROGRAM.C**

## DUMP

Синтаксис:

dump {d:}filename.typ

Утилита DUMP выводит на экран содержание дискового файла в шестнадцатеричном формате. Ниже приведен пример команды DUMP:

0A>**DUMP PROGRAM.COM**

Имя файла должно быть однозначным (т.е. без подстановочных символов).

Примечание: DUMP, не выводит на экран содержание файла защищенного паролем. DUMP выводит на экран содержание файла на консоль, 16 байтов за один раз с абсолютным адресом байта слева от каждой строки, как показано в следующем примере:

0000 CD 8A 02 1F D2 10 02 CD 58 02 32 64 03 CD D3 02

0010 CD 71 02 43 66 D9 01 57 0E 01 2D F5 05 3A 2E 04

0020 FE CA A2 E5 B3 32 02 E6 45 00 00 00 00 00 00 00

0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

. . . .

.

.

Пользователь может отправить вывод на принтер, введя Ctrl-P прежде, чем ввести команду DUMP, и запустить и остановить вывод на консоль с помощью Ctrl-S/Ctrl-Q. Нажатие любой клавиши, прерывает программу DUMP.

## LOAD

Синтаксис:

load {d:}filename{.typ}

Утилита LOAD принимает в качестве входных данных файл HEX и производит как выходной файл COM, который затем может быть выполнен. LOAD является неразрушающим. То есть она не меняет исходный файл HEX. Если имя файла не указано в командной строке, LOAD ищет на диске файл с расширением HEX. Ниже приведен пример команды LOAD:

0A>**LOAD B:PROGRAM.HEX**

# Создание файла PRL

## Формат PRL

Странично-перемещаемая программа хранятся на диске в виде файла с расширением .PRL. Формат показан в *Таблице 7-1*.

Таблица ‑ Формат файла PRL

|  |  |
| --- | --- |
| **Адрес** | **Содержание** |
| 0001-0002H | Размер программы |
| 0004-0005H | Минимальные требования к буферу (дополнительная память) |
| 0006-00FFH | В настоящее время не используется, зарезервированы для будущего использования |

0100H + Размер программы = Начало битовой карты

*Битовая карта* - последовательность битов, идентифицирующих те байты в исходном коде, которые требуют перемещения. В битовой карте для каждых 8 байтов исходного кода присутствует один байт. Старший значащий бит (бит 7) первого байта битовой карты, указывает требует ли первый байт исходного кода перемещения. Бит установленный в состояние "on" указывает, что перемещение требуется. Следующий бит (бит 6) первого байта соответствует второму байту исходного кода и т.д.

## Генерация PRL

Предпочтительным способом создания .PRL файла является использование Link-80 Digital Research, который способен генерировать .PRL файл из перемещаемого объектного файла REL. Этот способ подробно описан в *Руководстве Link-80*. Пример команды Link показан ниже:

0A>**link dump[op]**

Альтернативным способом создания файла PRL является использование ассемблера Digital Research ASM. Этот способ описан ниже.

Странично-перемещаемая программа может быть сгенерирована, ассемблированием исходного кода дважды. Во время второго ассемблирования, к каждому оператору ORG добавляется 100H. Два HEX файла, сгенерированные ассемблированием двух исходных кодов, объединяются с помощью PIP, и затем полученный файл предоставляется в качестве входных данных утилите GENMOD, которая производит выходной файл с расширением PRL. Утилита GENMOD описана в [Разделе 6.2](#_GENMOD).

Приложение G содержит пример странично-перемещаемой программы. Код в примере программы показывает использование операторов ORG для получения доступа к BDOS и FCB по умолчанию. Следует отметить следующие моменты:

* Начальным ORG является адрес 0000H. Он устанавливает приравниванием символу BASE, начало перемещаемого сегмента.
* Оператор ORG 100H устанавливает фактическое начало кода программы. Во время второго ассемблирования эти два оператора ORG изменяются на 100H и 200H, соответственно.
* Ассемблирование первого создает файл, который может быть изменен в исполняемый файл COM с помощью утилиты LOAD. На самом деле, желательно сначала отладить программу как файл COM и затем сделать его файлом PRL.
* Важно использовать BASE, для смещения всех ссылок сегмента памяти базовой страницы. Программа не может выполнить вызовы BDOS по абсолютному адресу 0005H. В этом примере BASE используется, для смещения приравниваний BDOS, FCB и BUFF. Если программа должна определить верхнюю границу своего сегмента памяти, могут использоваться следующее приравнивания и последовательность кода:

MEMSIZE EQU BASE+6

LHLD MEMSIZE ; HL = Верхняя граница сегмента памяти

Следующие шаги показывают, как генерировать cтранично-перемещаемый файл для этого примера, используя ассемблер Digital Research ASM:

1. Подготовьте программу, DUMP.ASM в примере, с правильными операторами ORG, как описано выше.
2. Предполагая, что системный диск находится на диске A, а файл DUMP.ASM находится на диске B, введите следующие команды:

1A>**ASM B:DUMP** <-- Ассемблирование файла DUMP.ASM

1A>**ERA B:DUMP.HX0**

1A>**REN B:DUMP.HX0=B:DUMP.HEX**

1A>**PIP LST:=B:DUMP.PRN[T8]**

1A>**ERA B:DUMP.PRN**

1. Отредактируйте файл DUMP.ASM, добавив 100H к каждому оператору ORG. Это может быть сделано, соединив специальный заголовок с программой, который содержит два начальных оператора ORG. Файл submit для выполнения этой функции, названный ASMPRL.SUB присутствует на дистрибутивной дискете.

1A>**ASM B:DUf4P.BBZ** <-- Ассемблирование файла DUMP.ASM второй раз

1A>**PIP B:DUMP.HEX=B:DUMP.HX0,B:DUMP.HEX** <-- Объединение HEX файлов

1A>**GENMOD B:DUMP.HEX B:DUMP.PRL** <-- Создание перемещаемого файла DUMP.PRL

# Создание RSP

## RSP и резидентные системные процедуры

Резидентные системные процессы (RSP) включаются в MP/M II во время генерации системы. Утилита GENSYS ищет в каталоге все файлы с расширением RSP, отображает имена файлов и затем запрашивает пользователя о том, нужно ли включать их в файл создаваемой системы MPM.SYS.

Также MP/M II поддерживает специальный тип RSP названный *резидентной системной процедурой*. Резидентная системная процедура обеспечивает метод последовательного использования блока кода в качестве системного ресурса. Резидентная системная процедура создается RSP. RSP создает очередь с именем резидентной системной процедуры и отправляет ей единственное двухбайтовое сообщение, содержащее адрес процедуры для получения к ней последовательного доступа. Затем RSP завершает себя.

## Генерация RSP

Метод генерации RSP походит на метод генерации странично-перемещаемой программы (как описано в [Разделе 7](#_Создание_файла_PRL)) за следующими исключениями:

* Если используется редактор связей LINK-80, расширение выходного файла RSP определяется с помощью параметра [or].
* В GENMOD выходной файла обозначается RSP, а не PRL.
* ORG в коде RSP устанавливается 000H, а не 100H.

## Код RSP

Приложение F содержит образец резидентного системного процесса. Код в примере программы иллюстрирует требуемую структуру RSP, а также механизм доступа BDOS/XDOS. Этот пример должен быть тщательно изучен, и отмечены следующие моменты:

* Первые два байта RSP устанавливаются на адрес точки входа BDOS/XDOS. Адрес заполняется загрузчиком. RSP может просто получить доступ к BDOS/XDOS, загрузив HL из начала области программы и затем выполнив инструкцию PCHL.
* Дескриптор процесса RSP должен следовать сразу за первыми двумя байтами, содержащими адрес точки входа BDOS/XDOS. Важно отметить, каким образом инициализируется дескриптор процесса. Инструкции DS используются, когда выделяется только место для хранения, в то время как инструкции DB и DW используются, когда необходимо инициализировать данные в дескрипторе процесса. Примечание: поле указателя стека дескриптора процесса указывает на адрес сразу после выделения стека. Это адрес возврата и фактически является точкой входа процесса.
* Файл HEX, сгенерированный ассемблированием RSP, должен охватывать всю программу и область данных. Для обеспечения этого, используйте инструкцию DW для определения первых двух байтов RSP, которые содержат адрес точки входа BDOS/XDOS. Используя инструкцию DS ассемблер не генерирует код в HEX файл для этих двух байтов. Конец программы и области данных должны быть определены аналогичным образом. Если RSP имеет инструкции DS, предшествующие оператору END, необходимо поместить оператор DB после операторов DS и перед оператором END.

## RSP с коммутацией банков

MP/M II поддерживает форму RSP, названную RSP с поддержкой коммутации банков, которая состоит из двух частей: резидентной и переключаемой части. Резидентная часть содержит дескриптор процесса RSP и любые другие структуры данных, например, очереди, которые должны находиться в общей памяти. Переключаемая часть с расширением файла BRS обычно содержит оставшуюся часть RSP, стек и другие структуры данных. Резидентная часть RSP с поддержкой переключения банков должна следовать правилам приведенным в предыдущем разделе для RSP. Присутствие переключаемой части RSP определяется, присвоением индексу сегмента памяти дескриптора процесса значения 0, вместо FFH. Имя, указанное в дескрипторе процесса, используется для получения переключаемой части RSP, которая имеет расширение файла BRS. Должны быть отмечены следующие моменты, связанные с BRS:

* Байты 0000-0001H переключаемой части RSP зарезервированы для адреса резидентной части RSP. Таким образом переключаемый RSP должен получить доступ к функциям BDOS/XDOS, косвенно загрузив из двух байтов 0000-0001H относительный адрес, который указывает на начало резидентной части RSP, два байта которого в свою очередь содержат адрес точки входа BDOS/XDOS.
* Байты 0002-0003H переключаемой части RSP, должны содержать начальное значение указателя стека процесса. Таким образом стек переключаемой части RSP находится в переключаемой части RSP и должен быть инициализирован таким образом, что обратный адрес вершины стека являлся адресом точки входа переключаемой части RSP.
* Байты 0004-000BH переключаемой части RSP, должны содержать ASCII имя процесса. Оно используется для отображения во время выполнения GENSYS и MPMLDR.

# Создание SPR

## Системные странично-перемещаемые файлы

Системные странично-перемещаемые файлы помещаются в файл MPM.SYS во время генерации системы. Ряд файлов SPR предоставляются как часть стандартной MP/M II: резидентная и поддерживающая коммутацию банков части BDOS, названные RESBDOS.SPR и BNKBDOS.SPR, резидентная и поддерживающая коммутацию банков части XDOS, названные XDOS.SPR и BNKXDOS.SPR и TMP с поддержкой коммутации банков, названный TMP.SPR.

Другой файл SPR названный RESXIOS.SPR, или BNKXIOS.SPR содержит настроенную пользователем XIOS, которая является уникальной для оборудования, на котором выполняется MP/M II. В этом разделе приведены обзор метода генерации этого пользовательского SPR. Подробное обсуждение генерации RESXIOS.SPR или BNKXIOS.SPR приведено в *Разделе 1.3 Системного руководства MP/M II*.

## Генерация SPR

Метод генерации SPR походит на метод генерации странично-перемещаемой программы (как описано в [Разделе 7](#_Создание_файла_PRL_1)) за следующими исключениями:

* если используется LINK-80, выходной файл с расширением SPR определяется параметром [os].
* выходной файл GENMOD определяется SPR, а не PRL.
* код в SPR с помощью ORG устанавливается 000H, а не 100H.

Приложение A Назначения флагов

┌───────┐

│ 0 │ Зарезервирован

├───────┤

│ 1 │ Единица системного такта

├───────┤

│ 2 │ Односекундный интервал

├───────┤

│ 3 │ Одноминутный интервал

├───────┤

│ 4 │ Не определен

│ │

│ . │ .

│ │

│ . │ .

│ │

│ . │ .

├───────┤

│ 31 │ Не определен

└───────┘

**Рисунок A-1. Назначения флагов**

Приложение B Назначения приоритетов процесса

|  |  |
| --- | --- |
| 0 - 31 | Обработчики прерываний |
| 32 - 63 | Системные процессы |
| 64 - 197 | Не определены |
| 198 | Процесс терминальных сообщений (TMP) |
| 199 | Интерпретатор командной строки (CLI) |
| 200 | Приоритет пользователя по умолчанию |
| 201 - 254 | Приоритеты пользователя |
| 255 | Процесс ожидания (IDLE) |

Приложение C Перечень функции BDOS

Таблица С-1 Перечень функции BDOS

| **№** | **Имя функции** | **Входные параметры** | **Возвращаемые значения** |
| --- | --- | --- | --- |
| 0 | Сброс системы | Нет | Нет |
| 1 | Ввод с консоли | Нет | A = символ |
| 2 | Вывод в консоль | E = символ | Нет |
| 3 | Необработанный консольный ввод | Нет | A = символ |
| 4 | Необработанный вывод на консоль | E = символ | Нет |
| 5 | Вывод на устройство печати | E = символ | Нет |
| 6 | Прямой ввод-вывод с консоли | см. описание | см. описание |
| 7 | Получить байт ввода-вывода | Не поддерживается в MP/M II | |
| 8 | Установить байт ввода-вывода | Не поддерживается в MP/M II | |
| 9 | Печать строки | DE = адрес строки | Нет |
| 10 | Чтение консольного буфера | DE = адрес буфера | см. описание |
| 11 | Получить статус консоли | Нет | A = 00/01 |
| 12 | Возврат номера версии | Нет | HL = номер версии |
| 13 | Сброс дисковой системы | Нет | см. описание |
| 14 | Выбор диска | E = номер диска | см. описание |
| 15 | Открытие файла | DE = адрес FCB | A = код каталога |
| 16 | Закрытие файла | DE = адрес FCB | A = код каталога |
| 17 | Поиск первого соответствия | DE = адрес FCB | A = код каталога |
| 18 | Поиск следующего соответствия | Нет | A = код каталога |
| 19 | Удаление файла | DE = адрес FCB | A = код каталога |
| 20 | Последовательное чтение | DE = адрес FCB | A = код ошибки |
| 21 | Последовательная запись | DE = адрес FCB | A = код ошибки |
| 22 | Создание файла | DE = адрес FCB | A = код каталога |
| 23 | Переименование файла | DE = адрес FCB | A = код каталога |
| 24 | Получить вектор регистрации | Нет | HL = вектор регистрации |
| 25 | Получить номер текущего диска | Нет | A = номер текущего диска |
| 26 | Установить адрес DMA | DE = адрес DMA | Нет |
| 27 | Получить адрес распределения | Нет | HL = адрес распределения |
| 28 | Защита диска от записи | Нет | см. описание |
| 29 | Получить вектор только для чтения | Нет | HL = вектор R/O |
| 30 | Установка атрибутов файла | DE = адрес FCB | см. описание |
| 31 | Получить адрес блока параметров диска | Нет | HL = адрес DPB |
| 32 | Установить/Получить номер пользователя | см. описание | см. описание |
| 33 | Произвольное чтение | DE = адрес FCB | A = код ошибки |
| 34 | Произвольная запись | DE = адрес FCB | A = код ошибки |
| 35 | Вычислить размер файла | DE = адрес FCB | r0, r1, r2 |
| 36 | Установить номер произвольной записи | DE = адрес FCB | r0, r1, r2 |
| 37 | Сброс диска | DE = вектор диска | A = код ошибки |
| 38 | Доступ к диску | DE = вектор диска | A = код ошибки |
| 39 | Освобождение диска | DE = вектор диска | Нет |
| 40 | Произвольная запись с заполнением нулями | DE = адрес FCB | A = код ошибки |
| 41 | Тестирование и запись | DE = адрес FCB | HL = код ошибки |
| 42 | Блокировка записи | DE = адрес FCB  Тек. адрес DMA->File ID | HL = код ошибки |
| 43 | Разблокирование записи | DE = адрес FCB  Тек. адрес DMA->File ID | HL = код ошибки |
| 44 | Установка многосекторного счетчика | E = число секторов | A = код ошибки |
| 45 | Установка режима ошибок | E = режим ошибок BDOS | Нет |
| 46 | Получить свободное место на диске | E = номер диска | см. описание |
| 47 | Цепочка к программе | см. описание | Нет |
| 48 | Очистка буферов | Нет | см. описание |
| 100 | Установка метки каталога | DE = адрес FCB | A = код каталога |
| 101 | Получить данные метки каталога | E = номер диска | A = метка каталога |
| 102 | Чтение XFCB файла | DE = адрес FCB | A = код каталога |
| 103 | Запись XFCB файла | DE = адрес FCB | A = код каталога |
| 104 | Установить дату и время | DE = адрес TOD | Нет |
| 105 | Получить дату и время | DE = адрес TOD | Нет |
| 106 | Установка пароля по умолчанию | DE = адрес пароля | Нет |
| 107 | Получить серийный номер | Нет | DE = Поле серийного номера |

В таблице используются следующие сокращения:

символ = символ ASCII

Тек. адр. = Текущий адрес

File ID = Идентификатор файла

R/O = Только для чтения

Примечание: DE относится к регистровой паре DE, HL относится к регистровой паре HL, и что A = L, и B = H при возврате.

Приложение D Перечень функции XDOS

Таблица D-1 Перечень функции XDOS

| **№** | **Имя функции** | **Входные параметры** | **Возвращаемые значения** |
| --- | --- | --- | --- |
| 128 | Запрос абсолютной памяти | DE = адрес MD | HL = код ошибки |
| 129 | Запрос перемещаемой памяти | DE = адрес MD | HL = код ошибки |
| 130 | Освобождение памяти | DE = адрес MD | Нет |
| 131 | Опрос устройства | E = устройство | Нет |
| 132 | Ожидание флага | E = номер флага | A = код возврата |
| 133 | Установка флага | E = номер флага | A = код возврата |
| 134 | Создание очереди | DE = адрес QCB | Нет |
| 135 | Открытие очереди | DE = адрес UQCB | A = код возврата |
| 136 | Удаление очереди | DE = адрес QCB | A = код возврата |
| 137 | Чтение из очереди | DE = адрес UQCB | Нет |
| 138 | Условное чтение из очереди | DE = адрес UQCB | A = код возврата |
| 139 | Запись в очередь | DE = адрес UQCB | Нет |
| 140 | Условная запись в очередь | DE = адрес UQCB | A = код возврата |
| 141 | Задержка | DE = количество тактов | Нет |
| 142 | Диспетчеризация | Нет | Нет |
| 143 | Завершение процесса | DE = код завершения | Нет |
| 144 | Создание процесса | DE = дескриптор процесса | Нет |
| 145 | Установить приоритет | E = приоритет | Нет |
| 146 | Присоединение консоли | Нет | Нет |
| 147 | Отсоединение консоли | Нет | Нет |
| 148 | Установка консоли | E = консоль | Нет |
| 149 | Назначение консоли | DE = адрес APB | A = код возврата |
| 150 | Отправка команды CLI | DE = адрес CLICMD | Нет |
| 151 | Вызов системной резидентной процедуры | DE = адрес CPB | HL = код возврата |
| 152 | Разбор имени файла | DE = адрес PFCB | см. описание |
| 153 | Получить номер консоли | Нет | A = номер консоли |
| 154 | Адрес системных данных | Нет | HL = адрес системных данных |
| 155 | Получить дату и время | DE = адрес TOD | Нет |
| 156 | Возврат адреса дескриптора процесса | Нет | HL = адрес дескриптора процесса |
| 157 | Завершить указанный процесс | DE = адрес ABTPB | A = код возврата |
| 158 | Присоединение устройства печати | Нет | Нет |
| 159 | Отсоединение устройства печати | Нет | Нет |
| 160 | Установить устройство печати | E = номер устройства печати | Нет |
| 161 | Условное присоединение устройства печати | Нет | A = код возврата |
| 162 | Условное присоединение консоли | Нет | A = код возврата |
| 163 | Возврат номера версии MP/M | Нет | HL = номер версии |
| 164 | Получить номер устройства печати | Нет | A = номер устройства печати |

Приложение E Пример странично-перемещаемой программы

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Примечание: \*

\* Эта распечатка программы была включена \*

\* в качестве примера и может не отражать \*

\* изменения, необходимые для последних \*

\* выпусков MP/M. Поэтому читатель должен \*

\* ассемблировать и распечатать программу \*

\* представленную на дистрибутивном диске. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

page 0

0000 org 0000h

0000 base equ $

0100 org 0100h

; Примечание: base0100.asm или base0200.asm должны

; применяться в начале этого файла перед ассемблированием.

;

title 'File Dump Program'

; Программа дампа файла, читает входной файл и печатает

; в шестнадцатеричном виде

; Copyright (C) 1975, 1976, 1977, 1978, 1979, 1980, 1981

; Digital Research

; Box 579, Pacific Grove

; California, 93950

0005 = bdos equ base+5 ; Точка входа DOS

0001 = cons equ 1 ; Ввод с консоли

0002 = typef equ 2 ; Вывод на консоль

0009 = printf equ 9 ; Печать строки

000B = brkf equ 11 ; Получить статус консоли

000F = openf equ 15 ; Открытие файла

0014 = readf equ 20 ; Последовательное чтение

005C = fcb equ base+5ch ; Адрес блока управления файлом

0080 = buff equ base+80h ; Адрес входного буфера диска

; Не графические символы

000D = cr equ 0dh ; Возврат каретки

000A = lf equ 0ah ; Перевод строки

; Определения блока управления файлом

005C = fcbdn equ fcb+0 ; Имя диска

005D = fcbfn equ fcb+1 ; Имя файла

0065 = fcbft equ fcb+9 ; Расширение файла (3 символа)

0068 = fcbrl equ fcb+12 ; Номер текущего экстента

006B = fcbrc equ fcb+15 ; Счетчик записей в файле (0 до 128)

007C = fcbcr equ fcb+32 ; Номер текущей (следующей) записи

007D = fcbln equ fcb+33 ; Длина FCB

;

; Настройка стека

0100 210000 lxi h,0

0103 39 dad sp

; Поместить указатель вершины стека в HL из CCP

0104 221F02 shld oldsp

; Установка SP к локальной области стека (восстановливается в конце)

0107 316102 lxi sp,stktop

; Чтение и печать последовательных буферов

010A CDC601 call setup ; Настройка входного файла

010D FEFF cpi 255 ; 255 если выйл не существует

010F C21B01 jnz openok ; пропустить если открыт нормально

; Файл не существует, вывести сообщение об ошибке и выйти

0112 11FD01 lxi d,opnmsg

0115 CDA101 call err

0118 C35601 jmp finis ; Перети к завершению

openok: ; Операция открытия успешна, установка индекса буфера в конец

011B 3E80 mvi a,80h

011D 321D02 sta ibp ; Установка указателя буфера к 80h

; HL содержит следующий адрес печати

0120 210000 lxi h,0 ; Начать с 0000

gloop:

0123 E5 push h ; Сохранить позицию строки

0124 CDA701 call gnb

0127 E1 pop h ; Восстановить позицию строки

0128 DA5601 jc finis ; gnb устанавливает перенос если конец файла

012B 47 mov b,a

; Распечатать шестнадцатеричные значения

; Проверить линию сгиба

012C 7D mov a,l

012D E60F ani 0fh ; Проверить младшие 4 бита

012F C24401 jnz nonum

; Печатать номера строки

0132 CD7701 call crlf

; Проверить наличие клавиши перерыва

0135 CD5E01 call break

; Младший разряд (lsb) акумулятора = 1, Если символ готов

0138 0F rrc ; в перенос

0139 DA5101 jc purge ; не печатать больше

013C 7C mov a,h

013D CD9401 call phex

0140 7D mov a,l

0141 CD9401 call phex

nonum:

0144 23 inx h ; к следующему номеру строки

0145 3E20 mvi a,' '

0147 CD6A01 call pchar

014A 78 mov a,b

014B CD9401 call phex

014E C32301 jmp gloop

purge:

0151 0E01 mvi c,cons

0153 CD0500 call bdos

finis:

; Конец дампа, вернуться к CCP

; (обратите внимание, что JMP на 0000h перезагружает компьютер)

0156 CD7701 call crlf

0159 2A1F02 lhld oldsp

015C F9 sphl

; Указатель стека содержит расположение стека CCP

015D C9 ret ; к CCP

;

; Подпрограммы

;

break: ; Проверка нажатия клавиши перерыва (фактически, любая клавиша)

015E E5D5C5 push h! push d! push b ; Сохранение окружения

0161 0E0B mvi c,brkf

0163 CD0500 call bdos

0166 C1D1E1 pop b! pop d! pop h ; Восстановление окружения

0169 C9 ret

pchar: ; Печать символа из A

016A E5D5C5 push h! push d! push b ; Сохранить

016D 0E02 mvi c,typef

016F 5F mov e,a

0170 CD0500 call bdos

0173 C1D1E1 pop b! pop d! pop h ; Восстановить

0176 C9 ret

crlf:

0177 3E0D mvi a,cr

0179 CD6A01 call pchar

017C 3E0A mvi a,lf

017E CD6A01 call pchar

0181 C9 ret

pnib: ; Печать полубайта из A

0182 E60F ani 0fh ; Младшие 4 бита

0184 FE0A cpi 10

0186 D28E01 jnc p10

; меньше или равно 9

0189 C630 adi '0'

018B C39001 jmp prn

; больше или равно 10

018E C637 p10: adi 'A' - 10

0190 CD6A01 prn: call pchar

0193 C9 ret

phex: ; Печать шестнадцатеричного символа в регистре A

0194 F5 push psw

0195 0F rrc

0196 0F rrc

0197 0F rrc

0198 0F rrc

0199 CD8201 call pnib ; Печать полубайта

019C F1 pop psw

019D CD8201 call pnib

01A0 C9 ret

Err:

; Печать сообщения об ошибке

; DE адресует сообщение, заканчивающееся "$"

01A1 0E09 mvi c,printf ; Функция печати буфера

0LA3 CD0500 call bdos

0LA6 C9 ret

;

;

gnb: ; Получить следующий байт

01A7 3A1D02 lda ibp

01AA FE80 cpi 80h

01AC C2B801 jnz g0

01AF CDD301 call diskr ; Чтение следующего буфера

01B2 B7 ora a ; нулевое значение, если чтение успешно

01B3 CAB801 jz g0 ; для другого байта

; Конец данных, возврат сустановленным переносом для EOF

01B6 37 stc

01B7 C9 ret

;

g0: ; Считайть байт в buff+reg A

01B8 5F mov e,a ; ls byte of buffer index

01B9 1600 mvi d,0 ; удвоенный индекс в DE

01BB 3C inr a ; index= index+1

01BC 321D02 sta ibp ; назад в память

; Увеличить указатель на 1

; сохранить текущий адрес файла

01BF 218000 lxi h,buff

01C2 19 dad d

; Абсолютный адрес символа находится в HL

01C3 7E mov a,m

; Байт находится в аккумуляторе

01C4 B7 ora a ; Сброс бита переноса

01C5 C9 ret

;

setup:

; Настройка файла

; откройтие файла для ввода

01C6 AF xra a ; Обнуление аккумулятора

01C7 327C00 sta fcbcr ; Очистить текущую запись

01CA 115C00 lxi d,fcb

01CD 0E0F mvi c,openf

01CF CD0500 call bdos

; Если ошибка открытия в аккумуляторе 255

01D2 C9 ret

diskr: ; Чтение записи из файла на диске

01D3 E5D5C5 push hl push d! push b

01D6 115C00 lxi d,fcb

01D9 0E14 mvi c,readf

01DB CD0500 call bdos

01DE C1D1E1 pop b! pop d! pop h

01E1 C9 ret

;

; Область фиксированных сообщенй

signon:

01E2 46696C6520 db 'File dump MP/M version 1.0$’

opnmsg:

01FD 0D0A4E6F20 db cr,lf,'No input file present on disk$'

;

; Область переменных

021D ibp: ds 2 ; Указатель входного буфера

021F oldsp: ds 2 ; Значение SP из CCP

;

; Область стека

0221 ds 64 ; Зарезервировать 32 уровня стека

stktop:

0261 end

Приложение F Пример резидентного системного процесса

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Примечание: \*

\* Эта распечатка программы была включена \*

\* в качестве примера и может не отражать \*

\* изменения, необходимые для последних \*

\* выпусков MP/M. Поэтому читатель должен \*

\* ассемблировать и распечатать программу \*

\* представленную на дистрибутивном диске. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

page 0

title 'Type file on console'

; Программа печати файла, читает входной файл и

; распечатывает его на консоли

; Copyright (C) 1979, 1980

; Digital Research

; P.O. Box 579

; Pacific Grove, CA 93950

0000 org 0000h ; Начало стандартного RSP

001A = ctlz equ 1ah ; Сtrl-Z используется для EOF

0002 = conout equ 2 ; Номер функции BDOS вывод на консоль

0009 = printf equ 9 ; Номер функции BDOS печать буфера

0014 = readf equ 20 ; Чтение следующей записи

000F = openf equ 15 ; Открытие FCB

0098 = parsefn equ 152 ; Разбор имени файла

0086 = mkque equ 134 ; Создание очереди

0089 = rdque equ 137 ; Чтение из очереди

0091 = stprior equ 145 ; Установка приоритета

0093 = detach equ 147 ; Отсоединение консоли

;

; Адрес точки входа BDOS

bdosadr:

0000 0000 dw $-$ ; Загрузчик заполнит это

;

; Определение дескриптора процесса

;

typepd:

0002 0000 dw 0 ; Указатель

0004 00 db 0 ; Состояние процесса

0005 0A db 10 ; Приоритет (начальный)

0006 1001 dw stack+38 ; Указатель стека

0008 5459504520 db 'TYPE ' ; Имя в верхнем регистре

pdconsole:

0010 ds 1 ; Консоль

0011 FF db 0ffh ; memseg

0012 ds 2 ; b

0014 ds 2 ; Список потоков процессов (thread)

0016 3601 dw buff ; Адрес установки DMA диска

0018 ds 1 ; Код пользователя & выбранный диск

0019 ds 2 ; dcnt

001B ds 1 ; searchl

001C ds 2 ; searcha

00LE ds 2 ; Активный диск

0020 ds 20 ; Область хранения регистров

0034 ds 2 ; scratch

;

; Определение блока управления связной очереди

;

typelqcb:

0036 0000 dw 0 ; Указатель

0038 5459504520 db 'TYPE ' ; Имя в верхнем регистре

0040 4800 dw 72 ; msglen - длина сообщения

0042 0100 dw 1 ; nmbmsgs

0044 ds 2 ; dqph

0046 ds 2 ; nqph

0048 ds 2 ; mh

004A ds 2 ; mt

004C ds 2 ; bh

004E ds 74 ; buf – буфер (72 + 2 байта указателя)

;

; Определение блока управления очередью пользователя (UQCB)

;

typeuserqcb:

0098 3600 dw typelqcb ; pointer - фактический адрес очереди

009A 9C00 dw field ; msgadr - адрес буфера пользователя

;

; Поле для сообщения читаемого из определения связанной QCB

;

field:

009C ds 1 ; Выбранный диск

console:

009D ds 1 ; Консоль

filename:

009E ds 72 ; Тело сообщения

;

; Parse file name control block

;

pcb:

00E6 9E00 dw filename ; Адрес имени файла

00E8 1201 dw fcb ; Адрес блока управления файлом

;

; Область стека и других локальныз структур данных

;

stack:

00EA ds 38 ; 20 уровней стека

0110 ba0: dw type ; Точка входа процесса

0112 fcb: ds 36 ; Блок управления файлом

0136 buff: ds 128 ; Буфер файла

;

; Процедура вызовов BDOS

;

bdos:

01B6 2A0000 lhld bdosadr ; HL = Адрес BDOS

01B9 E9 pchl

;

; Определение основной программы

;

type:

01BA 0E86 mvi c,mkque

01BC 113600 lxi d,typelqcb

01BF CDB601 call bdos ; Make typelqcb

01C2 0E91 mvi c,stprior

01C4 11C800 lxi d,200

01C7 CDB601 call bdos ; Установить приоритет равным 200

forever:

01CA 0E89 mvi c,rdque

01CC 119800 lxi d,typeuserqcb

01CF CDB601 call bdos ; Read from type queue

01D2 0698 mvi c,parsefn

01D4 11E600 lxi d,pcb

01D7 CDB601 call bdos ; Разбор имени файла

01DA 23 inx h

01DB 7C mov a,h

01DC B5 ora 1 ; Проверка на 0ffffh

01DD CA1F02 jz error

01E0 3A9D00 lda console

01E3 321000 sta pdconsole ; typepd.console = console

01E6 0E0F mvi c,openf

01E8 111201 lxi d,fcb

01EB CDB601 call bdos ; Открыть файл

01EE 3C inr a ; Проверить код возврата

01EF CA1F02 jz error ; Если он 0ffh, нет файла

01F2 AF xra a ; иначе,

01F3 323201 sta fcb+32 ; Установить следующую запись равной нулю

new$sector:

01F6 0E14 mvi c,readf

01F8 111201 lxi d,fcb

01FB CDB601 call bdos ; Чтение следующей записи

01FE B7 ora a

01FF C22702 jnz done ; Выход если EOF или ошибка

0202 213601 lxi h,buff ; Указатель на сектор данных

0205 0E80 mvi c,128 ; Получить счетчик байтов

next$byte:

0207 7E mov a,m ; Получить байт

0208 5F mov e,a ; Сохранить в E

0209 FE1A cpi ctlz

020B CA2702 jz done ; Выход если EOF

020E C5 push b ; Сохранить счетчик байтов

020F E5 push h ; Сохранить адрес регистра

0210 0E02 mvi c,conout

0212 CDB601 call bdos ; Вывод на консоль

0215 E1 pop h ; Восстановить указатель

0216 C1 pop b ; и счетчик

0217 23 inx h ; Продвинуть указатель

0218 0D dcr c ; уменьшить байт счетчика

0219 C207O2 jnz next$byte ; если больше 0 - в этом секторе

021C C3F601 jmp new$sector ; иначе, нужен новый

error:

021F 112F02 lxi d,err$msg ; Указатель на сообщение об ошибке

0222 0E09 mvi c,printf ; Получить код функции для печати

0224 CDB601 call bdos

done:

0227 0E93 mvi c,detach

0229 CDB601 call bdos ; Отсоеденить консоль

022C C3CA01 jmp forever

err$msg:

022F 0D0A46696C db 0dh,0ah,'File Not Found or Bad File Name$'

0251 end

Приложение G Сокращения и обозначения

В данном руководстве используются следующие условные обозначения при описании физических модулей MP/M II, его функциональных частей и структур данных:

**1 Физические модули**

BDOS - Базовая дисковая операционная система

XDOS - Расширенная дисковая операционная система

XIOS - Расширенная система ввода-вывода

**2 Функциональные части**

CLI - Интерпретатор командной строки

TMP - Процессор сообщений терминала

**3 Структуры данных**

FCB - Блок управления файлом

MD - Дескриптор памяти

PD - Дескриптор процесса

QCB - Блок управления очередью

UQCB - Блок управления очередью пользователя

XFCB - Расширенный блок управления файлом

**4 Соглашения о присвоении имен**

Именование файлов:

PRL - Странично-перемещаемый файл

SPR - Системная перемещаемая страница

RSP - Резидентный системный процесс (или процедура)

BRS - RSP с поддержкой коммутации банков

Системная точка входа - первая буква(ы) - прописные буквы и номер функции находятся в круглых скобках

Приложение H Глоссарий

**Блок**: Основная единица распределения дискового пространства в MP/M II. Каждый диск имеет фиксированный размер блока, определенный в своем блоке параметров диска в XIOS. Размер блока может быть 1K, 2K, 4K, 8K или 16K последовательных байт. Блоки нумеруются относительно нуля так, что каждый блок в файл уникален и имеет байтовое смещение равное Номер блока\*Размер блока.

**Блок параметров диска** (DPB): Таблица, находящаяся в XIOS, который определяет характеристики диска в дисковой подсистеме, используемой в MP/M II. Адрес DPB находится в заголовке параметров диска (DPH) в DPbase + 0AH. Диски с одинаковыми характеристиками могут использовать тот же заголовок параметров диска, и таким образом тот же DPB. Однако диски с разными характеристиками должны иметь свои собственные заголовки параметров диска и DPB. Адрес заголовка параметров диска возвращается в регистровой паре HL, когда BDOS вызывает точку входа SELDSK в BIOS. Функция BDOS 31 возвращает адрес DPB.

**Блок управления файлом**: Тридцать шесть последовательных байтов определяемых пользователем для функций файлового ввода-вывода. Поля FCB описаны в [Разделе 2.2.3](#_Определение_блока_управления_1). Термин FCB также относится к элементу каталога в части каталога распределения дискового пространства. Они содержат те же первые 32 байта FCB, описанные в Разделе 2.1 не хватает только байтов номера текущей записи и номера произвольной записи.

**Буфер по умолчанию**: 128-байтовый буфер, поддерживается начиная с 0080H в нулевой странице. Когда CLI загружает COM файл, CLI инициализирует этот буфер хвостом команды, т.е. любыми символами, введенными после имени COM файла. Первый байт в 0080H содержит длину хвоста команды, в то время как сам хвост команды начинается в 0081H. Двоичный ноль завершает хвост команды. Команда "I" в DDT инициализирует этот буфер таким же образом, как это делает CLI.

**Ввод-вывод**: Операции или подпрограммы, которые обрабатывают ввод и вывод данных в компьютерной системе.

**Вектор**: Ячейки памяти. Точки входа в операционную систему, используемые для системных вызовов или обработки прерываний.

**Вектор контрольной суммы**: Непрерывная область данных в XIOS с одним байтом для проверки каждого сектора каталога. Вектор контрольной суммы инициализируется и поддерживается для каждого зарегистрированного диска. Каждый доступ к каталогам приводит к вычислению системой контрольной суммы, которая сравнивается с контрольной суммой в векторе. Если есть несоответствие, то диск устанавливается в состояние "только для чтения". Это предотвращает пользователя от непреднамеренной смены дисков, без регистрации нового диска. Если новый диск не зарегистрирован, он рассматривается как, старый и данные на нем могут быть уничтожены при записи на него.

**Двоично-десятичное число** (BCD): Представление десятичных чисел, используя двоичные цифры. См. *Приложение I* для двоичных представлений кодов ASCII.

**Дорожка**: Концентрическое кольцо на диске. Стандартные дискеты одинарной плотности IBM имеют 77 дорожек. Каждая дорожка состоит из фиксированного числа пронумерованных секторов. Дорожки нумеруются от 0 до на единицу меньше, чем число дорожек на диске. К данным на дисковом носителе получают доступ с помощью комбинации номеров дорожек и номеров секторов.

**Заголовок параметров диска**: 16-байтовая область в XIOS, который содержит информацию о дисководе и области временной памяти для определенных операций BDOS. Учитывая n дисков, заголовки параметров дисков расположены в таблице с первой строкой из 16 байт, соответствующей диску 0 и последней строкой, соответствующей диску n-1.

**Запись**: Самая маленькая единица информации в дисковом файле, которая может быть считана или записана. Запись состоит из 128 последовательных байтов, смещение байтов которых в файле представляет собой номер записи умноженный на 128. 128-байтовая запись в файле занимает один 128-байтовый сектор на диске. Если используются алгоритм блокирования и разблокирование, несколько записей могут занимать каждый сектор диска.

**Запись каталога**: Запись файла выводимая на экран при использовании команды DIR. Этот термин также относится к физическому элементу каталога (FCB).

**Каталог**: Часть диска, содержащая записи для каждого файла на диске и расположение блоков, выделенных файлам. Каждый элемент каталога файлов находится в виде 32-разрядного FCB, хотя один файл может иметь несколько элементов в зависимости от его размера. Максимальное количество поддерживаемых элементов каталога указывается в блока параметров диска дисковода.

**Команда**: Набор инструкций, которые выполняются, когда название команды введено после системного приглашения. Эти инструкции могут быть "встроены" в систему MP/M II или находиться на диске в виде файла с расширением COM или PRL. В целом команды MP/M II состоят из трех частей: название команды, хвоста команды и возврата каретки

**Логическая переменная**: Переменная, у которой может только быть два значения. Обычно интерпретируется как Истина/Ложь, или Вкл./Выкл.

**Логический диск**: Логически выделенная область физического диска. Физический диск может быть разделен на один или более логических дисков, и обозначаться конкретными ссылками диска (например, d:a или d:f, и т.д.). Таким образом в пользовательском интерфейсе, кажется, что в системе существует несколько дисков.

**Нулевая страница**: Область памяти между 0000H и 0100H, которая содержит критические системные параметры и функции прежде всего как интерфейсная область между пользовательскими программами и модулем BDOS.

**Подстановочный символ**: Символы ? или \*. Функция BDOS поиска в каталоге ищет соответствие ? с любым единственным символом и \* с несколькими символами.

**Пользователь**: Логически обособленный подраздел каталога. Каждый каталог можно разделить на 16 областей пользователя.

**Разбор**: Разделение командной строки на ее составные части.

**Разделители**: Символы ASCII, которые разделяют составные части спецификации файла. CLI распознает определенные символы разделители как: ":", ".", "=", ";", "<", ">", пробел и возврат каретки. Несколько команд MP/M II также рассматривают символы "[", "]", "(", ")", "$" в качестве разделителей. Желательно избегать использования символов разделителей и символов в нижнем регистре в именах файлов.

**Реентерабельный код**: Код, который может использовать один процесс, в то время как другой уже выполняет его. Данные для повторно используемого кода обычно сохраняются в стеке.

**Сектор**: Основная единица данных, прочитанных и записанных XIOS на диск. Сектор может быть одной 128-байтовой записью в файле или секторе каталога. В некоторых дисковых подсистемах размер сектора диска больше, чем 128 байтов, обычно степень двойки - 256, 512, 1024 или 2048 байтов. Эти секторы диска упоминаются как укрупненные сектора. Когда размер укрупненного сектора больше, чем 128 байтов, укрупненные сектора должны быть буферизованы в памяти, и 128-байтовые сектора должны быть блокированы и разблокированы из них.

**Спулинг**: Печать файла с диска. Программа SPOOL, которая отсоединенной от консоли, может распечатать файл с диска. Она оставляет вашу консоль свободной для других задач, во время печати вашего файла.

**Стек**: Зарезервированная область памяти, где процессор сохраняет адрес возврата, когда он получает инструкцию CALL. Когда процессор встречается с инструкцией возврата, он восстанавливает текущий адрес из стека в счетчике команд. Такие данные, как содержимое регистров могут быть сохранены в стеке. Инструкция PUSH помещает данные в стек и инструкция POP удаляет их. Стек 8080 - 16 битный. Инструкции, воздействующие на стек, добавлять и удалять элементы стека одно слово за раз. Элемент продвинутый в стек, уменьшает указатель стека (SP) на два и записывает элемент по адресу SP. Другими словами стек растет в памяти вниз.

**Только для чтения**: Состояние, в котором диск может быть считан, но на него не возможна запись. Диск может быть установлен в состояние только для чтения при помощи утилит SET или STAT или функции *Set File Attributes* (Функция 30B DOS). Диск может также быть установлен в состояние только для чтения, если контрольная сумма, вычисленная при доступе к каталогу, не соответствует, сохраненной в CSV, для зарегистрированного диска. Это защищает пользователя от смены дисков, без выполнения сброса диска.

**Файл**: Набор данных, содержащий от 0 до 242144 записей. Каждая запись содержит 128 байт и может содержать или двоичный файл или данные ASCII. Файлы данных ASCII состоят из строк с данными, разделенных последовательностями символов перевода строки и возврата каретки, т.е. одна 128-байтовая запись может содержать одну или более строк текста. Файлы состоят из одного или более экстентов с 128 записями в экстенте. Каждый файл имеет один или более элементов каталога, но показывает только один элемент каталога при использовании команды DIR.

**Физический диск**: Периферийное аппаратное устройство, используемое в качестве запоминающее устройство большой емкости для хранения данных внутри компьютера.

**Формат файла HEX**: Абсолютный вывод ASM и MAC для Intel 8080. Файл HEX содержит последовательность абсолютных записей, которые задают адрес загрузки и значения байтов, которые будут сохранены, начиная с адреса загрузки. (см. [Раздел 4](#_Ассемблер_8080_ASM)).

**Экстент**: 16K последовательных байтов в файле. Экстенты нумеруются от 0 до 31. Один экстент может содержать 1, 2, 4, 8 или 16 блоков. EX - поле номера экстента в FCB и является однобайтовым полем в FCB + 12, где FCB метка первого байта в FCB. В зависимости от размера блока и максимального номера блока данных, FCB может содержать 1, 2, 4, 8 или 16 экстентов. Обычно, поле EX устанавливается пользователем равным 0, но содержит текущее номер экстента во время файлового ввода-вывода. Термин “Сворачивание FCB” описывает FCB, содержащий больше чем один экстент. В CP/M версии 1.4 каждый FCB содержит только один экстент. Пользователи, пытающиеся выполнить произвольный ввод-вывод записей и поддержать совместимость с CP/M 1.4, должны знать об этих различиях.

**Элемент каталога**: 32-байтовый элемент связанный с каждым файлом на диске. Файл может иметь более одного, связанного с ним, элемента каталога. В секторе каталога располагаются четыре элемента каталога. Элементы каталога могут также рассматриваться как каталог FCB.

**BCD**: См. Двоично-десятичное число (binary coded decimal).

**COM**: Расширение файла для командных файлов MP/M II. Это объектные модули машинного языка, готовые для загрузки и выполнения. Любой файл с этим расширением может быть выполнен, просто введя имя файла после подсказки диска (например, 0A>). Например, программа PIP.COM может быть выполнена, просто введя PIP.

**CSV**: См. Вектор контрольной суммы.

**DPB**: См. Блок параметров диска.

**DPH**: См. Заголовок параметров диска.

**EX**: См. Экстент.

**FCB**: См. Блок управления файлом.

**FCB по умолчанию**: Один из двух FCB, поддерживаемых в 005CH и 006CH соответственно, в нулевой странице. CLI инициализирует первый FCB по умолчанию из первого поля оделенного разделителем от хвоста команды и инициализирует второй FCB по умолчанию из следующего поля в хвосте команды.

**R/O**: См. Только для чтения.

Приложение I ASCII и шестнадцатеричные преобразования

Это приложение содержит таблицу символов ASCII, включая их двоичные, десятичных и шестнадцатеричных преобразования.

Таблица ‑ Символы ASCII

|  |  |  |  |
| --- | --- | --- | --- |
| **Символ** | **Значение** | **Символ** | **Значение** |
| ACK | Подтверждение | FS | Разделитель файлов |
| BEL | Звонок | GS | Разделитель групп |
| BS | Возврат на один символ | HT | Горизонтальная табуляция |
| CAN | Отмена | LF | Перевод строки |
| CR | Возврат каретки | NAK | Отсутствие подтверждения |
| DC | Контроль устройства | NUL | Нет данных |
| DEL | Удаление | RS | Разделитель записей |
| DLE | Переключение | SI | Режим обычного ASCII |
| EM | Конец носителя | SO | Режим национальных символов |
| ENQ | Запрос | SOH | Начало заголовка |
| EOT | Конец передачи | SP | Пробел |
| ESC | Управляющая последовательность | STX | Начало текста |
| ETB | Конец блока передачи | SUB | Подставить |
| ETX | Конец текста | SYN | Синхронизация |
| FF | Перевод страницы | US | Разделитель элементов |
|  |  | VT | Вертикальная табуляция |

Таблица ‑ Таблица преобразования ASCII

| **Binary** | **Decimal** | **Hexadecimal** | **ASCII** | **Binary** | **Decimal** | **Hexadecimal** | **ASCII** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0000000 | 0 | 00 | NUL | 1000000 | 64 | 40 | @ |
| 0000001 | 1 | 01 | SOH (CTRL-A) | 1000001 | 65 | 41 | A |
| 0000010 | 2 | 02 | STX (CTRL-B) | 1000010 | 66 | 42 | B |
| 0000011 | 3 | 03 | ETX (CTRL-C) | 1000011 | 67 | 43 | C |
| 0000100 | 4 | 04 | EOT (CTRL-D) | 1000100 | 68 | 44 | D |
| 0000101 | 5 | 05 | ENQ (CTRL-E) | 1000101 | 69 | 45 | E |
| 0000110 | 6 | 06 | ACK (CTRL-F) | 1000110 | 70 | 46 | F |
| 0000111 | 7 | 07 | BEL (CTRL-G) | 1000111 | 71 | 47 | G |
| 0001000 | 8 | 08 | BS (CTRL-H) | 1001000 | 72 | 48 | H |
| 0001001 | 9 | 09 | HT (CTRL-I) | 1001001 | 73 | 49 | I |
| 0001010 | 10 | 0A | LF (CTRL-J) | 1001010 | 74 | 4A | J |
| 0001011 | 11 | 0B | VT (CTRL-K) | 1001011 | 75 | 4B | K |
| 0001100 | 12 | 0C | FF (CTRL-L) | 1001100 | 76 | 4C | L |
| 0001101 | 13 | 0D | CR (CTRL-M) | 1001101 | 77 | 4D | M |
| 0001110 | 14 | 0E | SO (CTRL-N) | 1001110 | 78 | 4E | N |
| 0001111 | 15 | 0F | SI (CTRL-0) | 1001111 | 79 | 4F | O |
| 0010000 | 16 | 10 | DLE (CTRL-P) | 1010000 | 80 | 50 | P |
| 0010001 | 17 | 11 | DCI (CTRL-Q) | 1010001 | 81 | 51 | Q |
| 0010010 | 18 | 12 | DC2 (CTRL-R) | 1010010 | 82 | 52 | R |
| 0010011 | 19 | 13 | DC3 (CTRL-S) | 1010011 | 83 | 53 | S |
| 0010100 | 20 | 14 | DC4 (CTRL-T) | 1010100 | 84 | 54 | T |
| 0010101 | 21 | 15 | NAK (CTRL-U) | 1010101 | 85 | 55 | U |
| 0010110 | 22 | 16 | SYN (CTRL-V) | 1010110 | 86 | 56 | V |
| 0010111 | 23 | 17 | ETB (CTRL-W) | 1010111 | 87 | 57 | W |
| 0011000 | 24 | 18 | CAN (CTRL-X) | 1011000 | 88 | 58 | X |
| 0011001 | 25 | 19 | EM (CTRL-Y) | 1011001 | 89 | 59 | Y |
| 0011010 | 26 | 1A | SUB (CTRL-Z) | 1011010 | 90 | 5A | Z |
| 0011011 | 27 | 1B | ESC (CTRL-[) | 1011011 | 91 | 5B | [ |
| 0011100 | 28 | 1C | FS (CTRL-\) | 1011100 | 92 | 5C | \ |
| 0011101 | 29 | 1D | GS (CTRL-]) | 1011101 | 93 | 5D | ] |
| 0011110 | 30 | 1E | RS (CTRL-^) | 1011110 | 94 | 5E | ^ |
| 0011111 | 31 | 1F | US (CTRL-\_) | 1011111 | 95 | 5F | \_ |
| 0100000 | 32 | 20 | (Пробел) | 1100000 | 96 | 60 | ` |
| 0100001 | 33 | 21 | ! | 1100001 | 97 | 61 | a |
| 0100010 | 34 | 22 | " | 1100010 | 98 | 62 | b |
| 0100011 | 35 | 23 | # | 1100011 | 99 | 63 | c |
| 0100100 | 36 | 24 | $ | 1100100 | 100 | 64 | d |
| 0100101 | 37 | 25 | % | 1100101 | 101 | 65 | e |
| 0100110 | 38 | 26 | & | 1100110 | 102 | 66 | f |
| 0100111 | 39 | 27 | ' | 1100111 | 103 | 67 | g |
| 0101000 | 40 | 28 | ( | 1101000 | 104 | 68 | h |
| 0101001 | 41 | 29 | ) | 1101001 | 105 | 69 | i |
| 0101010 | 42 | 2A | \* | 1101010 | 106 | 6A | j |
| 0101011 | 43 | 2B | + | 1101011 | 107 | 6B | k |
| 0101100 | 44 | 2C | , | 1101100 | 108 | 6C | l |
| 0101101 | 45 | 2D | - | 1101101 | 109 | 6D | m |
| 0101110 | 46 | 2E | . | 1101110 | 110 | 6E | n |
| 0101111 | 47 | 2F | / | 1101111 | 111 | 6F | o |
| 0110000 | 48 | 30 | 0 | 1110000 | 112 | 70 | p |
| 0110001 | 49 | 31 | 1 | 1110001 | 113 | 71 | q |
| 0110010 | 50 | 32 | 2 | 1110010 | 114 | 72 | r |
| 0110011 | 51 | 33 | 3 | 1110011 | 115 | 73 | s |
| 0110100 | 52 | 34 | 4 | 1110100 | 116 | 74 | t |
| 0110101 | 53 | 35 | 5 | 1110101 | 117 | 75 | u |
| 0110110 | 54 | 36 | 6 | 1110110 | 118 | 76 | v |
| 0110111 | 55 | 37 | 7 | 1110111 | 119 | 77 | w |
| 0111000 | 56 | 38 | 8 | 1111000 | 120 | 78 | x |
| 0111001 | 57 | 39 | 9 | 1111001 | 121 | 79 | y |
| 0111010 | 58 | 3A | : | 1111010 | 122 | 7A | z |
| 0111011 | 59 | 3B | ; | 1111011 | 123 | 7B | { |
| 0111100 | 60 | 3C | < | 1111100 | 124 | 7C | | |
| 0111101 | 61 | 3D | = | 1111101 | 125 | 7D | } |
| 0111110 | 62 | 3E | > | 1111110 | 126 | 7E | ~ |
| 0111111 | 63 | 3F | ? | 1111111 | 127 | 7F | Rubout |

Алфавитный указатель

А

Адрес дескриптора процесса, 25

Адрес системных данных, 116

Архивный атрибут, 33

Атрибут системный, 33

Атрибут только для чтения, 33

Атрибуты интерфейса, 33, 39, 60, 61

Атрибуты файла, 33

Б

Базовая дисковая операционная система, 26

Битовая карта, 147

Битовый вектор, 73

Биты атрибутов, 32

Блок параметров диска, 44, 74

Блок управления очередью, 97

Блок управления очередью пользователя, 97

Блок управления файлом, 30

Блокирование и разблокирование, 43

Блокировка записи, 83

Быстродействие, 9

В

Ввод с консоли, 54

Вектор распределения, 72

Виртуальный размер файла, 78

Внутрисистемные дата и время, 92

Возврат адреса дескриптора процесса, 117

Возврат номера версии, 59

Возврат номера версии MP/M, 119

Время суток, 14

Выбор диска, 59

Вывод в консоль, 55

Вывод на устройство печати, 56

Вызов системной резидентной процедуры, 113

Вычислительный процесс, 8, 9, 10, 110

Вычислить размер файла, 78

Г

Генерация системы, 8, 22, 40

Д

Дата и время, 92

Дескриптор памяти, 101

Дескриптор процесса, 9, 25, 94, 149

Директивы ассемблера, 126

Диск по умолчанию, 18

Диспетчеризация, 9, 10, 57, 109

Длина FCB, 30

Доступ к диску, 44, 45, 80

Доступ к файлу, 41

З

Завершение процесса, 110

Завершить указанный процесс, 117

Загрузка программы, 21

Задержка, 109

Задержка выполнения, 14, 109

Закрытие файла, 32, 37, 62

Запись в очередь, 108

Запись XFCB файла, 91

Запрос абсолютной памяти, 104

Запрос перемещаемой памяти, 105

Защита диска от записи, 46, 72

Защита файлов, 40

И

Идентификатор файла, 31, 38, 42, 61, 69, 83

Индекс сегмента памяти, 101, 104, 105

Инициализация базовой страницы, 21

Инициализация FCB, 32

Интерпретатор командной строки, 8, 19, 86, 88

Исходные файлы, 30

К

Ключевое поле, 79

Код выбора диска, 28

Код каталога, 48, 49, 50

Код ошибки, 48, 50

Консоль, 16, 17, 25

Контрольная сумма FCB, 40

Контрольные точки, 139

Л

Логический диск, 26, 29

М

Макросы, 121

Метка даты и времени

доступа, 61

обновления, 62

создания, 69

Метка каталога, 27, 35, 36, 38, 63, 89, 90

Многосекторный ввод-вывод, 43

Многосекторный счетчик, 27, 43, 86

Н

Назначить консоль, 112

Неблокируемый режим, 38, 41

Недогрузка флага, 12

Необработанный вывод на консоль, 55

Необработанный консольный ввод, 55

Неоднозначная файловая ссылка, 34, 63

Номер пользователя, 34

Номер произвольной записи, 30

О

Области базовой страницы, 51

Область данных, 26

Область каталога, 26

Область каталога диска, 29

Обработка ошибок, 104

Ожидание флага, 12, 14, 106

Оператор языка ассемблера, 121

Оператор ORG, 127

Операторы, 125

Операция регистрации, 45

Операция сброса диска, 44

Опрос, 12, 105

Освобождение диска, 44, 46, 81

Освобождение памяти, 105

Открытие очереди, 107

Открытие файла, 32, 38, 60

Отправка команды CLI, 112

Отсоединить консоль, 111

Отсоединить устройство печати, 118

Очередь взаимного исключения, 11, 100

Очистка буферов, 44, 88

П

Память с коммутацией банков, 8, 13, 15

Параметры ассемблера, 120

Переименование файла, 70

Переполнение флага, 12, 106

Печать строки, 57

Планирование загрузки программ, 14

Поиск первого, 63

Поиск следующего, 64

Поле пароля, 20, 28, 115

Поле расширения файла, 20

Полубайт, 50

Получить адрес блока параметров диска, 74

Получить адрес распределения, 72

Получить вектор регистрации, 71

Получить вектор только для чтения, 73

Получить данные метки каталога, 90

Получить дату и время, 92, 116

Получить номер консоли, 116

Получить номер текущего диска, 71

Получить номер устройства печати, 119

Получить свободное место на диске, 87

Получить серийный номер, 93

Получить статус консоли, 58

Пользователь 0, 34

Пользовательские каталоги, 34

Поля базовой страницы, 53

Последовательная запись, 67

Последовательное чтение, 65

Постоянный диск, 43, 44, 46

Преобразование буквенных символов, 121

Приоритет процесса, 10, 95

Присоединение консоли, 25, 55, 111

Присоединение устройства печати, 25, 117

Проверка контрольной суммы, 40

Программы CP/M, 3, 7, 21

Произвольная запись, 76

Произвольная запись с заполнением нулями, 81

Произвольное чтение, 75

Процесс ожидания, 10

Процесс терминальных сообщений, 8, 16

Процессор 8080

регистры, 144

флаги, 143

Прочие функции, 27

Прямой ввод-вывод с консоли, 56

Прямой доступ к памяти, 71

Р

Разблокирование записи, 42, 85

Разбор имени файла, 28, 113

Разделители, 28

Размер блока данных, 29

Размер файла, 29, 78

Разреженные файлы, 30

Расширение файла, 42

Расширенные ошибки, 46, 95

Режим блокировки, 38

Режим группового обмена, 43

Режим записи, 37

Режим открытия файла, 38

Режим ошибок, 27, 46

Режим по умолчанию, 87

Режим совместного доступа, 41

Режим удаления, 37

Режим чтения, 37

Резидентная системная процедура, 23, 149

Резидентный системный процесс, 7

С

Сброс диска, 44, 80

Сброс дисковой системы, 44, 59

Сброс системы, 54

Связанные с диском функции, 27

Связи между процессами, 8

Связная очередь, 99

Сегмент внутренних данных, 103

Сегмент памяти, 20, 21, 95, 104, 105

Символы командной строки, 53

Система логических прерываний, 12

Системная консоль, 7, 18

Системная страница данных, 101

Системное время, 8, 14

Системные странично-перемещаемые файлы, 151

Системный диск, 20

Системный процесс MP/M II, 7

Системный стек, 95

Соглашение об именовании

очередей, 100

процессов, 97

файлов, 29

Создание очереди, 106

Создание процесса, 110

Создание файла, 68

Сообщение об ошибке, 46

Состояние процесса, 9

Спецификация диска, 20

Спецификация файла, 114

Список блокировки, 40, 42, 45, 80

Список готовых процессов, 10, 12, 13, 15, 19, 110

Список исключенных из очереди, 11

Список опроса, 13

Список поставленных в очередь, 11, 19

Стек системных вызовов пользователя, 16

Структура памяти, 15

Съемный диск, 44

Т

Текущий номер пользователя, 19, 20

Тестирование и запись, 42, 81

У

Удаление очереди, 107

Удаление файла, 64

Управление консолью, 13

Управление устройством печати, 13

*Управляющие символы*, 25

Управляющие символы редактирования, 58

Условная запись в очередь, 109

Условное присоединение консоли, 119

Условное присоединение устройства печати, 118

Условное чтение из очереди, 108

Установить адрес DMA, 71

Установить дату и время, 92

Установить номер произвольной записи, 79

Установить приоритет, 111

Установить устройство печати, 118

Установить/Получить номер пользователя, 74

Установка атрибутов файла, 33, 73

Установка консоли, 111

Установка метки каталога, 31, 89

Установка многосекторного счетчика, 43, 50, 86

Установка пароля по умолчанию, 93

Установка режима ошибок, 46, 87

Установка флага, 12, 106

Устройство печати, 25

Утилита

DUMP, 146

GENHEX, 145

GENMOD, 145

GENSYS, 149

LOAD, 146

PRLCOM, 145

TOD, 38

Ф

Файловая система BDOS, 26, 27, 29, 32, 40

Файловые ссылки, 27

Файлы SPR, 151

Физическая ошибка

выбора, 47

диск только для чтения, 47

плохой сектор, 47

файл только для чтения, 47

Физические ошибки, 46

Физический размер файла, 78

Флаг ошибки, 49

Формат командной строки, 19, 53

Формат файла PRL, 147

Формат FCB, 34

Формат Intel Hex, 120, 127

Функции доступа к файлу, 26

Функции каталога, 27

Ц

Цепочка к программе, 88

Цикл задержки, 9, 12, 14

Циклическая очередь, 97, 98

Ч

Частичный сброс, 45

Чтение из очереди, 108

Чтение консольного буфера, 57

Чтение XFCB файла, 90

Э

Элемент открытого файла, 45, 61

Элементы каталога файлов, 32

Я

Ядро, 9

A

ASM, 120, 147

B

BASE, 147

BRS, 150

F

FCB по умолчанию, 52

L

Link-80, 147

M

MPMLDR, 23, 44, 150

R

RSP с поддержкой коммутации банков, 17, 22, 150

S

SUBMIT, 18

X

XFCB, 35, 91

XIOS, 8, 43

**Операционная система MP/M**

**Версия 2.1**

Примечания к выпуску

Авторское право © 1982

Digital Research

P.O. Box 579

801 Lighthouse Avenue

Pacific Grove, CA 93950

(408) 649-3896

TWX 910 360 5001

Все права защищены

Уважаемый пользователь MP/M II:

Digital Research разработала операционную систему MP/M II в ответ на многочисленные запросы клиентов добавить возможность совместного доступа к файлам в MP/M версии 1.1. Дизайн MP/M II является отражением нашей цели предоставить Вам современную операционную систему, которая может быть сконфигурирована для большого разнообразия компьютерного оборудования.

Эта партия содержит выпуск версии 2.1 нашей операционной системы MP/M II. Мы были довольны реакцией на MP/M II версии 2.0 и надеемся видеть сопоставимую реакцию на MP/M II версии 2.1 относительно дизайна, возможных расширений и ошибок в реализации. Мы надеемся сохранить тот же уровень доверия у компьютерной отрасли которую имела наша операционная система CP/M 0.

На основе нашего опыта и опыта пользователей MP/M II, мы оцениваем, что требуется менее недели для реализации простой MP/M II, основанной на опросе, на компьютере, у которого есть рабочая версия CP/M версии 2.2. При реализации оптимизированной системы MP/M II с полной поддержкой прерываний и коммутации банков памяти может потребовать несколько недель. Конечно, время для выполнения такой реконфигурации будет существенно различаться в зависимости от опыта программиста и сложности оборудования.

Примечание: Удостоверьтесь, что вы используете команду SET или STAT для преобразования файла USER.PRL в системный файл.

Свяжитесь с сотрудниками технической поддержки Digital Research (408) 375 6262, если вы испытываете трудности при перенастройке MP/M II. Отправляя запросы в Вашу учетную карточку Вы можете обеспечить, чтобы мы отправили по почте указания по применению MP/M II и патчи исправляющие ошибки реализации.

С уважением

Техническая поддержка

Операционная система MP/M II™

Версия 2.1

Расширенная блокировка файла

Дополнение к Руководству программиста операционной системы MP/M II

Авторское право © 1982 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Скомпилировано в январе 1982 года

Расширенная блокировка файла - новое средство, реализованное в MP/M II™ версии 2.1. Расширенная блокировка файла позволяет процессу поддержать блокировку файла даже после закрытия файла. Это средство позволяет процессу переименовывать, устанавливать атрибуты или удалять файл, без необходимости бороться с возможностью вмешательства других процессов после закрытия файла. Кроме того, процесс может вновь открыть файл с расширенной блокировкой и продолжить нормальную обработку файла. Например, процесс может открыть файл, выполнить файловые операции с файлом, закрыть файл, переименовать файл, вновь открыть файл под его новым именем и продолжить операции с файлом, не теряя элемент списка блокировки этого файла и контроль над файлом.

Расширенная блокировка файла доступна только файлам, которые открыты в режиме открытия по умолчанию (заблокированном режиме). Чтобы расширить блокировку файла, установите атрибут интерфейса F6' при закрытии файла. Этот атрибут опрашивается только функцией *Close*, когда она закрывает файл навсегда. Таким образом атрибут интерфейса F5' должен быть сброшен, когда выполняется вызов *Close*. Кроме того, если файл был открыт N раз (несколько раз), этот атрибут опрашивается только, когда файл закрывается в N-ый раз.

Для поддержки расширенной блокировки файла с помощью вызова *Rename File* или вызова *Set File Attributes*, установите атрибут интерфейса F5' в FCB, указанном при совершении вызова. Этот атрибут учитывается только для расширенных блокировок файла, не нормальных блокировок. Установка атрибута F5' также поддерживает расширенную блокировку файла для функции *Delete File*, но установка этого атрибута также, изменяет поведение функции *Delete* приводя к удалению только XFCB. В случае успеха, все эти три функции удаляют элемент расширенной блокировки файла, если вызваны с сброшенным атрибутом F5'. С другой стороны, если они возвращаются с кодом ошибки, элемент расширенный блокировки не удаляется.

Для возобновления файловых операций с файлом с расширенной блокировкой может быть сделан стандартный вызов *Open*. Однако, режим открытия, ограничен блокированным режимом по умолчанию. Следующий список иллюстрирует использование расширенных блокировок.

* Откройте файл EXLOCK.TST в заблокированном режиме.
* Выполните файловые операции с файлом EXLOCK.TST используя открытый FCB.
* Закройте файл EXLOCK.TST с установленным атрибутом интерфейса F6', чтобы сохранить элемент блокировки файла.
* Используйте функцию *Rename File*, для изменения имени файла к EXLOCK.NEW с установленным атрибутом интерфейса F5', чтобы сохранить элемент расширенной блокировки файла.
* Откройте файл EXLOCK.NEW в заблокированном режиме.
* Выполните файловые операции с файлом EXLOCK.NEW используя открытый FCB.
* Закройте файл EXLOCK.NEW с установленным атрибутом интерфейса F6', чтобы сохранить элемент блокировки файла.
* Установите атрибут только для чтения и освободите элемент блокировки файла при помощи функции *Set File Attributes* с сброшенным интерфейсным атрибутом интерфейса F5'. В этот момент файл EXLOCK.NEW становится доступным для доступа другим процессам.

Операционная система MP/M II™ Версия 2.1

Атрибуты совместимости

Дополнение к Руководству программиста операционной системы MP/M II

Авторское право © 1982 принадлежит Digital Research

CP/M является зарегистрированной торговой маркой Digital Research

MP/M и MP/M II является торговой маркой Digital Research

Скомпилировано в феврале 1982 года

Файловая система MP/M II™ вводит некоторые новые ограничения, касающиеся файловых операций, которые не присутствовали в MP/M™ 1.1 или CP/M©. Например, если процесс открывает файл в режиме по умолчанию (заблокирован), MP/M II не позволяет другим процессам в системе, открыть, удалить или переименовать файл, пока процесс открывший файл не закрывает файл или не завершиться. Кроме того, MP/M II не позволяет процессу выполнение файловых операций с FCB, который не был активирован успешной операцией открытия или создания, или с FCB, который был деактивирован операцией закрытия. Эти ограничения защищают пользователей MP/M II от вмешательства со стороны других пользователей к своим открытым файлам. В качестве примера, эта защита позволяет пользователю в MP/M II редактировать файл с уверенностью, что другой пользователь не может удалять или изменять его файл во время его сеанса редактирования.

Новые ограничения, добавленные в MP/M II, обеспечивают защиту файлов, когда несколько пользователей работают в системе. Предыдущий пример описывает ограничения, необходимые для предупреждения коллизий файловых операций между независимыми процессами. Другое новое ограничение MP/M II устанавливают границы каким образом процесс может изменять, открытый FCB. Эти границы осуществляются проверкой контрольной суммы открытых FCB и защищают целостность файловой системы MP/M II от поврежденных FCB. Обратите внимание, что новые ограничения MP/M II не предназначены для защиты пользователя от его собственных действий. Вместо этого они гарантируют, что действия одного пользователя не оказывают негативного влияния на других пользователей в системе.

Как правило, новые ограничения файловой системы MP/M II не создают больших трудностей при разработке новых приложений. Фактически, они заставляют соблюдать хороший стиль программирования. Однако из-за этих новых ограничений, некоторые программы CP/M и MP/M, написанные до выпуска MP/M II не работает в MP/M II. Кроме того, несколько копий некоторых программ не работают, потому что режим открытия по умолчанию в MP/M II является заблокированным режимом, в котором только один процесс может открыть файл.

Для решения этих проблем Digital Research добавил в MP/M II версии 2.1 атрибуты совместимости. Атрибуты совместимости определены как атрибуты F1' до F4' программных файлов. Новый параметр GENSYS определяет, должны ли атрибуты быть активированы. Если они активированы, интерпретатор командной строки (CLI) опрашивает эти атрибуты во время загрузки программы и изменяет основные правила MP/M II для загруженной программы, как описано ниже.

Обратите внимание, что атрибуты совместимости не должны использоваться с новым программным обеспечением. Они предназначены для использования программ, разработанных для CP/M и MP/M 1.1. Особенно это относится к атрибуту совместимости F4', который отключает проверку контрольной суммы FCB в операциях чтения и записи. Этот атрибут следует использовать с осторожностью и только с программами, которые точно работают.

**Определения атрибутов совместимости**

**F1'** *Открытие по умолчанию в MP/M 1.1*. У процессов, работающих с этим атрибутом все файлы, открытые в заблокированном режиме, помечаются в системном списке блокировки как только для чтения. Это позволяет всем процессам с установленным этим атрибутом читать и писать в общие файлы без ограничений. Однако, не предусмотрена блокировка записей. Кроме того, этот атрибут также позволяет процессу писать в файл, открытый другим процессом в режиме только для чтения. Чтобы быть в безопасности, все статические файлы, такие как программы и файлы справки, должны быть сделаны только для чтения, когда используется этот атрибут совместимости.

**F2'** *Частичное закрытие по умолчанию*. У процессов, работающих с этим атрибутом, есть свой режим закрытия по умолчанию, измененный с постоянного закрытия на частичное закрытие. Этот атрибут предназначен для программ, которые закрывают файл, чтобы обновить каталог, но продолжают использовать файл. Обратите внимание, что MP/M II предполагает, что процесс завершился с освобождением файла, когда число закрытий, выданных к файлу, равняется числу открытий. Побочный эффект этого атрибута заключается в том, что файлы, открытые процессом, не освобождаются до завершения процесса. Возможно, необходимо присвоить параметрам системного списка блокировок высокие значения при использовании этого атрибута.

**F3'** *Игнорировать ошибки контрольных сумм при закрытии*. Этот атрибут изменяет способ обработки процессом ошибок контрольной суммы при закрытии. Обычно, в консоль выводится сообщение и процесс завершается. Если установлен этот атрибут, и во время операции закрытия обнаружена ошибка контрольной суммы, файл закрывается, если элемент списка блокировки существует для этого файла. В противном случае, процессу вызова возвращается код ошибки неуспешного закрытия.

**F4'** *Отключение проверки контрольной суммы FCB для операций чтения и записи*. Установка этого атрибута также устанавливает атрибуты F2' и F3'. Этот атрибут следует использовать осторожно, потому, что он эффективно отключает безопасность файлов MP/M II. Используйте этот атрибут только с программами, которые точно работают.

**Процедура использования атрибутов совместимости**

1. Ответьте yes на вопрос GENSYS, "Enable Compatibility Attributes (N) ?" (Включить атрибуты совместимости (N)?).
2. Используйте утилиту SET MP/M II для установки желаемой комбинации атрибутов совместимости в имени программы.

Примеры:

0A>SET filespec [Fl=on]

0A>SET filespec [Fl=on,F3=on]

0A>SET filespec [F4=on]

Если у вас есть программа, которая работает в CP/M или MP/M 1.1, но не работает должным образом в MP/M II, используйте следующие рекомендации по выбору атрибутов совместимости для программы.

1. Если программа заканчивается сообщением, "File Currently Opened" (Файл в настоящее время открыт), когда выполняется несколько копий программы, установите атрибут совместимости F1'. В качестве альтернативы, вы могли бы рассмотреть вопрос о размещении всех общих статических файлов в области пользователя 0 с установленными атрибутами SYS и R/O.
2. Если программа завершается с сообщением, "Close Checksum Error" (ошибка контрольной суммы закрытия), установите атрибут совместимости F3'.
3. Если программа завершается с ошибкой ввода-вывода, попытайтесь выполнить программу с установленным атрибутом F2'. Если проблема не исчезнет, попробуйте атрибуты F2' и F3'. Если проблема все еще сохраняется, то попробуйте атрибут F4'. Используйте атрибут F4' только в крайнем случае.

Возможно при использовании атрибутов совместимости F2' и F4' необходимо увеличить параметры GENSYS, которые определяют максимальное количество файлов, которые может открыть процесс, и размер системного списка блокировок. Это может потребоваться, потому что оба атрибута по умолчанию используют частичное закрытие. В результате записи системного списка блокировок, используемые открытыми файлами, не освобождаются, пока процесс не завершается. Как правило, если процесс завершается сообщением "No Room in System Lock List" (нет места в системном списке блокировок) или "Open File Limit Exceeded" (превышен лимит открытых файлов), это обычно означает, что вышеупомянутое параметры GENSYS должны быть увеличены. Другим вариантом является перенос вызова BDOS *Free Drive* в программе в точку, где файлы не активны. Обратите внимание, что вызов *Free Drive* с определением всех дисков, производит очистку записей блокировки во всех файловых системах, принадлежащих процессу вызова.

Когда GENSYS активирует атрибуты совместимости, интерпретатор командной строки копирует параметры настройки атрибутов F1' до F4' имени файла загруженной программы в байт 1DH дескриптора процесса как показано ниже:

**Байт 1DH дескриптора процесса**

(Биты определены 7-0 старший бит к младшему биту)

Бит 7 установлен = F1

Бит 6 установлен = F2

Бит 5 установлен = F3

Бит 4 установлен = F4

Операционная система MP/M II™ Версия 2.1

Рекомендации по программированию

Дополнение к Руководству программиста операционной системы MP/M II

Авторское право © 1982 принадлежит Digital Research

CP/M является зарегистрированной торговой маркой Digital Research

MP/M и MP/M II является торговой маркой Digital Research

Скомпилировано в январе 1982 года

Настоящее пособие содержит дополнительное обсуждение информации, представленной в Руководстве программиста операционной системы MP/M II™. В частности, этот документ подчеркивает те области MP/M II, где существуют ограничения, которых не было в CP/M© версий 1 и 2 и MP/M™ версии 1 Намерение состоит в том, чтобы прикладные программисты MP/M II, избежали потенциальные проблем с новым программным обеспечением. В качестве предварительного условия, читатель должен быть знаком с материалом, представленным в *Руководстве программиста операционной системы MP/M II*.

1. Всегда используйте следующую последовательность при выполнении файловых операций , которые требуют открытого файла. В MP/M II этими операциями являются команды BDOS - чтение , запись, блокирование и разблокирование записей.

Активируйте FCB файла с помощью вызовов функций BDOS открытия или создания перед использованием FCB в файловых операциях. Убедитесь, что операция открытия или создания была успешной. MP/M II принимает только FCB, активированные путем успешного вызова открытия или создания для операций открытия файла. Если используется FCB, которое не было активировано, MP/M II возвращает ошибку контрольной суммы.

Выполняйте все операции с файлами, используя активированные FCB. Обратите внимание, что MP/M II не отключает активированный FCB при возвращении файловыми операциями кодов ошибок. Как правило, только поля текущей записи и произвольной записи активированного FCB должны быть изменены. Кроме того, все файловые операции с активированным FCB должны быть сделаны под номером пользователя, который был в силе, когда FCB был активирован. Аналогичное ограничение применяется к активированным FCB, определяющим диск по умолчанию. Все файловые операции, определяющие такую FCB должны быть сделаны в соответствии с текущим диск, который был в силе при активации FCB. Пункт 3 в этом списке включает полный перечень правил, касающиеся модификации активированных FCB.

Если процесс завершил файловые операций с файлом, но еще должен выполнить обработку информации, файл должен быть закрыт. Это применяется, даже если файл не был изменен. За некоторыми исключениями, запись в списке блокировки, связанная с файлом в списке блокировки системы не освобождается до тех пор, пока файл не закрыт окончательно (см. Раздел 2.2.9 Руководства программиста операционной системы MP/M II). MP/M II ограничивает доступ к файлу другим процессам, пока элемент в списке блокировки файла находится в системном списке блокировки. Нет необходимости закрывать входные файлы, если процесс собирается закончиться. При завершении, все элементы блокировки принадлежащие процессу освобождаются. Выходные файлы, однако, всегда должна быть закрыты или данные могут быть потеряны. Обратите внимание, что успешная операция окончательного закрытия деактивирует FCB и удаляет файл из списка блокировки системы. Если деактивированный FCB используется в последующей операции открытого файла, MP/M II возвращает ошибку контрольной суммы.

1. Если процесс открывает один и тот же файл несколько раз, соответствующее число команд закрытия должно быть выполнено с файлом для удаления элементов списка блокировки файлов из системного списка блокировки. Таким образом, если файл был открыт N раз, то первые N-1 операций закрытия файла, выполняются для частичного закрытия операций. Только последнее закрытие, операция закрытия N, интерпретируется как окончательное закрытие. По определению окончательным закрытием является операция закрытия, которая удаляет элемент указанного файла из системного списка блокировки. Обратите внимание, что для файла выделяется только один элемент списка блокировки в системном списке блокировки независимо от числа FCB, открытых процессом для файла.
2. Следующий список определяет, как активированный FCB может быть изменены не затрагивая контрольную сумму FCB. MP/M II возвращает код ошибки контрольной суммы и не выполнить запрошенную операцию, если FCB с неверной контрольной суммой используется в операции открытого файла.

* FCB(0) не может указывать новый диск.
* За исключением атрибутов интерфейса F5' и F6' для функции BDOS *Close*, FCB(1) до FCB(11) не могут быть изменены.
* Старшие 3 бита FCB(12) не могут быть изменены. Младшие 5 битов могут быть изменены. Обратите внимание, что когда файл открыт в режиме открытия по умолчанию (режим блокировки), 3-й бит этого поля FCB устанавливается равным нулю.
* FCB(13) не может быть изменен.
* FCB(14) и FCB(15) не могут быть изменены.
* FCB(16) до FCB(31) не могут быть изменены.
* FCB(32) до FCB(35) могут быть изменены.

Если требуется совместимость с будущими версиями MP/M и CP/M, программы должны ограничивать изменение полей FCB от 32 до 35 открытых FCB. В частности Digital Research не поддерживает методы, которые включают в себя изменение полей 12, 14 и 15 открытых FCB.

1. Процессы, которые получают доступ к принтеру, должны выдать вызов *Detach List* (отсоединение устройства печати) для освобождения принтера, прежде чем другой процесс сможет использовать принтер. Если вызов *Detach List* не выполнен, процесс, который получает доступ к принтеру, продолжает владеть принтером до завершения.
2. Программы CP/M, которые создают submit файлы для объединения в цепочку должны быть изменены, чтобы работать в MP/M II. MP/M II требует другое имя для submit файлов, которое включает в себя номер исходной консоли и требует установить флаг submit в странице системных данных. Метод создания и выполнения submit файлов описан в *Указаниях по применению 07 MP/M II*. Обратите внимание, что MP/M II имеет команду *Program Chain* (функция 47), которая обеспечивает эффективный механизм для объединения программ в цепочку.
3. Программы CP/M, которые выполняют прямые вызовы BIOS дискового ввода-вывода, не работают в MP/M II. MP/M II поддерживает прямые вызовы XIOS для консоли и принтера, но не для диска. Если программы должны выполнить прямые дисковые вызовы XIOS, используется метод, настоятельно не рекомендуемый в многопользовательской среде, двух уровневой косвенной адресации для получения реального адреса таблицы переходов XIOS. Второй уровень косвенной адресации требуется, потому что таблица прерываний обрабатывает консоль и принтер.

Следующие два шага должны быть выполнены в программе, прежде чем сделать прямые вызовы XIOS к диску. Первый шаг должен выполнить вызов BDOS *Write Protect Disk* (Функция 28) для диска, чтобы гарантировать, что ни у какого другого процесса нет открытых файлов на диске. Во-вторых, должно быть считано сообщение очереди взаимного исключения MXDisk, чтобы препятствовать тому, чтобы другие программы выполняли дисковые вызовы функций BDOS, пока ваша программа выполняет прямые вызовы XIOS. После завершения ваших прямых вызовов XIOS запишите сообщение MXDisk обратно и затем сбросьте диски, которые вы установили только для чтения.

1. Следующая процедура - это протокол, который могут использовать несколько процессов для координации обновления записи и операций дополнения в совместно используемый файл. Каждый процесс должен открыть совместно используемый файл в неблокируемом режиме. Также эта процедура предполагает, что записи, содержащие двоичные нули являются нулевыми записями.

* Попытайтесь заблокировать запись.
* Если попытка блокировки не удалась, потому что другой процесс заблокировал запись, выполните задержку и повторите процедуру.
* Если попытка блокировки не удалась, потому что запись не существует в файле, добавьте запись инициализированную двоичным нулям к файлу с помощью команды BDOS *Write Random with Zero Fill*, и повторите процедуру. Обратите внимание, что файлы, открытые в неблокируемом режиме, расширяются блоками а не записями, как файлы, открытые в режиме блокировки по умолчанию.
* Если попытка блокировки выполнена успешно, считайте запись, обновите ее, и затем разблокируйте.

1. Метод ввода-вывода с несколькими FCB включает в себя открытие каждого экстента файла самостоятельно и обслуживание их в таблице в памяти. После этого произвольный ввод-вывод осуществляется, путем выбора правильного FCB из таблицы, установкой текущего поля записи в поле в надлежащий номер записи в экстенте и выполнение команды последовательного чтения или записи. После завершения обработки, каждый FCB закрывается. Максимальный размер файла, к которому можно получить доступ с помощью этого метода, составляет 512K байт. Это ограничивает максимальный размер таблицы 32 FCB. Обратите внимание, что этот метод обеспечивает выполнение операций произвольного ввода-вывода способом совместимым с CP/M 1.4.

Ввод-вывод с несколькими FCB в MP/M II должен быть выполнен аккуратно из-за ограничений MP/M II накладываемых на файловые операции для обеспечения защиты файлов. Обычно FCB не должен использоваться в файловом вводе-выводе, если он не был активирован, и он не должен изменяться, пока активирован (см. пункты 1 и 3). Кроме того, выполненное число открытий и закрытий файла имеет важное значение (см. пункт 2). Обратите внимание, что все 32 байта FCB каждого экстента должны сохраняться в таблице открытых FCB. Кроме того, проверьте, что атрибут интерфейса F8', установлен в 1 во всех FCB, если у первого FCB F8' установлен в 1. Установка F8' в 1 указывает, что файл был открыт пользователем 0 несмотря на то, что текущий код пользователя ненулевой (см. Функцию 15 в *Руководстве программиста операционной системы MP/M II*).

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 01, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M и MP/M II является торговой маркой Digital Research

Подавление отображения загрузчика MP/M™

Применимые продукты и номера версий: MP/M II версии 2.0

Программа: MPMLDR.COM

Когда загрузчик MP/M II читает файл MPM.SYS, он выводит на экран карту распределения памяти в консоль #0. В некоторых приложениях может потребоваться подавить этот вывод.

Чтобы подавить отображение карты загрузки на консоль #0, введите следующие инструкции RET в файле LDRBIOS.ASM с помощью любого стандартного редактора. Инструкция RET заменяет код вывода консоли.

;Loader BIOS jump vector:

...

jmp conout

...

conout:

ret

Ассемблируйте LDRBIOS.ASM для создания LDRBIOS.HEX. Интегрируйте новый файл LDRBIOS.HEX в файл MPMLDR.COM согласно инструкциям, приведенным в *Руководстве по операционной системе MP/M II*. Затем обновите системные дорожки загрузочного диска с новым загрузчиком.

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 02, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Установка/сброс режима необработанного ввода-вывода консоли

Применимые продукты и номера версий: MP/M II версии 2.0

Некоторые прикладные программы требуют необработанный ввод в консоли. Необработанный ввод подразумевает, что операционная система не принимает мер на специальные символы, такие как Ctrl-C.

Выполните следующий код, чтобы поместить прикладную программу в режим необработанного консольного ввода.

MVI C,9CH

CALL XDOS ; Получить адрес дескриптора процесса

LXI D,6

DAD D

MOV A,M

ORI 80H ; Включить старший бит первого

MOV M,A ; символа в имени процесса

...

Выполните следующий код, чтобы выйти из режима необработанного консольного ввода.

...

MVI C,9CH

CALL XDOS ; Получить адрес дескриптора процесса

LXI D,6

DAD D

MOV A,M

ANI 7FH ; Выключить старший бит первого

MOV M,A ; символа в имени процесса

...

Функции 3, 4, и 6 помещают систему в необработанный режим консольный ввода. Все другие функции ввода-вывода с консоли сбрасывают систему к нормальному режиму консольного ввода.

Режим необработанного консольного ввода может вызвать проблемы. Вы не можете прервать процесс, работающий в режиме без предварительной обработки, потому что система игнорирует все управляющие символы. Чтобы прервать процесс, используйте Функцию 11 перед использованием любых дисковых функций ввода-вывода. Функция 11 возвращает систему к нормальному консольному режиму.

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 03, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Изменение минимального размера буфера PRL файла

Применимые продукты и номера версий: MP/M II версии 2.0

Вы можете выделить больший буфер по умолчанию для программы, например редактора. Вы можете изменить требования к размеру минимального буфера для файлов PRL. Следующая процедура демонстрирует, как изменить требования к размеру минимального буфера для ED от 4k до 8k байтов.

0A>**ddt ed.prl**

[MP/M] DDT VERS 1.1

NEXT PC

2300 0100

-**s104**

0104 00 00

0105 10 20

0106 .

-**v2300**

0044

-ied.prl

-**w44**

-g0

Байты 4 и 5 из записи заголовка PRL (относительно начала) содержат младший и старший байты для минимальной спецификации размера буфера.

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 04, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M и MP/M II является торговой маркой Digital Research

Доступ к внутренним TOD MP/M™

Применимые продукты и номера версий: MP/M II версии 2.0

Некоторым прикладным программам может потребоваться доступ к внутренним полям времени и даты MP/M II для установки начальных значений. Выполните следующую последовательность кода в конце вашей системной процедуры инициализации XIOS MP/M™. Поместите код в конце, потому что вызов XDOS, получения адреса системной страницы данных разрешает прерывания.

...

MVI C,9AH

CALL XDOS ; Получить адрес системной страницы данных

; \*\*\* Предупреждение \*\*\*

; вызов XDOS включает прерывания

LXI D,00FCH

DAD D ; hl -> указатель -> TOD

MOV E,M

INX H

MOV D,M ; de -> TOD

...

Подпрограмма на ассемблере TODCNV.ASM, представленная на дистрибутивном диске с версией MP/M II, преобразует время и дату представленные в строке ASCII во внутренний формат времени и даты МП/М II.

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 05, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Дисковые контроллеры с DMA в системах с коммутацией банков

Применимые продукты и номера версий: MP/M II версии 2.0

Будьте особенно осторожны с системами с коммутацией банков памяти, у которых есть дисковые контроллеры прямого доступа к памяти. Во время передачи данных от дискового контроллера в банк назначения переключение банков не допускается.

DMA от дискового контроллера осуществляется через общую память, затем копируется из общей памяти в требуемый пользователю буфер. Сектора, размер которых превышает 128 байт, помещаются в буфер общей памяти. Указанный сектор затем пересылается в буфер назначения. Это разумный метод в системах, где требуется деблокирование.

Используйте следующую процедуру, если DMA должен осуществляться непосредственно в пользовательский буфер минуя общую память. Присвойте флагу активности DMA значение “Истина” перед каждой операцией DMA. Сбросьте флаг после каждой операции.

...

MVI A,FFH

STA DMACTVE

; Инициировать операцию DMA

; Выполнить ожидание флага или опрос

; для завершения операции

XRA A

STA DMACTVE

...

Поместите следующую последовательность кода в процедуру XIOS выбора памяти, чтобы исключить переключение банка во время операции DMA:

SELMEMORY:

LDA DMACTVE

ORA A

JZ OKTOSWITCH ; Перейти если не в операции DMA

; Затем, банк, который будет переключен, может быть

; сравнен с текущим банком. Если они совпадают, то

; операция DMA не будет выполнена.

JZ OKTOSWITCH ; Изменение банка не требуется

; Новый банк определен, и осуществляется операция DMA.

; Теперь должно быть выполнен Цикл ожидания освобождения,

; чтобы дождаться завершения операции DMA.

; \*\*\* Предупреждение \*\*\*

; Вызов выбора памяти выполняется из диспетчера поэтому,

; прерывания отключены, и ничего не должно быть сделано,

; что может вызвать диспетчеризацию.

BUSYWAIT:

IN DMASTATUSPORT ; Это "цикл ожидания освобождения"

ANI DMADONE

JZ BUSYWAIT ; выполняется до завершения DMA

Поместите следующий код в остающейся части процедуры выбора памяти.

OKTOSWITCH:

...

...

RET

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 06, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Использование функции Send CLI Command

Применимые продукты и номера версий: MP/M II версии 2.0

Используя функцию XDOS *Send CLI Command* можно эффективно реализовать прикладную программу управляемую с помощью меню. Следующие шаги в общих чертах описывают использование функцию XDOS *Send CLI*.

1. Измените приоритет процесса вызова так, чтобы он был выше (фактически нижнее значение), чем TMP.
2. Получите номер консоли вызывающего процесса.
3. Присвойте консоль интерпретатору командной строки.
4. Выполните вызов функции XDOS *Send CLI Command*
5. Выполните функцию присоединения консоли, чтобы вернуть консоль после того, как инициированный процесс завершится.
6. Восстановите приоритет процесса вызова к его исходному значению (обычно 200).

Сегменты меню управляемой программы с названием MENU приведены в следующем примере.

; Таблица определений функций XDOS

setpriority equ 145

attachconsole equ 146

assignconsole equ 149

sendCLIcommand equ 150

getconsole equ 153

MENU:

mvi e,190

mvi c,setpriority

call BDOS ; Установить приоритет 190

mvi c,getconsole

call BDOS ; Получить номер консоли в A

sta AssignPB ; заполнить

sta CLIcommand+1 ; поля консоли

lxi d,AssignPB

mvi c,assignconsole

call BDOS ; Назначить консоль CLI

inr a

jz cannotassign ; Назначение не удалось

lxi d,CLIcommand

mvi c,sendCLIcommand

call BDOS ; send CLI command

mvi c,attachconsole

call BDOS ; Присоеденить консоль

mvi e,200

mvi c,setpriority

call BDOS ; Установить приоритет обратно 200

...

AssignPB:

db $-$ ; Номер консоли

db 'cli ' ; Имя (CLI в нижнем регистре)

db 0

...

CLIcommand:

db 0 ; Код диска/пользователя

db $-$ ; Номер консоли

; Это строка ASCII, завершенная с нулем,

; которая является точно такой, которую Вы

; бы ввели для запуска в консоли, например.

db 'PIP LST:=MYPROG.LST[PT8]',0

...

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 07, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Создание submit файла из прикладной программы

Применимые продукты и номера версий: MP/M II версии 2.0

Следующая процедура показывает Вам, как создать submit файл из прикладной программы и вызвать ее выполнение. Процедура для завершения задания submit файла включена.

1. Получите диск временного файла из страницы системных данных.
2. Получите номер консоли, в которой выполняется программа.
3. Создайте файл $n$.SUB. Используйте n для указания номера консоли.
4. Установите соответствующий submit флаг в массиве в состояние 'on'. Массив находится в странице системных данных.

; Таблица определений функций BDOS/XDOS

;

closefile equ 16

searchfirst equ 17

deletefile equ 19

makefile equ 22

getconsole equ 153

getsysdatadr equ 154

subflgofst equ 128

...

mvi c,getsysdatadr

call BDOS

lxi d,196 ; Смещение диска временного файла

dad d

mov a,m

sta FCB

mvi c,getconsole

call BDOS

sta console

adi '0'

sta FCB+2 ; Поместить номер консоли в fname

lxi d,FCB

mvi c,searchfirst

call BDOS ; Убедиться что файл существует

inr a

jz nofile

lxi d,FCB

mvi c,deletefile

call BDOS ; Сначала удалить старую версию

nofile: lxi d,FCB

mvi c,makefile

call BDOS ; Создать файл $n$.SUB

...

; Теперь, запишите записи в $n$.SUB файл следующим образом:

; - одна строка submit файла на запись

; - Последняя запись сначала (т.е. в обратном порядке как

; они должны быть выполнены

; - каждая запись в следующей форме:

;

; [CNT][Командная строка ASCII][NULL]

; где: CNT = число символов в командной строке, 1 байт

; Командная строка ASCII <= 125 символов

; NULL = 0, 1 байт

...

lxi d,FCB

mvi c,closefile

call BDOS ;close the $n$.SUB; file

mvi c,getsysdatadr

call BDOS ;get system data page adr

lda console ;retrieve the saved con #

adi subflgofst ;add offset to base of

mov e,a ;submit flag array

mvi d,0

dad d ;DE = submitflag(console);

mvi m,0ffh ;set 'on' submit flag

...

; Завершение программы

...

FCB:

DB 1 ; Дисковод, обычно A:

DB '$n$ ' ; Имя файла

DB 'SUB' ; Расширение файла

DB 0 ; Экстент файла

DS 20 ; Остаток FCB

console:

ds 1 ; Временный локальный номер консоли

...

Завершите работу задания submit, обнулив флаг submit, расположенный в области памяти страницы системных данных. Чтобы найти и обнулить флаг submit для консоли используйте следующий код процедуры.

;

; Таблица определений функций XDOS

;

getconsole equ 153

getsysdatadr equ 154

subflgofst equ 128

...

mvi c,getconsole

call BDOS ; Получить номер консоли

push psw ; Сохранить номер консоли

mvi c,getsysdatadr

call BDOS ; Получить адрес станицы системных данных

pop psw ; Восстановить номер консоли

adi subflgofst

mov l,a ; hl = адрес флага submit

mvi m,0 ; Обнулить флаг submit

...

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 08, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M и MP/M II является торговыми марками Digital Research

CP/M является торговой маркой Digital Research

Wordstar является торговой маркой MicroPro International Corporation

Совместный доступ к файлам

Применимые продукты и номера версий: MP/M II**™** версии 2.0

Несколько пользователей могут совместно использовать файлы, используя файловую систему MP/M II. Прикладная программа, такая как Wordstar® требует, чтобы файлы были открыты во время работы программы. Несколько пользователей программы должны совместно использовать открытые файлы. Обычно, в MP/M II, совместное использование файлов вызывает проблемы, если прикладная программа не структурирована для открытия файла в режиме только для чтения. Режим по умолчанию для функции открытия - блокированный режим, который предотвращает совместное использование файлов. Файлы открываются в блокированном режиме для более ранних версий CP/M® 0 и также MP/M™.

Чтобы включить совместный доступ к файлам, поместите все файлы, которые будут совместно использованы в область пользователя 0 на диске по умолчанию. Используя утилиту SET, присвойте атрибуты SYS (системный) и RO (только для чтения) файлам. BDOS открывает файл в режиме только для чтения, независимо от указанного режима функции открытия. Пример показан ниже.

0A>set wsmsgs.com [SYS,RO]

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 09, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Программное управление переключателем Ctrl-P

Применимые продукты и номера версий: MP/M II™ версии 2.0

Прикладной программе может потребоваться эхо консоли ввода-вывода на принтер с помощью программного управления. Используйте следующие процедуры, чтобы установить и очистить флаги Ctrl-P. Массив флагов расположен по адресу SYSTEM DATA PAGE + 126.

Установка флага Ctrl-P

mvi c,9ah ; Получить адрес станицы системных данных

call BDOS

lxi d,126

dad d ; Добавить 126 к адресу станицы системных данных

mov e,m

inx h

mov d,m ; DE = адрес массива Ctrl-P

push d

mvi c,0a4h ; Получить номер устройства печати

call BDOS

mov e,a

mvi d,0

pop h

dad d

mvi m,0ffh ; Установить флаг Ctrl-P

; Эхо консольного ввода-вывода включено

Очистка флага Ctrl-P

mvi c,9ah ; Получить адрес станицы системных данных

call BDOS

lxi d,126

dad d ; Добавить 126 к адресу станицы системных данных

mov e,m

inx h

mov d,m ; DE = адрес массива Ctrl-P

push d

mvi c,0a4h ; Получить номер устройства печати

call BDOS

mov e,a

mvi d,0

pop h

dad d

mvi m,0 ; Сбросить флаг Ctrl-P

; Эхо консольного ввода-вывода снова выключено

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 10, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Автозагрузка при холодном старте

Применимые продукты и номера версий: MP/M II™ версии 2.0

MP/M II может выполнить одну команду во время "холодного" старта. Однако система может выполнить любое число команд во время "холодного" старта, если начальная команда - SUBMIT.

Для выполнения команды автозагрузки поместите команду автозагрузки в единственном числе в файл, используя стандартный формат команды. Назовите этот файл $n$.SUP где n номер консоли, которая выполняет команду. Файл $n$.SUP находится на системном диске в требуемой области пользователя или области пользователя 0 с атрибутом SYS (системный). Примеры приведены ниже.

Файл автозагрузки: $0$.SUP

Команда в файле автозагрузки: SUBMIT START$0$

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 11, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Усовершенствования SUBMIT

Применимые продукты и номера версий: MP/M II™ версии 2.0

Усовершенствования SUBMIT включают следующие новые функции и возможности.

**Увеличенный размер файла** $n$.SUB: размер файла SUBMIT теперь неограничен. файл $n$.SUB первоначально был ограничен одним экстентом, 128 строк.

**Изменение номера пользователя**: Чтобы изменить текущий номер пользователя в SUBMIT, SUBMIT файл включает команду USER.

**Включаемые файлы**: включаемый файл является стандартным SUBMIT файлом, с учетом всех правил и функций SUBMIT. Формат команды INCLUDE демонстрируется ниже.

$INCLUDE filename parm1 parm2 parm3 ...

Имя файла команды INCLUDE должно иметь расширение файла SUB, чтобы обозначить SUBMIT файл, и параметры, являющиеся стандартными параметрами SUBMIT. Файл INCLUDE может вложить до четырех SUBMIT в команду SUBMIT.

**Встроенные управляющие символы**: Управляющие символы могут быть встроены в файл SUBMIT, предварив заглавным буквам символ ASCII стрелка вверх "^". Например, введите ^X, чтобы встроить Ctrl-X. Встроенные управляющие символы не интерпретируются MP/M II, но могут быть полезны для программ, которые выполняет SUBMIT.

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 12, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Модификации утилиты SPOOL

Применимые продукты и номера версий: MP/M II™ версии 2.0

SPOOL может возвратить сообщение об ошибке, если не найден файл для помещения в очередь печати. Чтобы использовать это изменение, SPOOL разделен на транзитную часть (SPOOL.PRL) и резидентную часть. Транзитная часть разбирает хвост команды, открывает файл, передает файл в очередь печати (названную SPOOLQ) и выводит на экран сообщение об ошибке, если не удается открыть файл последовательно. Затем транзитная часть заканчивает себя.

Выполните команду SPOOLQ, если вы не хотите использовать сегмент памяти для буферизации распечатываемого файла. Однако, сообщения об ошибках не возвращаются. Примеры команд для буферизации распечатываемого файла приведены ниже.

SPOOL file1.typ,file2.typ ...

Процесс SPOOL передает хвост команды, проверяет ошибки, и отправляет файл очереди печати (SPOOLQ).

SPOOLQ file1.typ,file2.typ ...

Хвост команды отправлен очереди печати (SPOOLQ), обходя проверку ошибок или сообщения об ошибках.

Утилита SPOOL устанавливает свой приоритет равным 201. Большинство процессов выполняется до диспетчера очереди печати. Чтобы изменить приоритет диспетчера очереди печати, измените файл SPOOL.BRS. Удостоверьтесь, что у вас есть резервная копия SPOOL.BRS перед использованием RDT для выполнения следующих изменений.

0A>**rdt spool.brs**

[MP/M II] DDT VERS 2.0

NEXT PC

0B00 0100

-**s3b5**

03B5 C9 **c8**

03B6 00 .

-**ispool.brs**

-**w14**

-**g0**

0A>**gensys**

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 13, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Блокирование/Разблокирование записей

Применимые продукты и номера версий: MP/M II™ версии 2.0

Блокирование/разблокирование записей позволяет нескольким процессам совместно использовать доступ к одному файлу. Файлы открыты в неблокируемом режиме. Запись, заблокированная одним процессом, может быть только считана другим процессом, однако, заблокированная запись может быть изменена начальным процессом. Избегайте чтения заблокированных записей, чтобы предотвратить чтения обновляющихся в данный момент данных. Чтобы избежать чтения заблокированных записей процесс может попытаться заблокировать запись. Если попытка завершается неудачей, запись не читают. Следующий сегмент кода демонстрирует, как заблокировать записи.

mvi c,2ch ; Установка многосекторного счетчика

mvi e,# ; # = число секторов

call BDOS ; 1 <= # <=16

mvi c,2ah ; Блокируемая запись

lxi d,fcb ; Запись будет блокирована

call BDOS

Следующий сегмент кода демонстрирует, как разблокировать записи.

mvi c,2ch ; Установка многосекторного счетчика

mvi e,# ; # = число секторов

call BDOS ; 1 <= # <=16

mvi c,2bh ; Разблокируемая запись

lxi d,fcb ; Запись будет разблокирована

call BDOS

fcb:

db 0,'DATA',A0H,20H,20H,20H,'DAT',0

ds 20

db 10,0,0 ; начиная с записи 10

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 14, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Усовершенствования GENSYS

Применимые продукты и номера версий: MP/M II™ версии 2.0

Улучшения GENSYS включают следующие новые функции и возможности.

**Механизм автоматического включения резидентных системных процессов**: механизм автоматической генерации системы GENSYS может быть изменен, чтобы по умолчанию включать все файлы RSP на диске. Введите GENSYS $AR для автоматического включения файлов RSP. Параметр R должен использоваться в сочетании с параметром A. Измените расширение файла у файлов, которые вы хотите исключить из GENSYS.

**Восстановление после ошибок**: если встречается ошибка при работе в автоматическом режиме (параметр $A), GENSYS перезапускается в ручном режиме.

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 15, 14.09.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Изменение приоритета SPOOL.PRL

Применимые продукты и номера версий: MP/M II™ версии 2.0

Утилита SPOOL устанавливает свой приоритет равным 201, поэтому, большинство других процессов выполняется до диспетчера очереди печати. Измените файл SPOOL.PRL, чтобы изменить приоритет до диспетчера очереди печати. Если серийный номер вашего продукта между 4-000-00001 и 4-000-00464, установите патч 11 MP/M II прежде, чем изменить приоритет SPOOL.PRL по умолчанию.

Удостоверьтесь, что у вас есть резервная копия SPOOL.PRL перед использованием DDT для внесения следующих изменений.

0A>**ren spool.sav=spool.prl**

0A>**ddt spool.sav**

[MP/M II] DDT VERS 2.0

NEXT PC

0980 0100

-**s269**

0269 C9 **c8**

026A CD .

-**ispool.prl**

-**W11**

-**g0**

0A>

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.

**Операционная система MP/M II™ Версия 2.0**

**Указание по применению 16, 01.12.81**

Авторское право © 1981 принадлежит Digital Research

MP/M II является торговой маркой Digital Research

Изменение функций клавиш BACKSPACE и RUBOUT

Применимые продукты и номера версий: MP/M II™ версии 2.0

Программа: RESBDOS

В MP/M II клавиша BACKSPACE или Ctrl-H (ASCII 08H) делает возврат с удалением, удаляя последний знак в буфере команд. Клавиша RUBOUT (ASCII 7FH) или клавиша DELETE удаляет последний символ в буфере команд и повторяет его на экран.

Процедура инвертирования функций клавиш BACKSPACE и RUBOUT:

0A>**ren resbdos.sav=resbdos.spr**

A>**ddt resbdos.sav**

[MP/M II] DDT VERS 2.0

NEXT PC

0f80 0100

-**sc0a**

0C0A 08 **7f**

0C0B C2

-**sc22**

0C22 7F **08**

0C23 C2 .

-**iresbdos.spr**

-**w1d**

-**g0**

0A>**gensys**

Процедура, чтобы сделать RUBOUT идентичной BACKSPACE:

0A>**ren resbdos.sav=resbdos.spr**

0A>**ddt resbdos.sav**

[MP/M II] DDT VERS 2.0

NEXT PC

0f80 0100

-**lc26**

0C26 **MOV A,B**

0C27 **ORA A**

0C28 **JZ 09F6**

-**ac26**

0C26 **jmp a0e**

0C29

-**bb28,1**

-**iresbdos.spr**

-**w1d**

-**g0**

0A>**gensys**

Процедура, чтобы сделать BACKSPACE идентичной RUBOUT:

0A>**ren resbdos.sav=resbdos.spr**

0A>**ddt resbdos.sav**

[MP/M II] DDT VERS 2.0

NEXT PC

0f80 0100

-**Ic0e**

0C0E **MOV A,B**

0C0F **ORA A**

0C10 **JZ 09F6**

-ac0e

0C0E **jmp, a26**

0c11

-**bb10,1**

-**iresbdos.spr**

-**w1d**

-**g0**

0A>gensys

Лицензированным пользователям предоставляется право включать эти улучшения в программное обеспечение MP/M II версии 2.0.