

ETH Цюрих

# OBERON для PC на базе MS-DOS

Автор(ы): Disteli, Andreas R.

1993г.

# Оглавление

Глава 1. Введение.....	2
Глава 2. Процессоры Intel.....	3
2.1 Модель памяти для Oberon.....	8
2.2 Переключение между Защищенным режимом и Реальным режимом.....	9
2.3 Вызовы операционной системы и ловушки.....	10
2.4 Файловая система DOS.....	11
2.5 Вызовы BIOS.....	13
Глава 3. Модель памяти.....	14
Глава 4. Компилятор.....	18
4.1 Архитектура процессора.....	18
4.2 Специфические особенности.....	22
4.3 Формат Объектного файла.....	23
4.4 Размер программы.....	24
Глава 5. Загрузчик и ядро.....	25
5.1 Метазагрузчик.....	25
5.2 Загрузчик модулей.....	26
5.3 Ядро.....	30
Глава 6. Модуль DOS.....	31
Глава 7. Дисплей.....	35
Глава 8. Арифметика с плавающей запятой.....	38
8.1 Математические модули .....	39
Глава 9. Файловая система.....	41
Ссылки.....	45

## Глава 1. Введение

Oberon - это полноценная операционная система и язык, разработанный в Швейцарском институте компьютерных систем в Цюрихе. Весь проект был ориентирован на один конкретный тип машин, а именно на персональную рабочую станцию Ceres. Система и язык Oberon привлекли большое внимание за пределами ETH, но не были доступны на коммерческих машинах. Это было большим препятствием для распространения.

Несколько лет назад несколько сотрудников института начали переносить Oberon на коммерческие аппаратные платформы, такие как DECstation, SPARC, RS/6000 и Macintosh-II. Это был хороший шанс распространить как систему, так и язык Oberon среди большого числа пользователей. Но большинство людей имеют доступ лишь к простым персональным компьютерам на базе MS-DOS. Поэтому до сих пор Oberon не был доступен на наиболее часто используемых компьютерах во всем мире.

С портом для DOS-машин открывается огромный круг новых потенциальных пользователей. Для целей обучения было бы интересно использовать и Oberon, поскольку он оказался очень адекватным языком и системой для студентов. Еще следует учитывать, что в последние годы машины PC становятся все более недорогими, что делает возможным для каждого иметь компьютер для работы дома. По всем этим причинам летом 1991 года мы решили начать проект DOS-Oberon, нацеленный на систему и язык Oberon в сочетании с самыми популярными компьютерами в мире.

Перенос был разбит на несколько частей. Прежде всего, необходимо было перенести компилятор, хотя бэкенды компилятора существовали для всех вышеперечисленных машин, ни одна из них не генерировала код для семейства intel 80x86. Таким образом, пришлось переписать всю часть компилятора,

связанную с генерацией кода.

## **Язык**

Было решено написать всю систему на языке Oberon, за исключением некоторых специальных модулей, которые по соображениям производительности должны быть написаны на ассемблере. Поэтому пришлось реализовать ассемблер, генерирующий intel-код и работающий под Oberon. Только один модуль был написан под родным DOS: мета-загрузчик (загрузчик модулей) был реализован на ассемблере Microsoft. Вся система была разработана на рабочей станции Ceres под управлением родного Oberon. Все модули были кросс-компилированы для загрузчика.

## **Хранилище**

На втором этапе была разработана среда выполнения, включая разметку и управление памятью, а также механизм загрузки модулей. Многие проблемы возникли из-за того, что процессор должен работать в различных и несовместимых режимах (см. главу 3, Модель памяти). Другой серьезной проблемой была система отображения с ее различными стандартами и разрешениями. Обычное разрешение VGA является стандартным для всех машин. Однако вскоре выяснилось, что его размер слишком мал. Кроме того, программирование VGA-карты громоздко из-за множества регистров, которые должны быть установлены для каждой операции рисования. Это замедляет работу дисплея, но он работает на всех машинах, оснащенных VGA-картой. С другой стороны, Super-VGA обеспечивает большее разрешение, но должна программироваться отдельно, если требуется быстрый дисплей, поскольку каждая карта имеет свои собственные расширения (см. главу 7, Дисплей).

## **Ограничения**

Мы намеренно ограничили нашу реализацию процессорами 80386 и выше потому что Oberon – это полноценное 32-битное приложение, а более старые процессоры типа 80286 поддерживают только 16-битные. Кроме того, наша реализация не позволяет устанавливать специфические менеджеры памяти. Причина этого может быть найдена в необходимости монополизации защищенного режима, если только не используется специальное программное обеспечение (так называемые расширители) (см. главу 2: Среда Intel).

Нашей целью было использовать как можно меньше разработанных извне программных пакетов, таких как драйверы, специальные менеджеры и т.д. Для совместимости мы использовали стандартный драйвер мыши logitech и драйвер HIMEM.SYS для расширенной памяти (более 1 мегабайта).

# **Глава 2. Процессоры Intel**

## **8088 процессор**

В 1981 году компания IBM анонсировала первый ПК с процессором Intel 8088. В этом нет ничего особенного, если смотреть сегодняшними глазами, но в то время это был большой скачок от 8-битного к 16-битному процессору. Новый процессор мог получить доступ к 1 МБ физического адресного пространства, однако только 640 КБ были доступны для обычных программ, а остальное было зарезервировано для видеобuffers и BIOS (рис. 1). Кроме того, большинство

новых драйверов, контроллеров и т.д. были размещены в этой зарезервированной области.

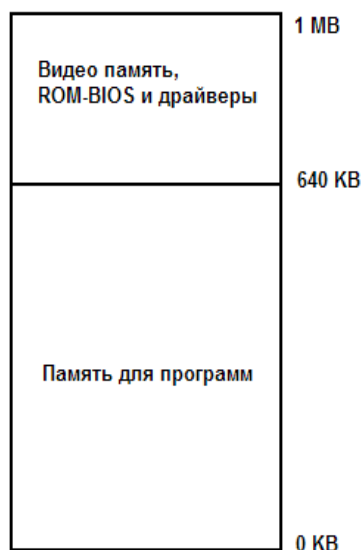


Рис 1: Карта памяти ПК с процессором 8088

### 80286 процессор

В 1982 году Intel представила новый процессор 80286, преемник процессоров 8080 и 8088 (рис. 2). Он поддерживал больший диапазон физических адресов – 16 МБ и что самое главное, был готов к многозадачности. Как следствие, пришлось изобрести метод, позволяющий сохранить область кода и данных одной программы от доступа других программ. Этот метод был назван Защищенный режим (ЗР) в противовес старому Реальному режиму (РР).

### Реальный режим (РР)

В РР каждый адрес состоит из сегмента и смещения. Сегмент можно рассматривать как часть памяти с базовым адресом и лимитом.

В случае сегмента РР, базовый адрес выравнивается по 16 байт, а лимит равен 64 КБ. Благодаря этому выравниванию базовый адрес может быть разделен на 16 и все равно остается уникальным. Эта техника экономит 4 бита, и базовый адрес, хотя он адресует память объемом до 1 МБ, может храниться в 16-битовом месте. Физический адрес вычисляется следующим образом:

```
address := segment * 16 + offset
```

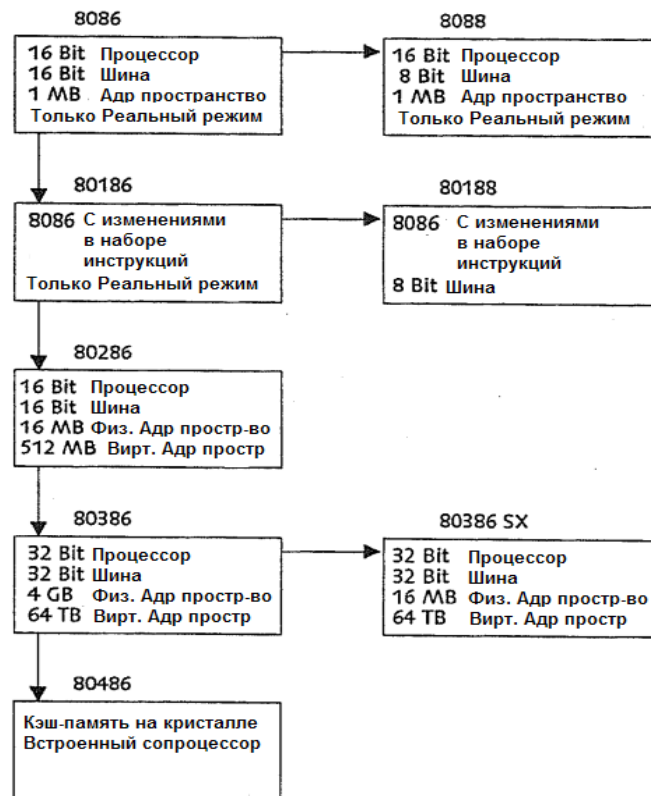


Рис 2: Семейство процессоров Intel

### Защищенный режим (3P)

С помощью сегмента и смещения, оба из которых имеют размер 16 бит, могут быть сгенерированы адреса размером до 1 МБ. В процессоре 80286 сегментная часть адреса является уже не простым адресом, а указателем на дескрипторную таблицу, содержащую серию 4-битных адресов. Добавляя 16-битное смещение, программист может получить доступ к 16 МБ памяти. Конечно, одна и та же пара сегмент:смещение может представлять разные физические адреса, в зависимости от содержимого таблицы дескрипторов. Поэтому ее часто называют виртуальным адресом (рис. 3). Во время генерации адреса сегментная часть каждого адреса должна содержаться в одном из сегментных регистров, называемых SS, CS, DS или ES (пояснения см. в таблице 1). Смещение находится либо в памяти либо в одном из регистров процессора BX, BP, SI или DI. Другие регистры, AX, CX, DX и SP, не могут быть использованы для вычисления адреса. Эти регистры являются выделенными и, как таковые, являются причиной многих проблем. Желательно использовать все регистры в общем случае.

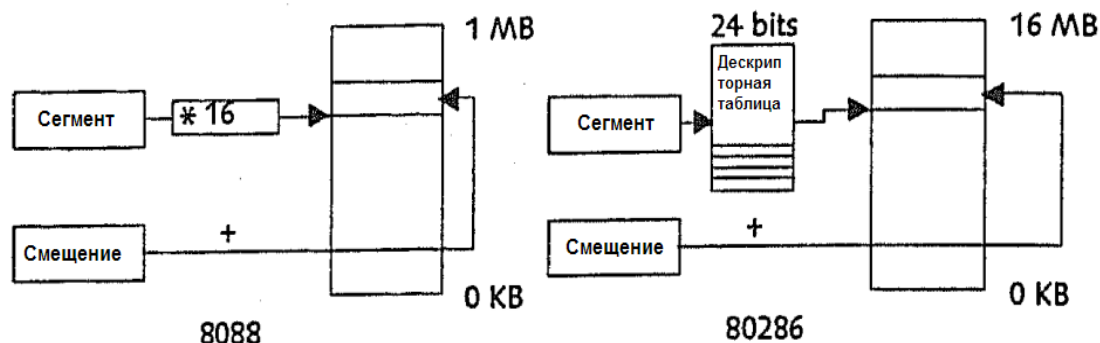


Рис 3: Образование адреса

### Сегментные регистры:

CS = Сегмент кода                      ES = Дополнительный сегмент  
DS = Сегмент данных                SS = Сегмент стека

### Регистры процессора:

AS = Аккумулятор                      SI = Индекс источника  
BX = База                                DI = Индекс приемника  
CX = Счетчик                            BP = Указатель кадра  
DX = Регистр данных                SP = Указатель стека

Таблица 1: Регистры процессора 80286

### Таблица дескрипторов

80286 означает сосуществующие задачи, выполняемые под безопасной операционной системе. Каждая задача имеет свой собственный диапазон локальных адресов, и она также может получить доступ к глобальному диапазону адресов операционной системы. Это позволяет предотвратить перезапись данных одной задачи другими задачами, при этом доступ к общим данным остается возможным. Однако это подразумевает необходимость в глобальной дескрипторной таблице (GDT) для системы и локальной дескрипторной таблицы (LDT) для каждой задачи. Кроме того, каждая задача работает на определенном уровне привилегий, который может быть использован для предоставления различных прав доступа. Каждый так называемый селектор, выглядит следующим образом (рис. 4):

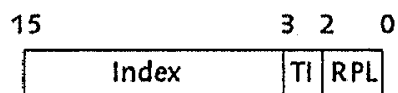


Рис 4: Селектор

Биты 0 и 1 указывают на запрашиваемый уровень привилегий (RPL) для данного сегмента, бит 2 определяет активную таблицу GDT (TI=0) или LDT (TI=1). Биты с 3 по 15 используются как номер индекса в таблице дескрипторов. Каждый дескриптор содержит информацию об описываемом сегменте он включает базовый адрес, уровень привилегий, размер и тип

сегмента (рис. 5). Базовый адрес имеет размер 24 бита. Таким образом, сегмент может начинаться в любом месте в пределах возможного диапазона адресов 16 МБ. Поле limit указывает размер сегмента в байтах. Это ограничивает размер сегмента до 64 КБ. Другими словами, РМ процессора 80286 имеет те же ограничения сегментации, что и РР, но он может адресовать больший объем памяти. За подробной информацией обращайтесь к [Dun90].

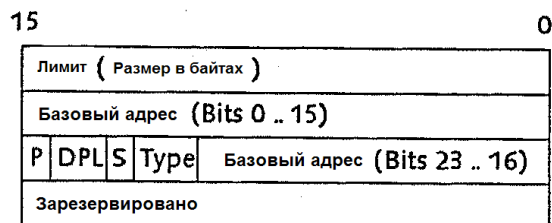


Рис 5: 80286 Дескриптор сегмента

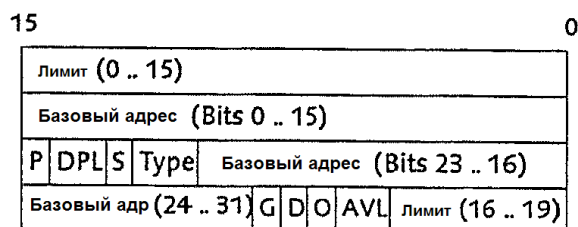


Рис 6: 80386 Дескриптор сегмента

## Прерывания

Еще одним важным моментом является обработка прерываний. В РР таблица прерываний начинается с физического адреса 0. Она содержит 4-байтовый вектор для каждого канала прерывания. Каналы прерываний нумеруются от 0H до 0FFH. Таким образом, первые 1024 байта зарезервированы для векторов прерываний. В РР таблица дескрипторов прерываний (IDT) может быть расположена произвольно и она также содержит 8-байтовые дескрипторы, такие как LDT и GDT. Еще одной приятной особенностью 80286-го заключается в следующем: Intel посчитала, что старый РР постепенно исчезнет, и они предоставили только механизм для перехода из РР (которое является начальным состоянием после загрузки) в ЗР. Однако нет никакой возможности переключить процессор обратно из состояния ЗР в РР, кроме как путем перезагрузки процессора!

## 80386 процессор

В 1985 году 80386 процессор стал доступен. Это настоящий 32-разрядный процессор с 32-битной аппаратной шиной. По сравнению со старым 80286, есть новые возможности, которые делают процессор более удобным для использования. Например, есть новые режимы адресации (масштабная индексация) и все регистры могут быть использованы для вычисления адреса. К сожалению, все еще есть некоторые специальные инструкции, которые требуют выделенных регистров, как например, запись в порт или чтение из порта, инструкция деления и т.д. Дескриптор сегмента также был изменен (рис. 6). Теперь можно создать сегмент размером более 64 КБ. База адреса увеличена до 32 бит, предел сегмента – до 20 бит, идополнительный бит в

расширенной части дескриптора указывает, является ли ограничение в байтах ( $G=0$ ) или в страницах размером 4 КБ ( $G=1$ ). Таким образом, теперь возможен размер сегмента в 4 ГБ, в то время как программы, написанные для старой версии процессора, остаются исполняемыми (обратная совместимость).

[Dun90]. Кроме того, переключение обратно из 3Р в РР теперь возможно без операции сброса. Но это не делается просто установкой процессора обратно в реальный режим. Все сегментные регистры должны иметь законное значение РР после этого. Можно подумать, что это будет сделано простым присвоением старого значения селектора РР соответствующему сегментному регистру. Однако, когда в РМ информация о сегменте хранится в теновом регистре, который недоступен программисту. Возвращаясь в РР, теневой регистр, похоже, все еще проверяется, хотя все размеры и ограничения указаны для 3Р. Следовательно, перед переключением обратно из 3Р в РР, селекторы должны быть установлены на дескриптор с действительными значениями для РР (предельная гранулярность ...), иначе произойдет сбой. Так называемый режим V86 здесь не рассматривается см. [Dun 90]. Режим V86 позволяет нескольким программам РР работать одновременно. Адрес вычисляется как в РР, без использования GDT и LDT. Таким образом, каждая РР программа имеет свое обычное адресное пространство в 1 МБ, как на 8086, хотя и работает в защищенном режиме.

## 2.1 Модель памяти для Oberon

Необходимо было найти модель памяти, которая обеспечивала бы простой и быстрый доступ к памяти и не была бы сложной для генерации кода.

Учитывая приведенное выше, есть выбор:

### 1. Использовать только реальный режим

- + переключатели режимов не нужны
- Ограничение сегмента в 64 КБ.  
Только 640 КБ обычной памяти.  
Работает со старыми версиями 80x86.  
Компилятор/загрузчик должен поддерживать сегментацию.  
dedicated registers for calculations

### 2. Использовать 80286 3Р

- + 16 МБ доступной памяти
- Ограничение сегмента в 64 КБ  
компилятор/загрузчик должен поддерживать сегментацию  
выделенные регистры для вычислений  
необходимы переключатели режимов

### 3. Использовать 80386 3Р. Один большой сегмент кода и данных.

- + 4 ГБ доступной памяти  
Регистры общего назначения  
Компилятор/загрузчик не обязан поддерживать сегментацию  
Линейные адреса  
Отсутствие изменений сегментных регистров в 3Р



- Необходимость переключения режимов

#### 4. Использовать 80386 ЗР. Отдельные сегменты кода и данных.

- + 4 Гб доступной памяти  
Регистры общего назначения  
Линейные адреса
- Необходимость переключения режимов  
Сегментный регистр может изменяться во время работы в ЗР  
Специальная обработка адресов данных и кода

Наиболее простой и выгодной моделью представляется модель 3 с кодом и данными в одном сегменте. Модели 1 и 2 имеют слишком много ограничений, а модель 4 требует особого обращения в некоторых ситуациях (например, загрузчик должен загрузить код в сегмент кода и поэтому ему нужен доступ на запись для этого сегмента. Поскольку сегмент кода обычно доступен только для чтения, настройки такого сегмента должны быть изменены для загрузки модуля).

## 2.2 Переключение между Защищенным режимом и Реальным режимом.

Код для переключения между реальным и защищенным режимами должен быть помещен в 16-битный сегмент, чтобы не превышать ограничение в 640 КБ, так чтобы он был выполнен как в реальном, так и в защищенном режиме. Следующие процедуры показывают отдельные шаги этих режимов. Они будут подробно описаны в главе 5, где рассматриваются загрузчик и ядро.

```
PROCEDURE SwithToProtectedMode
BEGIN
    Сохранить все сегментные регистры;
    Отключить прерывания;
    Загрузить IDT защищенного режима;
    Получить указатель стека защищенного режима;
    Установить бит защиты в CR0;
    Заполнить очередь предварительной выборки;
    Выполнить FAR-переход для установки регистра CS;
    Установить селекторы ЗР;
    Включить прерывания;
END SwithToProtectedMode.
```

```
PROCEDURE SwithToRealMode
BEGIN
    Сохранить все сегментные регистры;
    Выключить прерывания;
    Загрузить PP IDT;
    Получить указатель стека реального режима;
```

```
Очистить бит защиты в CR0;  
Заполнить очередь предварительной выборки;  
Выполнить FAR-переход для установки регистра CS;  
Установить селекторы PP;  
Включить прерывания;  
END SwithToRealMode.
```

## 2.3 Вызовы операционной системы и ловушки.

MS-DOS является однопользовательской операционной системой.

Как следствие, когда приложение запущено, операционная система не имеет никакого контроля и она вызывается только через ловушку или прерывание. Эти события можно разделить на 3 группы:

- Системные вызовы
- Синхронные прерывания
- Асинхронные прерывания

Каждая системная программа вызывается через прерывание, которое активируется прикладной программой (программное прерывание). Система предоставляет большую библиотеку внутренних функций. Кроме того, устанавливаемые драйверы предлагают свои функции обычно через системный вызов. Обычно параметры для системного вызова передаются в регистрах. Например, вывод одного символа на экран выполняется следующей процедурой:

```
MOV AL, 02h    ;выбор функции  
MOV DL, char   ;символ для отображения  
              ;параметры, хранящиеся в регистрах  
INT 21h        ;номер системного вызова. Освобождение прерывания
```

Синхронные прерывания являются исключениями. Они вызываются ошибками выполнения, такими как "деление на ноль", "недопустимый опкод" и т.д. В отличие от них, асинхронные прерывания вызываются аппаратными устройствами, например, "тиканьем таймера".

Системные вызовы должны выполняться в RM, чтобы гарантировать правильное функционирование, в то время как прерывания могут обрабатываться и в RM, если существует соответствующая запись в PM IDT. В противном случае прерывание должно быть перенаправлено в RM. Это задача так называемого расширителя.

### Расширитель

Расширитель должен позаботиться о правильной инициализации GDT, LDT и IDT, предполагая, что мы начинаем в RM. Кроме того, он должен правильно обрабатывать прерывания, например, обрабатывать их в RM, если это возможно, и/или переключать процессор в RM, имитируя прерывание снова после установки параметров и переключаясь обратно в RM, когда обработка прерывания завершена.

Подводя итог, для каждого прерывания в RM должен быть установлен обработчик вида:

```
SwitchToRealMode;
Повторный вызов прерывания;
SwitchToProtectedMode;
```

Эту процедуру можно проиллюстрировать следующей блок-схемой (рис.7):

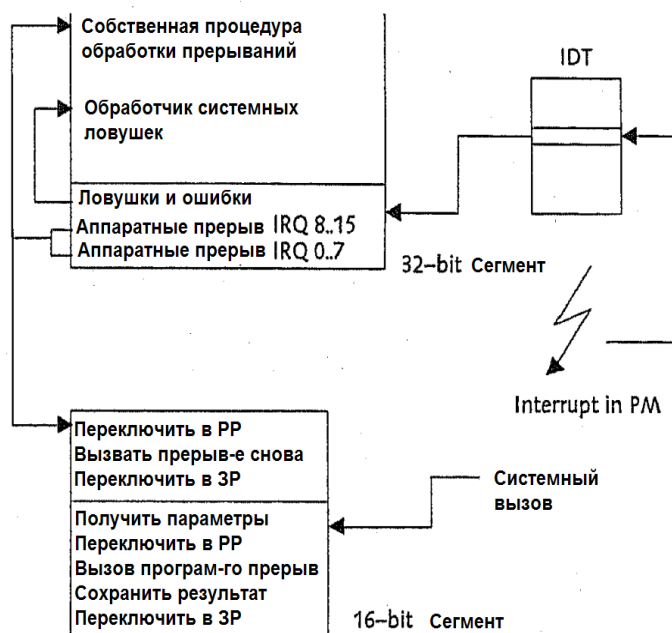


Рис 7: Структура расширителя

## 2.4 Файловая система DOS

DOS предлагает два различных вида сервиса для управления файлами и управления каталогами [Dun90, Bro91]. Первый вид – это доступ через блоки управления файлами (FCB). Его больше не следует использовать. Первые версии DOS поддерживали FCB, потому что операционная система CP/M работала таким же образом. Функции FCB все еще поддерживаются, но файловые дескрипторы заменили их в более поздних версиях. Они более простые и более гибкие, чем FCB. Аналогичный метод используется в UNIX. Следующий список дает обзор используемых файловых операций. Полный справочник см. в [Bro91].

## **Файловые функции используемые в Oberon:**

Get Default Drive (int 21H, функция 19H):

Получает текущий диск по умолчанию.

Create Subdirectory (int21H, функция 39H):

Создает новую поддиректорию под заданным именем.

Removes Subdirectory (int 21H, функция 3AH):

Удаляет подкаталог.

Change Directory (int 21H, функция 3BH):

Устанавливает текущий каталог на указанный.

Open New File (int 21H, функция 3CH):

Открыть новый файл или усечь уже существующий файл под этим именем.

Open Old File (int 21H, функция 3DH):

Открывает уже существующий файл.

Close File (int 21H, функция 3EH):

Промывка буферов, обновление информации о каталоге и аннулирование дескриптора файла.

Read File (int 21H, Function 3FH):

Чтение из файла и помещение данных в буфер.

Write File (int 21H, функция 40H):

Запись данных из буфера в файл.

Delete File (int 21H, функция 41H):

Удалить указанный файл.

Set Position (int 21H, функция 42H):

Установить позицию в указанное место в файле.

Duplicate Handle (int 21H, функция 45H):

Когда файл должен быть обновлен на диске, но должен оставаться открытым, можно дублировать хэндл этого файла. Затем файл может быть закрыт с помощью этого новым хэндлом. Старый хэндл по-прежнему дает доступ к открытому файлу.

Get Current Directory (int 21H, функция 47H):

Определить текущий каталог.

Get First File (int 21H, функция 4EH):

Берет шаблон имени и ищет совпадающие имена файлов.

Можно использовать символы подстановки (\*).

Get Next File (int 21H, функция 4FH):

Получает следующий файл, соответствующий шаблону имени, который был указан в функции Get First File. Эта функция должна быть вызвана первой.

Rename File (int 21H, функция 56H):

Переименовывает файл из старого в новый. Если старый и новый находятся на одном логическом томе, переименование действует как

перемещение.

Get Date and Time (int 21H, функция 57H):  
Считывает дату и время указанного файла.

Set Handle Count (int 21H, функция 67H):  
Устанавливает количество файлов, доступных вызывающей программе.

### Концепция райдера

Файловая система Oberon гораздо более гибкая. Oberon вводит концепцию райдера. Райдер – это механизм доступа, который ассоциируется с файлом и обозначает позицию внутри этого файла. Чтение и запись выполняются под контролем этого райдера. Более чем один райдер может быть связан с одним и тем же файлом одновременно. В обычных моделях это соответствует нескольким открытым файлам. См. [Wir92] для более подробной информации о файловой системе Oberon. DOS не знает понятия райдер. Хотя возможно иметь два разных хэндла, указывающих на один и тот же файл, они жестко связаны друг с другом: Изменение позиции одного дескриптора также изменяет информацию о положении другого дескриптора. Философия райдера полностью отсутствует в DOS.

### Имена файлов

Еще одним неудовлетворительным фактом являются короткие имена файлов. Dos допускает только 8 символов плюс 3 символа расширения разделенные точкой, в качестве имени файла. Oberon работает с полными 32-символьными именами файлов, которые могут использовать более одной точки в качестве разделителя. Таким образом, необходимо найти решение которое отображает имена Oberon на имена DOS. В главе 9 описана реализация файловой системы Oberon под DOS.

## 2.5 Вызовы BIOS

BIOS (Basic In-Out System) содержит набор процедур, обеспечивающих интерфейс к компонентам системы, таким как экран, клавиатура, дисковый контроллер, мышь и т.д. Она также содержит тестовые процедуры и загрузочную программу, которая загружает операционную систему с диска [Dun90]. В BIOS используются следующие группы вызовов. Для полного обзора всех доступных вызовов см. [Bro91].

#### Прерывание 10H, **Видео**

Функция:	Назначение:
0H	Установить видеорежим
2H	Установить положение курсора
3H	Получить положение курсора
9H	Запись символа

#### Прерывание 11H, **Среда**

Функция:	Назначение:
Нет номера	Получить список оборудования

Прерывание 13H, **Низкоуровневый дисковый ввод/вывод**

Функция:	Назначение:
0H	Сброс диска
2H	Считать сектор
3H	Поместить сектор
5H	Форматировать дорожку

Прерывание 14H, **Последовательный порт**

Функция:	Назначение:
0H	Инициализация порта
1H	Запись символа в порт
2H	Считать символ из порта
3H	Получить статус порта

Прерывание 16H, **Клавиатура**

Функция:	Назначение:
0H	Получить нажатие клавиши
2H	Получить флаги сдвига
11H	Проверка наличия расширенного нажатия клавиши

Прерывание 17H, **Параллельный порт**

Функция:	Назначение:
0H	Запись символа в порт
1H	Инициализация порта
2H	Получить статус порта

Прерывание 1AH, **Таймер**

Функция:	Назначение:
0H	Получить системное время (тики с полуночи)
2H	Получить реальное время
3H	Установить реальное время
4H	Получить реальную дату
5H	Установить реальную дату

Прерывание 33H, **Мышь**

Функция:	Назначение:
0H	Сброс драйвера
3H	Получить положение и кнопки
4H	Установить положение
7H	Установить горизонтальные границы
8H	Установить вертикальные границы
0FH	Отношение Mickey к пикселю

## Глава 3. Модель памяти

### Низкий уровень и память

На данном этапе мы хотим объявить выражения, между которыми должны быть четкие разграничения. Во-первых, существует так называемая низкая или обычная память. Она обозначает первый мегабайт физической памяти и является реликтом старых версий процессора, которые могли обращаться только к 1 МБ памяти.

Во-вторых, существует большая или расширенная память, которая обозначает всю физическую память свыше 1 МБ.

### **Модель сегмента**

Как уже говорилось в главе 2, существует несколько причин для использования односегментной модели в качестве адресного пространства для системы Oberon. Наиболее важное всего то, что она проста в реализации и быстра. Для определения точной модели разметки, необходимо было учесть следующее обстоятельство:

1. Видеобуфер должен содержаться в том же сегменте, что и код и данные. Таким образом, нет необходимости в изменении сегмента, что обеспечивает фактический доступ к памяти дисплея.
2. Куча для динамического распределения памяти должна быть непрерывным участком памяти.
3. Должен быть выделен стек для активации процедур.
4. Часть обычной памяти для выделения низкоуровневых модулей необходим. Их назначение – загрузка Ядра и передача ему управления ему и управление собственной памятью. Перед загрузкой Ядра и инициализации, память выделяется под управлением DOS операционной системы. Ее процедуры распределения памяти управляют только обычной памятью, что подразумевает необходимость этой части памяти с малым объемом части. В противном случае, сохранение деаллокации используемой памяти не гарантируется.
5. Вся физическая память должна быть доступна для адресации Oberon. BIOS и DOS-вызовы часто используют адреса буферов для получения или возврата результаты. Эти буферы выделяются в обычной памяти, поэтому они также должны быть адресуемы Oberon для доступа к данным.

### **Последовательность загрузки DOS**

Низкая память состоит из нескольких частей. После последовательности загрузки, первый КБ содержит все 256 векторов прерываний. Следующие 512 байт используются для таблиц ROM BIOS. Затем выделяются резидентные драйверы устройств выделяются драйверы резидентных устройств: BIOS MS-DOS (IO.SYS) и MS-DOS Kernel (MSDOS.SYS), затем таблицы MS-DOS, дисковые буферы, блоки управления файлами и возможные устанавливаемые драйверы устройств. Наконец, есть резидентная часть оболочки MS-DOS, называемая COMMAND.COM. Резидентная часть используется для завершения программ, которые были запущены COMMAND.COM. Ниже границы 640 Кб находится переходная часть оболочки. Она получает команды с клавиатуры или из пакетного файла и выполняет их. После каждого вызова переходная часть проверяется на согласованность и возможно, перезагружается резидентной частью. Свободная оперативная память между резидентной и переходной частями оболочки доступна для программ. Выше 640 КБ находятся ПЗУ-БИОС и другие ПЗУ и ОЗУ, т.е. память дисплея. На следующих рисунках (рис. 1+2) показана структура нижней памяти. Для полного описания последовательности загрузки обратитесь к [Dun90a].

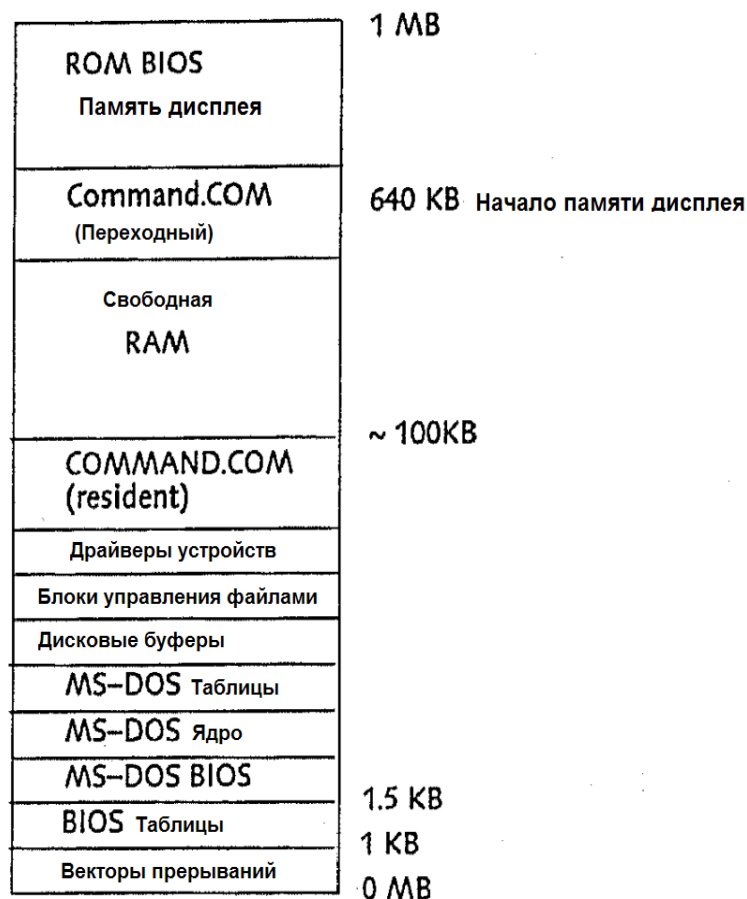


Рис 1: Структура памяти после загрузки [Dun90a].

### Последовательность загрузки Oberon

Прежде чем управление памятью Oberon получит контроль, под модули Loader.Obj, Modules.Obj и Kernel.Obj и их дескрипторы выделяется 1 МБ. Loader.Obj обеспечивает загрузку модулей в память и перемещение адресов переменных и вызовов процедур.

Modules.Obj является интерфейсом между Loader.Obj и системой Oberon. Загрузка модуля обрабатывается Modules.Obj, который вызывает процедуры Loader.Obj.

Loader.Obj, Kernel.Obj управляет распределением памяти и системными вызовами.

Более подробную информацию можно найти в главе 5: Загрузчик и Ядро. Как только Kernel.Obj инициализируется Loader.Obj, он получает всю информацию о расширенной памяти (начальный адрес). Затем, все запросы на память приводят к выделению памяти в расширенной памяти. Минимальная потребность в расширенной памяти составляет 2 МБ. Хотя для базовой системы 2 МБ будет достаточно, мы рекомендуем 4 МБ или больше при работе с приложениями, интенсивно использующими память. Сборщик мусора Oberon работает только в части кучи расширенной памяти. Для этого есть причины: Во-первых, непрерывная часть памяти предполагается для сборки мусора. Это означает, в частности что куча не может быть выделена ниже области видеобуфера. Вторая причина заключается в том, что область низкой памяти является идеальным местом для размещения стека.



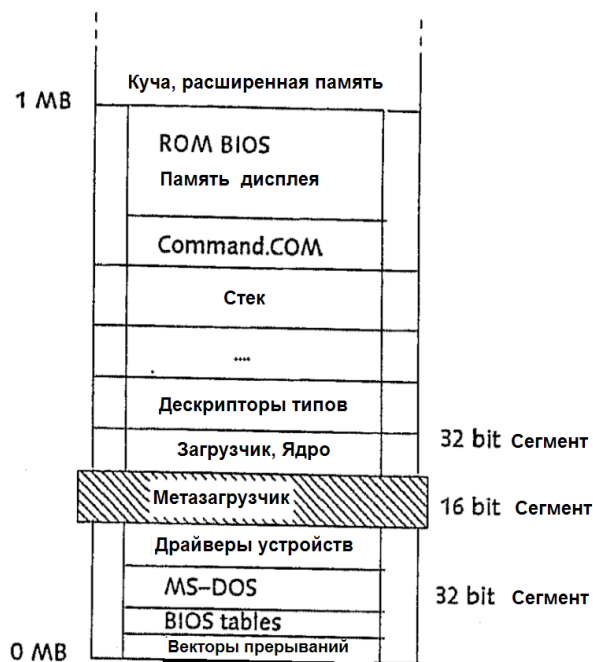


Рис. 2: Схема памяти для Oberon

### Расположение памяти

Как видно из рис. 2, вся физическая память покрывается 32-битным сегментом. Это означает, что все адреса могут быть доступны Oberon. 16-битный сегмент (заштрихован) метазагрузчика находится где-то в памяти, в зависимости от структуры памяти во время загрузки Oberon. Обратите внимание, что к этому небольшому сегменту Oberon может получить доступ, потому что он полностью встроен в 32-битный сегмент. Этот метазагрузчик является исполняемым под DOS. Он загружает фактический загрузчик модулей и обрабатывает системные вызовы, например, берет на себя функцию так называемого расширителя. См. также главу 5: Загрузчик и ядро. Недостатком этой модели является объем оперативной памяти, которая не используется ни управлением памятью Oberon, ни DOS. Однако управление этой памятью, особенно под Oberon была бы слишком сложной. Эта память может быть использована для дополнительных модулей драйвера, которые должны находиться в низкой памяти, поскольку они используют буферные адреса. На самом деле, компилятор Oberon обеспечивает некоторую поддержку для загрузки модулей в низкую память. См. главы 4 и 5 для более подробной информации. На рисунке 3 показана модель времени выполнения с различными селекторами и регистрами. Сегмент кода и сегмент данных системы Oberon начинаются с отметки одного и того же адреса. Это облегчает загрузку и инициализацию модулей, потому что сегмент кода, в который загружается модуль обычно защищен от записи. Два сегмента ES и FS указывают на одну и ту же позицию, но они не используются, за исключением ES для некоторых специальных системных вызовов GS установлен на начало видеобуфера. Таким образом, буфер дисплея может быть доступен без добавления смещения от начала сегмента до начала видеобуфера. Два регистра ESP (расширенный указатель стека) и EBP (расширенный базовый указатель или указатель кадра, соответственно) указывают на сегмент стека. Они не используются для других целей. EIP (расширенный указатель инструкции) показывает текущую позицию выполнения.

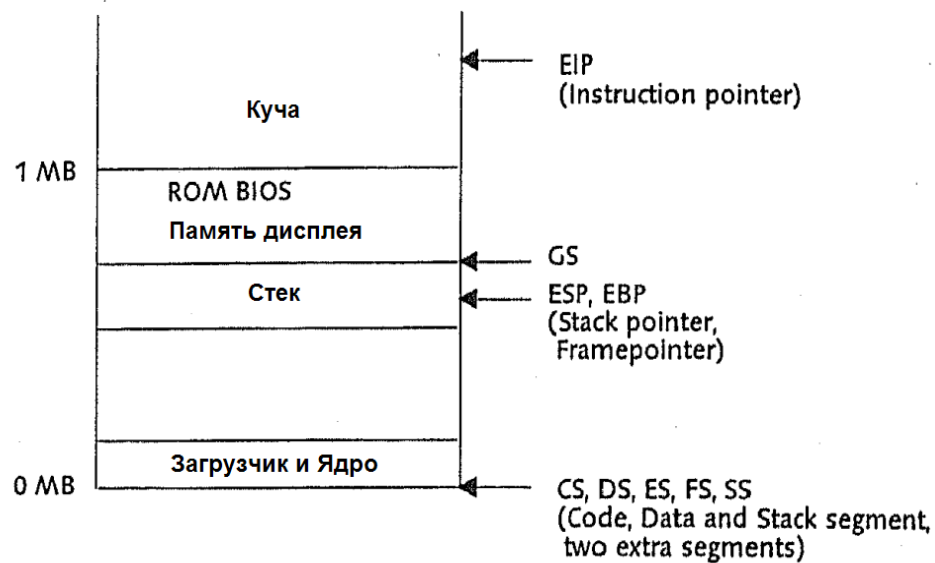


Рис 3: Модель времени выполнения (Run-time).

## Глава 4. Компилятор

Компилятор был первой программой, которую нужно было перенести на DOS. Без компилятора не было возможности перевести программы Oberon в объектный код Intel. Мы решили взять стандартный однопроходный компилятор Oberon [Wir92] за основу и адаптировать части бэкенда Intel. Оригинальный бэкенд производил код для NS32000. Этот микропроцессор имеет несколько преимуществ перед чипом Intel. В следующем обзоре сравниваются два процессора.

### 4.1 Архитектура процессора.

80386 предлагает восемь 32-битных регистров, два из которых уже используются для управления стеком. Каждый регистр предоставляет доступ либо к битам 0-5 (16 бит, совместимость со старыми процессорами) или 0-31 (расширенный 32-битный доступ). Биты 0-7 и биты 8-15 регистров EAX, EBX, ECX и EDX можно также использоваться отдельно.

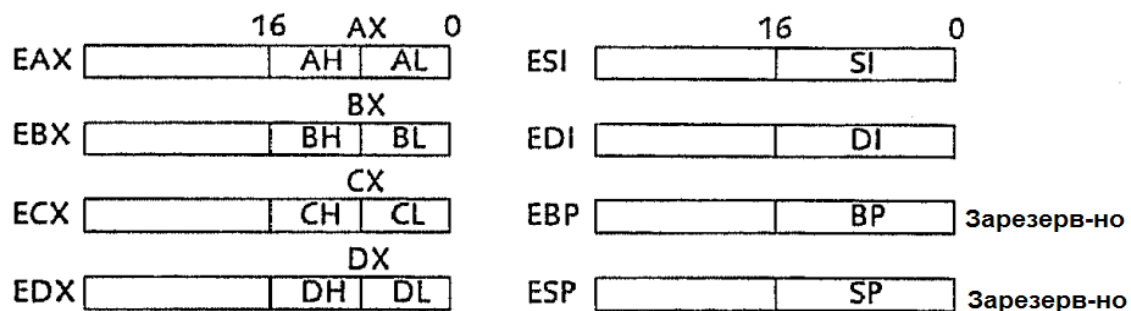


Рис 1: Регистры Intel

## Регистры NS 32000

NS 32000 имеет восемь регистров общего назначения, доступ к каждому из них может осуществляться по двойному, пословному и побайтному принципу.

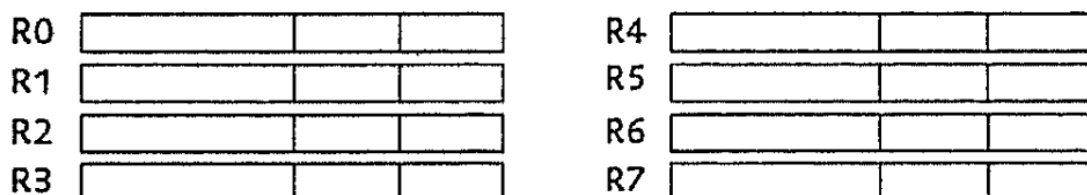


Рис 2: Регистры NS

## Режимы адресации

Расчет эффективного адреса прост:

$\text{Eff.Address} := \text{База} + \text{Индекс} * \text{Масштаб} + \text{Смещение}$

База, Индекс: Регистр общего назначения.

Масштаб: Масштабный коэффициент 1, 2, 4 или 8.

Смещение: 8-битное или 32-битное смещение которое может быть добавлено к адресу.

Возможна любая комбинация этих трех элементов. Это позволяет использовать следующие режимы адресации:

Режим	Пример	
Непосредственный	Значение	87654321h
Регистр	Reg	EAX
Косвенный регистр	[Reg+disp]	[EAX], [EAX+1234h]
Масштабирование регистра	[Reg+Ind*scale]	[EAX+EBX*2]
	[Reg+Ind*scale+disp]	[EAX+EBX*4+8]
	[Ind*scale+disp]	[EBX*1+12345678h]
	[Ind*scale]	[EBX*8]
Абсолютный	[disp]	[12345678h]

Если сегментный регистр не указан, по умолчанию берется DS. Все остальные сегментные регистры должны быть указаны в начале. Например:

DS:[EAX+4] то же самое, что [EAX+4]

FS:[EBX+ECX\*2]

NS32000 предлагает супернабор режимов адресации [NS83]:

Режим		Пример
Непосредственный Регистр	Значение Reg	7654321h R0
Косвенный регистр	disp(Reg)	0(R0),1234h(R0)
Память относительная	disp1(disp2(Reg))	8(-4(SP)),8(-4(FP))
Абсолютный	@disp@12345678h	
Внешний	EXT(disp)+disp2	
Верхняя часть стека	TOS	
Объем памяти	*+disp, disp(FP)	*+1234h, 1234h(FP)
Масштаб	basemode[Rn:i]	0(R0) [R1:B]@12345678h[R1:D],R0[R1:Q]

Смещение может быть 7-, 14- или 30-bit знаковое значение.

### Выделенные регистры

В процессоре 80386 двойное перенаправление и возможность перемещения из памяти в память отсутствуют. Кроме того, 80386 нуждается в *выделенных регистрах* для некоторых операций. Хотя набор регистров называется общим, в действительности все еще существуют инструкции, использующие только специальные регистры (см. ниже).

Функция	Мнемоника	Выделенный регистр
Порт IO	IN AL,DX OUT DX,AL	EAX,EDX EAX,EDX
Преобразование байта в слово	CBW	EAX
Преобразование слова в двойное число	CWDE	EAX
Сравнить	CMPSB, CMPSW, CMPSD	ESI,EDI
Беззнаковый делитель	IDIV EAX, reg/mem (EDX:EAX - делитель, делимое хранится в EAX, остаток в EDX)	EAX,EDX
Перемещение строки	MOVSB, MOVSW, MOVSD	ESI,EDI
Повторить	REP	ECX
Поворот, сдвиг	ROL, ROR, SHL, SHR, SAL, SAR	ECX
Сдвиг двойной	SHLD, SHRD	ECX
Хранить строку	STOSB, STOSW, STOSD	ESI,DI

### Управление регистрами

Философия регистров Intel создала множество проблем в управлении регистрами. Компилятор может очень легко попасть в ситуацию необходимости в выделенном регистре, который уже используется. Педотвратить такую нехватку регистров – это сделать доступными часто используемые выделенные регистры позже остальных. В компиляторе выбран следующий порядок предпочтений: EAX, EBX, EDX, EDI, ESI, ECX.

Хотя это работает в большинстве случаев, это не исключает конфликтов во всех случаях. В случае конфликта, т.е. если требуется уже занятый регистр, компилятор временно освобождает нужный регистр, копируя его содержимое в свободный регистр или (если все регистры заняты) в стек. Эта техника используется для инструкций DIV и MOD. Обратите внимание,

что push и pop должны происходить в контексте, где мы можем убедиться, что значение, которое мы получим обратно в результате pop будет таким же, как и прежде. В оригинальном компиляторе регистры освобождаются в конце оператора или условного выражения. Это возможно потому, что NS32000 имеет больше регистров, перемещение из памяти в память и двойное перенаправление. Нет необходимости освобождать регистры раньше. Однако компилятор Intel должен освободить регистры как можно скорее, иначе у него закончатся регистры. Поскольку все вышеперечисленные возможности отсутствуют в Intel, необходимо найти дополнительные методы, такие как:

-Перемещение из памяти в память выполняется через перемещение из памяти в регистр и перемещение регистра в память. Очевидно, что временный регистр может быть снова освобожден.

-Если выражение состоит из нескольких коэффициентов и членов, результат каждой части помещается в свой регистр. Части, которые больше не используются, возвращаются в свои регистры.

-регистры индексов могут быть освобождены сразу после использования.

-В случае косвенных вычислений один и тот же регистр может быть использован многократно. Например:

```
TYPE Ptr = POINTER TO P;
TYPE P = RECORD next: Ptr;
val: INTEGER END;
VAR a: Ptr;
BEGIN a.next.next.next.next.next.val:=10 END
```

переводится в следующий код NS32000

```
MOVD 0(a(SB)),R7          a.next
MOVD 0(R7),R6             a.next.next
MOVD 0(R6),R5             a.next.next.next
MOVD 0(R5),R4             a.next.next.next.next
MOVXBW 10,4(R4)           a.next.next.next.next.x := 10
```

или в код Intel

```
MOV EAX, DWord[a]         a
MOV EAX, DWord[EAX]        a.next
MOV EBX, DWord[EAX]        a.next.next
MOV EAX, DWord[EBX]        a.next.next.next
MOV EBX, DWord[EAX]        a.next.next.next.next
MOV Word 4[EBX],10         a.next.next.next.next.x:=10
```

В первом примере при каждом перенаправлении занимает новый регистр. Второй пример повторно использует освобожденные регистры и, таким образом, может компилировать более сложные выражения. Загрузка первой части (a.next) может быть отложена, поскольку компилятор генерирует косвенный элемент[Wir92]. В коде NS32000 загрузка такого элемента приводит к одной инструкции. В коде Intel необходимо две инструкции (Memory-to-Register и Register-indirect).

Блок плавающей запятой Intel (80387 или включен в 80486 DX) имеет стековую архитектуру по сравнению с регистровой (F0-F8) NS32000. Практически все инструкции работают с элементом, находящимся сверху стека.

### **Перемещение**

В оригинальном компиляторе константы и глобальные переменные, все ссылаются на статическую базу (SB). В Intel такого регистра нет. Все места, где используются глобальные переменные и константы, должны быть перемещены с абсолютным адресом во время загрузки. Для того чтобы найти перемещения, при компиляции создается цепь исправлений. Также для внешних вызовов процедур, во время компиляции строится цепь исправлений которая должна быть перемещена во время загрузки. Это приводит к созданию отдельного раздела ссылок в каждом объектном файле.

### **NIL-проверки**

Поскольку в памяти нет областей "сохранения", защищенных от чтения и записи. Отдельные NIL-проверки должны быть дополнительно обработаны программным обеспечением. Каждый раз при разыменовании указателя компилятор генерирует код для проверки того, является ли адрес NIL (=0) или нет. Когда программа не имеет разыменований или когда программа проверяется, так что NIL-пересылки не могут возникнуть (при этом предполагается, что все остальные программы проверены и работают правильно) эта опция может быть отключена. Обратите внимание, что Intel предлагает четыре так называемых отладочных регистра, каждый из которых может контролировать один адрес. Когда к одному из этих адресов происходит обращение, процессор генерирует ловушку. Эти четыре отладочных регистра могут быть использованы для этого но их недостаточно, так как только 16 байт могут быть проверены таким образом. Доступ к NIL обычно выглядит следующим образом:

```
MOV Reg, offset(NIL)
```

где смещение может быть больше 16. Рекомендуется компилировать программу сохранения без NIL-проверки с точки зрения производительности и размера кода. Программа, интенсивно работающая с указателями, может получить на 20% меньший размер i, если компилировать ее без NIL-чеков. Другими доступными опциями компилятора являются:

```
/n NIL-проверки отключены  
/x Проверка индексов отключена  
/t Проверка типов отключена  
/o Проверка переполнения отключена  
/s Разрешить новый файл символов  
/i Информация о неиспользуемых и неинициализированных переменных
```

## **4.2 Специфические особенности**

В компилятор Intel были внесены некоторые специфические для данного процессора дополнения. Все они реализованы в модуле SYSTEM.

```
SYSTEM.GETREG(reg, value)  
SYSTEM.PUTREG(reg, value)  
reg:= 0=EAX, 1=ECX, 2=EDX, 3=EBX, 4=ESP, 5=EBP, 6=ESI, 7=EDI
```

value: целое число или символ.

Эти процедуры получают значение из указанного регистра или помещают его в указанный регистр. Размер используемого регистра (1 байт, 2 байта или 4 байта) определяется размером значения параметра.

```
SYSTEM.PORTIN(port, value)
SYSTEM.PORTOUT(port, value)
    port:      Адрес порта
    value:     Целое число или символ.
```

Эти процедуры считывают значение из указанного порта или записывают значение в указанный порт. Размер используемого регистра (1 байт, 2 байта или 4 байта) определяется размером значения параметра.

```
SYSTEM.CLI()
Сбрасывает флаг прерывания и отключает входящие прерывания.
```

```
SYSTEM.STI()
Устанавливает флаг прерывания и разрешает входящие прерывания.
```

## FOR-оператор

Также в состав языка входит FOR-оператор

```
FORStatement=
  FOR i := beg TO end BY step Последовательность операторов DO END.
```

Beg, end и step оцениваются в начале. Они рассматриваются как константы. Переменная step добавляется к i после каждого выполнения последовательности операторов. Если шаг не объявлен, он принимается равным 1. FOR-выражение эквивалентно одной из следующих последовательностей:

```
(*Step>0*)
i:=beg;
WHILE i<= end DO
последовательность операторов;
INC(i,step)
END
```

```
(*Step<0*)
i:=beg;
WHILE i<= end DO
последовательность операторов;
DEC(i,step)
END
```

## 4.3 Формат Объектного файла

Компилятор переводит исходные файлы в так называемые объектные файлы. Они содержат помимо закодированной программы информацию об импортируемых модулях, экспортируемых процедурах, типах и переменных, а также цепь отладки и структуры типов. Используется следующая нотация:

Размер указывается после двоеточия. Например: 4 означает, что размер отчета составляет 4 байта.

Объектный файл =

Заголовки Записи Команды Указатели Проц-переменные Импорты  
Ссылки Цепь исправлений Типы Ссылки.

```

Заголовок =
    objmark refpos:4 nofentries:2 nofcmds:2 nofpointers:2
    nofimports:2 noflinks:2 noftypdscs:2 datasize:4
    constantsize:2 codesize:4 key:4 name.

    Entries=entrytag{entries:4}.
    Commands=commandtag{name entry:4}.
    Pointers=pointertag{pointer:4}{typedesc:4}.
    Imports=importtag{key:4 name}.
    links=Linktag{links:4}.

Типы =
    typetag{tdsize:2 tdadr:4 recsize:4 nofptrs:2 noftbp:2{ptrs:4}
    {typeboundprocs:4}}.

Ссылки =
    reftag {objmark adr:2 name{mode:1 from:1 adr:4 name}}.
    name={byte}0X.
    constants={byte}.
    code = {byte}.

    objmark=0F8X
    entrytag=81X.
    commandtag=82X.
    pointertag=83X.
    procvartag=84X.

    importing=85X.
    linktag=86X.
    fixuptag=87X.
    codetag=88X.
    typetag=89X.
    reftag=8AX.

```

## 4.4 Размер программы

В следующих таблицах показаны различия в размерах оригинального NS-компилятора и нового компилятора Intel. Компилятор Intel был расширен, чтобы быть больше по размеру из-за неортогонального набора инструкций процессора и управления регистрами. DOS означает DOS Oberon Компилятор, ОС означает NS Oberon Компилятор.

### Размеры DOS Oberon

Модуль	Функция	Размер
DOCS.Mod	Сканнер	331 строк
DOCT.Mod	Обработчик таблиц	608 строк
DOCC.Mod	Генератор кода	624 строк
DOCE.Mod	Выражения	1345 строк
DOCH.Mod	Высокоуровневый генер-р кода	672 строк
Compiler.Mod	Парсер	1024 строк
Всего		4594 строк



## Размеры NS Oberon

Модуль	Функция	Размер
OCS.Mod	Сканнер	309 строк
OCT.Mod	Обработчик таблиц	536 строк
OCC.Mod	Генератор кода	561 строк
OCE.Mod	Выражения	916 строк
OCH.Mod	Высокоуровневый генер-р кода	506 строк
Compiler.Mod	Парсер	920 строк
Всего		3748 строк

## Глава 5. Загрузчик и ядро

### 5.1 Метазагрузчик

Метазагрузчик является загрузочным и интерфейсным модулем. Он подготавливает необходимую структуру памяти, загружает фактический загрузчик модулей и передает ему управление для загрузки Modules.Obj. Кроме того, он берет на себя функции DOS-расширителя, вызовы DOS и BIOS передаются в часть расширителя метазагрузчика для дальнейшей обработки. Порядок различных действий, выполняемых метазагрузчиком, показан ниже:

#### 1. Получить переменные окружения:

-Максимальный размер кучи может быть указан с помощью

SET OBERONMEM=xxxx(KB) .

Если размер не указан, вся доступная память будет занята.

-Путь к каталогу Oberon может быть задан с помощью:

SET OBERON = drive:\path.

Указанный каталог принимается за основной для системы Oberon (см. главу 9: Файловая система). По умолчанию это текущий каталог.

Обе переменные окружения могут быть установлены через AUTOEXEC.BAT или через командную оболочку.

#### 2. Установить правильные параметры дискеты.

#### 3. Построить таблицу дескрипторов прерываний защищенного режима (IDT).

С каждым прерыванием в PM ассоциируется вектор прерывания selector.offset.

4. Проверить наличие драйвера HIMEM.SYS и выделить непрерывный блок памяти нужного размера. Эта память впоследствии будет управляться ядром. HIMEM.SYS – это драйвер, который реализует расширенную спецификацию памяти (XMS). Он предоставляет функции для выделения, изменения и освобождения блоков расширенной памяти. См. также [Dun90].

5. Прочитать заголовок загрузчика объектного файла. Информация заголовка необходима для выделения блока памяти для кода и данных, перемещения глобальных переменных и адреса процедур.

6. Получить память для загрузчика модулей Oberon, аппаратной обработки прерываний и для стека. Эта память полностью содержится в 32-битном сегменте и доступна из 3Р (см. главу 3: Модель памяти).

Скопировать код для аппаратной обработки прерываний в 32-битный сегмент. Некоторые аппаратные прерывания (деление на ноль, переполнение, неверный опкод, нарушение сегмента) вызывают ловушку Oberon и не передаются в РР. В таких случаях обработчик прерывания помещает в стек номер ловушки, в зависимости от того, какая ловушка произошла и вызывает обработчик ловушек Oberon.

7. Построить таблицу глобальных дескрипторов (GDT).

entry 0-->Null. Доступ к памяти с помощью этого вызывает ловушку

entry 1 --->Code16 дескриптор

Установить ограничение в 64 КБ.

Установить базовый адрес. Находится в текущем сегменте кода (CS).

Установить соответствующие атрибуты для этого сегмента.

entry2--->Data16 дескриптор

то же, что и выше, но этот сегмент должен быть доступен для чтения и записи.

entry3--->Code32 дескриптор

Установить ограничение до указанного размера (OBERONMEM) или по умолчанию.

Установить базовый адрес. Этот адрес равен 0.

Установить соответствующие атрибуты для этого сегмента.

entry4--->Data32 дескриптор

то же, что и выше, но этот сегмент должен быть доступным для чтения

и записи.

8. Прочитать Loader.Obj (загрузчик модулей Oberon) и передать важные адреса из мета-загрузчика в этот загрузчик. Просмотрите все ссылки и выполните исправления.

9. Загрузить таблицу глобальных дескрипторов (GDT) и таблицу дескрипторов прерываний (IDT), переключиться на 3Р и загрузить указатель стека. Перейти к началу модуля Oberon загрузчика.

В этот момент модули Oberon берут управление на себя. Метазагрузчик используется только для системных вызовов и обработки прерываний. См. главу 2: Среда Intel механизм переключения из защищенного режима в реальный режим и наоборот.

## 5.2 Загрузчик модулей

Загрузчик модулей отвечает за загрузку и освобождение модулей Oberon. Он должен выделить память под дескриптор модуля, часть код/данные модуля с дополнительной информацией и для дескрипторов типа. Он также обрабатывает импорт и перемещает глобальные переменные и внешние вызовы процедур. Когда сам загрузчик загружается и запускается, он загружается с помощью модуля Modules.Obj (интерфейс загрузчика), который затем загружает модуль

Oberon.Obj (содержит главный цикл системы Oberon). До загрузки ядра памятью управляет DOS. Это означает, что вся память, необходимая для загрузки Modules.Obj и Kernel.Obj (Kernel.Obj импортируется Modules.Obj), выделяется в первом мегабайте памяти. Для этого была реализована специальная процедура:

```
PROCEDURE DOSNew(VAR adr:LONGINT; size:LONGINT);
```

```
BEGIN
```

Получить память из DOS. Размер должен быть выровнен по границе 32 байт.

Вычислить адрес.

Создать дескриптор, как в процедуре Oberon NEW.

Инициализировать память значением 0.

```
END DOSNew;
```

### **Распределение памяти**

Блоки памяти, выделенные с помощью этой процедуры имеют ту же структуру, что и блоки, выделенные с помощью управления памятью Oberon. Номер прерывания 'MyInt' – это специальный фиктивный номер, не используемый другими вызовами. В дополнение к обычному выделению памяти DOS блок регистрируется в таблице и освобождается при выходе пользователя из системы Oberon. В конце сеанса работы Oberon вся нижняя память, а также расширенная память освобождаются. Как только ядро Oberon загружается и инициализируется, управление памятью выполняется ядром и вышеупомянутая процедура больше не используется, за исключением модулей, которые должны быть загружены в низкую память. Это могут быть модули драйверов, модули, работающие с буферными адресами, например, драйверы CD-ROM. Специальный тег в объектного файла указывает, может ли модуль быть загружен нормально в расширенную память или он должен быть загружен в низкую память. За исключением этого, используется только расширенная память. Мы снова замечаем, что сборщик мусора не заботится о низкой памяти. Любой модуль в низкой памяти может быть освобожден, как и все остальные модули, но сборщик мусора не собирает низкую память. Однако поскольку модули низкой памяти все равно являются постоянными (ядро, загрузчик и драйверы), память не нужно собирать.

### **Имена файлов**

Модуль Files предоставляет таблицу перевода в 32-символьные имена файлов. До того, как эта таблица инициализируется, все имена модулей должны соответствовать 8+3-символьным именам DOS. Этот касается модулей Kernel.Obj, DOS.Obj, Files.Obj, FileDir.Obj и Modules.Obj. Все модули, которые загружаются после этого, больше не связаны с ограничением 8+3. Еще раз обратите внимание, что только Kernel.Obj и Module.Obj в нижней памяти, а все остальные модули находятся в расширенной памяти. На следующем рисунке показано, как подробно организованы дескрипторы и часть код/данные. (Рис. 1).

### **Перенос**

Когда все дескрипторы модулей и дескрипторы типов распределены и импортирование завершено, загрузчик начинает перемещение глобальных переменных и внешних процедур.

Помните, что такие элементы объединяются в цепочки исправлений. Запись исправления состоит из адресной части (16 бит) и смещения в кодовой части

модуля. В случае переменной, эффективный адрес статической базы (SB) добавляется к этому смещению и дает абсолютный адрес переменной.

Адресная часть связывает со следующим исправлением в этом модуле. Этот механизм дает ограничение на максимальный размер кода в 64 КБ (16 Бит без знака) на модуль. В случае внешней процедуры адрес может быть найден через список импорта (модуль) и таблицу вхождения (процедура). Техника исправления такая же, как и для переменных.

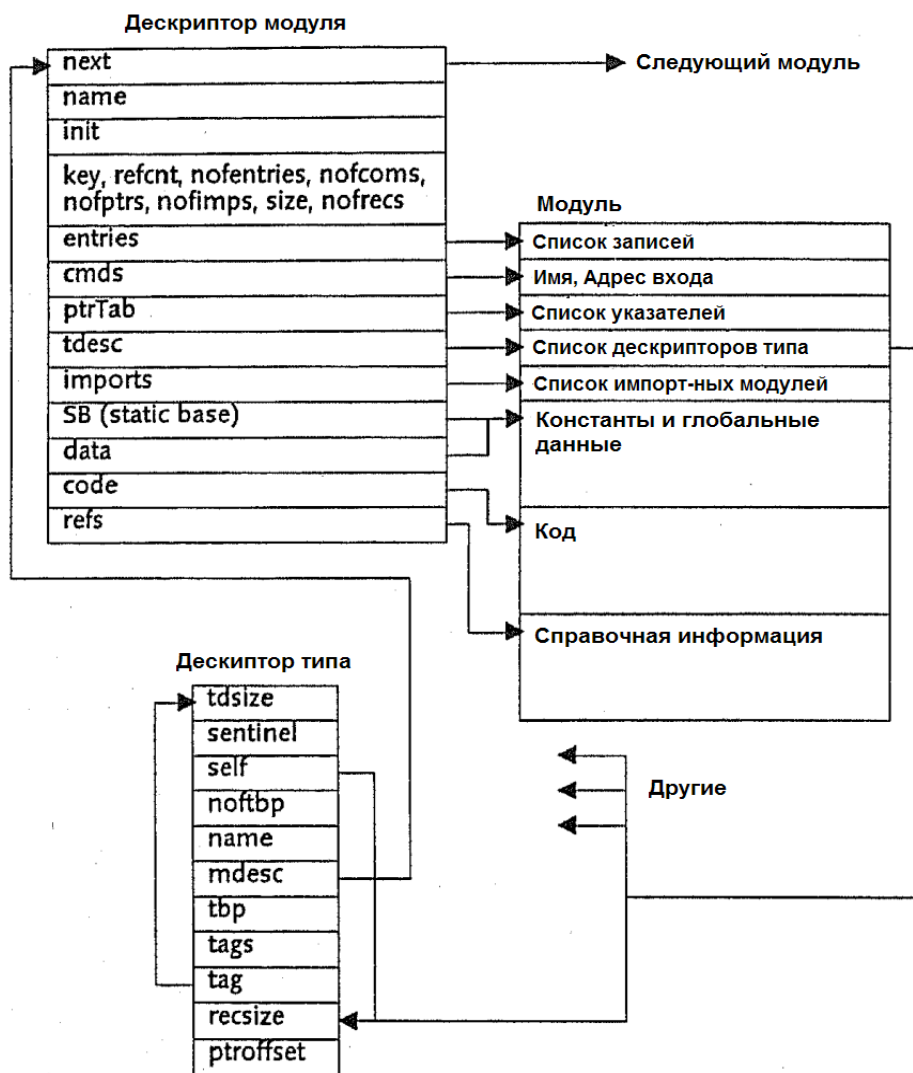


Рис 1: Дескрипторы модулей

Загрузчик является первым модулем, загружаемым в сегмент Oberon (см. главу 3: Модель памяти). Некоторые данные метазагрузчика необходимы в среде Oberon. В качестве побочного эффекта, эти данные устанавливаются метазагрузчиком при загрузке загрузчика модулей (упомянутого выше). Аналогичные отношения существуют между загрузчиком и ядром. Все загруженные модули вставляются в цепочку модулей. Это позволяет получить доступ к коду и данным каждого модуля, так как вся информация о модуле хранится в дескрипторе модуля (рис. 1). Ядро является первым модулем в этой цепочке. Загрузчик, не включенный в эту цепочку, не может быть доступным для системы исполнения Oberon поскольку он полностью изолирован. Между модулем Oberon и загрузчиком нет никакой связи.

Поэтому, когда загружается ядро, некоторые важные данные копируются в ядро загрузчиком. Эти данные включают, например, адрес DOShandler, область передачи данных между Oberon и DOS, адрес кучи, адрес видеопамати и адрес различных буферов.

(Процедуру инициализации см. ниже)

```
PROCEDURE initKernel(m:Module);
VAR data: LONGINT;
handler:Proc;
Load: Proc1;
getproc: Proc3;
free: Proc4;
BEGIN
data:=m.SB;
m.refcnt:=1; (*Ядро не может быть загружено*)
handler:=DOSHandler;
load:=Load;
getproc:=GetProc;
free:=Free;
SYSTEM.PUT(data-4, SYSTEM.VAL(LONGINT, handler));(*DOSHandler*)
SYSTEM.PUT(data-8, SYSTEM.ADR(Reg));(*Paramfterblock*)
SYSTEM.PUT(data-16, SYSTEM.ADR(Transfer[0]));(*Data transfer area*)
SYSTEM.PUT(data-20, heapAdr);
SYSTEM.PUT(data-24, heapSize);
SYSTEM.PUT(data-28, StackOrg);(*Stackorigin*)
SYSTEM.PUT(data-32, IntTransfer);(*Address of int transfer buffer*)
SYSTEM.PUT(data-36, SYSTEM.ADR(modules));(*Module list anchor*)
SYSTEM.PUT(data-40, SYSTEM.VAL(LONGINT, getProc));
SYSTEM.PUT(data-44, SYSTEM.VAL(LONGINT, load));
SYSTEM.PUT(data-48, SYSTEM.VAL(LONGINT, free));(*loader proc*)
SYSTEM.PUT(data-52, SYSTEM.ADR(main));
SYSTEM.PUT(data-56, SYSTEM.ADR(File1));
SYSTEM.PUT(data-60, SYSTEM.ADR(File2));
SYSTEM.PUT(data-64, Display);
SYSTEM.PUT(data-68, BufBeg);
TableRoot:=data-72;
body:=SYSTEM.VAL(Proc, m.entries[0]);body; m.init:=TRUE;
SYSTEM.GET(data-76, Routine[0]); (*NEW*)
SYSTEM.GET(data-80, Routine[1]); (*SYSTEM.NEW*)
SysNew:=SYSTEM.VAL(sysnew, Routine[1]);
```

```

New:=SYSTEM.VAL(new,Routine[0]);
SYSTEM.PUT (data-102, DS);
SYSTEM.PUT (data-104, ES);
KernelLoaded:=TRUE;
END InitKernel;

```

## 5.3 Ядро

Изначально ядро было реализовано для станции DEC. Оно предоставляет процедуры для выделения и сбора памяти и некоторые утилиты обработки ловушек. Только раздел памяти свыше 1 мегабайта обрабатывается ядром. Раздел ниже используется только загрузчиком и управляется исключительно DOS. Причина этого заключается в том, что проще управлять одним большим блоком памяти вместо двух. В противном случае диапазон от 0A0000H до 0FFFFFFH (видео и часть BIOS) находился бы прямо между двумя блоками памяти. Как упоминалось выше, важные данные копируются из метазагрузчика или загрузчика в ядро. Эти данные перечислены ниже.

```

TYPE
    REGISTER=RECORD
        AX, CX, DX, BX:INTEGER;
        SP, BP, SI, DI:INTEGER;(*SP,BP:read only*)
        Flags:INTEGER;
        CS, SS:LONGINT;
        DS, ES:LONGINT
    END;
Buffer=ARRAY 4096 OF CHAR;

Buf=POINTER TO Buffer;
Register=POINTER TO REGISTER;

(*порядок объявления следующих переменных известен загрузчику!*)
(*Загрузчик инициализирует все переменные кроме Reg и Transfer*)

VAR
    INT:PROCEDURE;
    Reg: Register;
    Transfer: Buf;
    heapAdr: LONGINT;
    heapSize: LONGINT;
    StackOrg: LONGINT;
    modules: LONGINT;
    Load: PROCEDURE(mod: ARRAY OF CHAR; VAR m: Module;
        VAR res: INTEGER);
    GetProc: PROCEDURE(cmd: ARRAY OF CHAR; VAR m: Module;
        VAR command:Proc; VAR res:INTEGER);
    Free: PROCEDURE(mod: ARRAY OF CHAR; all: BOOLEAN;
        VAR res:INTEGER);

    Main: LONGINT;
    File1: LONGINT;
    File2: LONGINT;
    Display: LONGINT;
    dx: LONGINT;
    TableRoot: LONGINT;

```

```
new:PROCEDURE (tag:Tag):LONGINT;

(*доступен загрузчику для получения адреса для NEW*)
systemNew: PROCEDURE(size:LONGINT):ADDRESS;
(*доступ загрузчика для получения адреса SYSTEM.NEW*)
```

### Процедура прерывания

Ядро также предоставляет механизм для вызовов DOS и вызовов BIOS (см. главу 2: Среда Intel) и для установки процедур прерывания. В следующем примере показано, как устанавливается такая процедура:

```
PROCEDURE +Int;                (*процедура прерывания*)
VAR...
BEGIN
    SYSTEM,CLI();              (*отключить прерывания*)
                                (*код*)
    SYSTEM,STI();              (*включить прерывания*)
    SYSTEM,PORTOUT(20H,20H);    (*окончание прерываний*)
END Int;

POCEDURE Install*(port:INTEGER);
BEGIN
    IF port=COM1 THEN Kernel.InstallIP(Int,4)
    (* Процедура инсталляции Int. Второй параметр в installIP обозначает
    к какому аппаратному прерыванию процедура должна быть подключена. В
    данном случае в прерывание COM1 *)
    ELSIF port=COM2 THEN Kernel.InstallIP(Int,3)
    (* Подключиться к прерыванию COM2 *)
    (* ELSIF ... *)
    ELSE HALT(88) END;
END Start;
```

## Глава 6. Модуль DOS

Модуль DOS является интерфейсом между MS-Disk Operating System и Oberon. В частности, он предоставляет интерфейс всем вызовам DOS и BIOS, которые необходимы системе Oberon. Хотя такие вызовы могут быть сделаны любым модулем на месте, они сконцентрированы в одном месте. Это дает четкий интерфейс к низкоуровневым процедурам DOS.

### Передача аргументов

Параметры системного вызова обычно передаются операционной системе в регистрах. В некоторых случаях, например, когда строка является вложенной, используется буфер памяти. В таких случаях регистр параметров содержит адрес буфера данных. Фактический вызов реализуется как программное прерывание. В качестве примера можно привести следующие команды ассемблера:

```
(*Установка системных часов*)
MOV AH, 2H      (*Установить номер функции*)
INT 1AH         (*Генерировать прерывание 1AH*)
```

Результаты также передаются в регистрах. Более того, некоторые вызовы

поднимают бит переноса, если операция не была успешной, а другие передают информацию об успешном выполнении в регистре, обычно в EAX. В модуле Kernel существует переменная global переменная Reg, предоставляющая адрес блока параметров и буфер под названием Transfer поддерживающий передачу данных между Oberon и DOS. Системные вызовы должны выполняться в реальном режиме и поэтому не могут быть выполнены через буфер передачи. Вместо прямого вызова прерывания, вызывается процедура которая переключается обратно в реальный режим, копирует параметры из Reg в регистры, вызывает соответствующее прерывание, записывает результаты обратно в Reg и снова переключается в защищенный режим. Следующая процедура показывает это:

```
PROCEDURE SystemCall;
BEGIN
Сохранить параметры системного вызова в регистровой переменной;
Заполнить буфер передачи, если необходимо;
Переключить в реальный режим;
Скопировать содержимое регистровой переменной в реальные регистры;
(*Буфер передачи уже находится в реальном режиме, здесь ничего делать не
нужно*)
Вызвать заданное прерывание;
Скопировать результаты обратно в регистровую переменную;
Переход в защищенный режим;
END SystemCall;
```

Здесь показано определение параметра block и transfer buffer:

```
TYPE
    REGISTER=RECORD
    AX, CX, DX, BX:INTEGER;
    SP, BP, SI, DI:INTEGER; (*SP,BP:Только для чтения*)
    Flags:INTEGER;          (*Только для чтения*)
    CS, SS: LONGINT;        (*Только для чтения*)
    DS, ES: LONGINT;
END;
```

```
(* Результаты в поле Flags
Bit0: Carry CF      Bit4: Auxiliary AF
Bit1: unused        Bit5: Overflow OF
Bit2: Parity PF     Bit6: Zero ZF
Bit3: unused        Bit7: SignSF      *)
```

```
Buffer=ARRAY 4096 OF CHAR;
Buf=POINTER TO Buffer;
Register=POINTER TO REGISTER;
```

```
VAR
    INT: PROCEDURE;
    Reg: Register;
    Transfer: Buf;
```

### **Системные вызовы**

Вызов операционной системы осуществляется следующим образом:

```
PROCEDURE WaitForKeyboard*; (*Вызов без буфера*)
BEGIN
```



```

Kernel.Reg.AX:=700H;
Kernel.INT(Kernel.Reg, 21H);
END WaitForKeyboard;

PROCEDURE DeleteFile*(name: ARRAY OF CHAR); (*Вызов с буфером*)
VAR i: INTEGER;
BEGIN i:=0;
Kernel.Reg.AX:=4100H;
Kernel.Reg.DS:=Kernel.DS; (* DS:DX начало буфера передачи *)
Kernel.Reg.DX:=Kernel.DX;
REPEAT Kernel.Transfer[i]:=name[i];INC(i) UNTIL name[i]=0X;
Kernel.Transfer[i]:=0X;
Kernel.INT(Kernel.Reg, 21H)
Done:=~SYSTEM.BIT(SYSTEM.ADR(Kernel.Reg.Flags), 0);
END DeleteFile;

```

### **Системные параметры**

Модуль DOS также содержит некоторые системные параметры.

```

VAR
Done,                                (*Последний вызов был успешным*)
CoAvail: BOOLEAN;                    (*сопроцессор доступен*)
DispHeight, DispWidth: INTEGER;    (*Разрешение экрана*)
tag:ARRAY 3 OF CHAR;                (*Тег файлов Oberon*)

```

### **Доступные процедуры обслуживания:**

Вывод на экран и ожидание:

```

PROCEDURE WriteChar(ch:CHAR);
PROCEDURE WriteString(s:ARRAY OF CHAR);
PROCEDURE WriteInt(i:LONGINT);
PROCEDURE WriteLn;
PROCEDURE WriteHex(i:SYSTEM.BYTE);
PROCEDURE Wait;

```

Последовательный порт:

```

PROCEDURE InitSerialPort(port,wklength, stopbits, parity, baud:INTEGER);
PROCEDURE RecCharSer(port:INTEGER; VAR ch: CHAR);
PROCEDURE SendCharSer(port:INTEGER; ch: CHAR);
PROCEDURE GetStatusSer(port:INTEGER; VAR status: INTEGER);

```

Параллельный порт:

```

PROCEDURE InitParallelPort(port:INTEGER);
PROCEDURE SetClok(time, date:LONGINT);
PROCEDURE GetTicks():LONGINT;

```

Клавиатура:

```

PROCEDURE KBAvail():BOOLEAN;
PROCEDURE GetChar(VAR ch:CHAR; VAR ext: BOOLEAN);

```

PROCEDURE ControlKeys (VAR Keys:INTEGER);

Экран:

PROCEDURE InitDisplay(mode:INTEGER);

PROCEDURE GetColor(col:INTEGER; VAR red, green, blue:INTEGER);

(\*реализована в драйвере\*)

PROCEDURE SetColor(col, red, green, blue:INTEGER); (\*реализована в

драйвере\*)

Мышь :

PROCEDURE InitMouse;

PROCEDURE SetMouse(x,y: INTEGER);

PROCEDURE GetMouseInfo (VAR x,y, buttons: INTEGER);

Файлы:

PROCEDURE Open (VAR name:ARRAY OF CHAR; new: BOOLEAN; VAR handle: LONGINT;  
accessmode: INTEGER);

PROCEDURE Close (handle:LONGINT);

PROCEDURE Delete (name: ARRAY OF CHAR);

PROCEDURE Rename (old,new: ARRAY OF CHAR);

PROCEDURE Length (VAR len: LONGINT; handle: LONGINT);

PROCEDURE DirOpt (dos:ARRAY OF CHAR; VAR time, date, size: LONGINT);

PROCEDURE GetDateTime (VAR date, time: LONGINT; handle: LONGINT);

PROCEDURE SetPos (pos, handle:LONGINT);

PROCEDURE DoubleHandle (handle:LONGINT):LONGINT;

PROCEDURE Write (size, adr, handle:LONGINT);

PROCEDURE Read (size, adr, handle:LONGINT; VAR read:LONGINT);

PROCEDURE GetID (VAR ID:ARRAY OF CHAR; handle:LONGINT);

PROCEDURE SetID (ID:ARRAY OF CHAR; handle:LONGINT);

PROCEDURE GetTag (VAR tag:ARRAY OF CHAR; handle:LONGINT);

PROCEDURE SetFileCount (no:INTEGER);

(\*Для дальнейшего использования\*)

PROCEDURE LockFileAccess (handle, Offset, Length:LONGINT;

shareable:BOOLEAN):BOOLEAN;

PROCEDURE UnlockFileAccess (handle, Offset, Length:LONGINT;

shareable:BOOLEAN):BOOLEAN;

PROCEDURE IsHandleRemote (handle:LONGINT):BOOLEAN;

PROCEDURE BufferingRecommended (AccessMode:INTEGER; FileRemote:BOOLEAN;

VAR BufferedRead, BufferedWrite:BOOLEAN);

Каталог файлов:

PROCEDURE GetFirstFile (VAR name: ARRAY OF CHAR);

PROCEDURE GetNextFile (VAR name: ARRAY OF CHAR; VAR end: BOOLEAN);

PROCEDURE MakeDir (name: ARRAY OF CHAR);

PROCEDURE RemoveDir (name: ARRAY OF CHAR);

PROCEDURE ChangeDir (name: ARRAY OF CHAR);

PROCEDURE GetDir (drv: ARRAY OF CHAR; VAR name: ARRAY OF CHAR);

PROCEDURE GetDrive (VAR drv: ARRAY OF CHAR);

Разное:

PROCEDURE AllocTermProc (proc:Proc);

(\*Процедуры, которые вызываются перед завершением работы системы,  
например, закрытие всех временных файлов\*)

PROCEDURE Quit; (\*Выход из системы\*)

## Глава 7. Дисплей

Драйвер дисплея был наиболее критичным по времени частью всей реализации. Он должен быть быстрым, чтобы обеспечить хорошую отзывчивость системы и должен быть способен работать с разумным разрешением. Хотя существует большой выбор различных интерфейсных карт и стандартов, мы должны были ограничить нашу реализацию только некоторыми из них. Некоторые из этих стандартов – SVGA, VGA, EGA, MCGA, MDA, Hercules. Для получения дополнительной информации об интерфейсных картах и стандартах см. [Fer91]. Оригинальная система Oberon работает в монохромном графическом режиме с разрешением 1024 \* 800 пикселей. Целью было разрешение, которое оптимально соответствует нашим машинам Ceres. Кроме того, мы хотели чтобы стандарт был как можно более распространенным.

### Монохромный дисплей

На первом этапе был написан драйвер дисплея для карты VGA с разрешением всего лишь 640\*480 пикселей и монохромного дисплея. По соображениям эффективности, этот модуль закодирован на ассемблере, а BIOS обойден, т. е. доступ к видео-RAM осуществляется напрямую.

### Цветной дисплей

Следующим шагом был цветной дисплей. Выбранная карта предлагает 16 цветов с разрешением VGA, такое же количество цветов, как и у наших машин Ceres машины. Эта реализация до сих пор поддерживается вместе со всеми остальными частями порта [Off92].

### Карта ET4000

Позже мы разработали драйвер для карты ET4000 SVGA с разрешением 1024\*768 и 256 цветами, потому что ET4000 хорошо известна и быстра. Мы также оценили Trident8900. Однако в процессе реализации, оказалось, что ET4000 с двумя банковскими регистрами быстрее, чем Trdent с одним банковским регистром (особенно для Copy Block). Точную информацию об обеих интерфейсных картах см. [Fer91].

Но все же дисплей был медленным по сравнению с нашей оригинальной машиной Ceres. Теперь у нас было почти такое же разрешение дисплея с еще большим количеством цветов, но нам нужна была более быстрая карта. Одной из возможностей было использовать карту-ускоритель. На самом деле, мы нашли идеальное решение в наборе микросхем S3. Он не очень дорогой и обладает высокой производительностью благодаря своему собственному графическому процессору.

В нашей машине для разработки используется версия S3C805 с локальной шиной, работающего с разрешением 1024\*768 пикселей и 256 цветами. Это примерно в 4-5 раз быстрее, чем цветной дисплей Ceres при разрешении 1024\*800 16 цветов. Чип S3 знает 4 операции:

- Провести линию
- Заполнение прямоугольника
- Передача блока BIT
- NOP, resp. set parameters

Возможно, к удивлению, этого как раз достаточно для наших нужд. А именно,

все процедуры отображения Oberon могут быть сопоставлены с одной из вышеупомянутых команд:

-CopyPattern -> Нарисовать линию (текстурированную)  
-CopyBlock -> Перенос битового блока  
-ReplConst -> Заполнить прямоугольник  
-ReplPattern -> Нарисовать линию (текстурированную)  
-FillPattern -> Нарисовать линию (текстурированную)  
-Dot -> Нарисовать линию

В дополнение к основным операциям отображения Oberon, S3-карта даже реализована операция Line для рисования общей линии.

### Обрезка

Также в S3-процессоре реализована аппаратная обрезка. Чип позволяет задать прямоугольную область, которая определяет текущие границы. Последующие операции набора по выбору рисуют внутри или за пределами области обрезки.

### Расширение шаблона

Работая сейчас с 256 цветами, мы используем 8 бит на пиксель. Однако все шрифты Oberon хранятся в виде черно-белого шаблона с 1 битом на пиксель. К счастью, чип S3 предлагает возможность задать фон и цвет переднего плана и соответствующее автоматическое расширение режим. 0 в шаблоне отображаются на цвет фона, 1 - на цвет переднего плана. Комбинация двух цветов задается в специальном регистре fg/bg mix. Подробнее см. ниже.

Регистр Vg/Fg Mix (биты 0-3)

	Описание	Используется для
0	инвертированный экран	
1	все биты=0	режим замены (Фон)
2	все биты=1	
3	без изменений	режим инвертирования и закрашивания (Фон)
4	инвертированный цвет	
5	экран XOR цвет	режим инверсии (Передний план)
6	инвертированный экран XOR цвет	
7	цвет	режим закрашивания и замены (Передний план)
8	инвертированный экран OR инвертированный цвет	
9	экран OR инвертированный цвет	
10	инвертированный экран OR цвет	
11	экран OR цвет	
12	экран AND цвет	
13	инвертированный экран AND цвет	
14	экран AND инвертированный цвет	
15	инвертированный экран AND инвертированный цвет	

### Передача параметров

Параметры передаются в регистры интерфейсной карты через порты ввода/вывода. Это делает программирование очень простым. Существует только один факт, о котором необходимо позаботиться. Карта управляет очередью FIFO для инструкций. От типа карты зависит, 8 или 16 стеков

доступны. Прежде чем записать какое-либо значение в регистр карты, должно быть гарантировано, что есть свободная запись в очереди для их хранения. Самым простым и рекомендуемым способом является предварительная проверка на предмет пустой очереди. Если свободных записей нет, программа должна подождать, пока очередь не станет пустой, иначе будет сгенерировано прерывание по переполнению. Подробную информацию см. в [S3 92]. Следующие последовательности кода показывают, насколько компактно могут быть реализованы операции при использовании карты S3:

```

PROCEDURE InitDisplay; (*переводит дисплей в режим S3*)
BEGIN
Kernel.Reg.AX:=4F02H;
Kernel.Reg.BX:=205H;  (*S3 1024x768x256 (VESA)*)
Kernel.INT(Kernel.Reg, 10H);
END InitDisplay;

PROCEDURE WaitFIFOempty; (*Ждет, пока стек FIFO не опустеет*)
BEGIN
(*$ Dinline.Assemble
  MOV DX, $9AE8
lab1 IN AX, DX
      AND AX, $200
      JNZ lab1
END*)
END WaitFIFOempty;

PROCEDURE Dot(col, x, y, mode: INTEGER);
BEGIN
y:=Height-y;
WaitFIFOempty;
IF mode = invert THEN
  SYSTEM.PORTOUT(FGmix, LONG925H)) (*Установка регистра Fg mix*)
ELSE
  SYSTEM.PORTOUT(FGmix, LONG(27H))
END;
SYSTEM.PORTOUT(FGcol, col);          (*Set Fg color*)
SYSTEM.PORTOUT(MFcont, SHORT(pixctrl1)); (*Function control*)
SYSTEM.PORTOUT(curX, x);              (*X coordinate*)
SYSTEM.PORTOUT(curY, y);              (*Y coordinate*)
SYSTEM.PORTOUT(cmdReg, 121BH);         (*X Command*)
SYSTEM.PORTOUT(shortStroke, LONG(10H)); (*X Draw dot*)
END Dot;

PROCEDURE CopyBlock*( SX, SY, W, H, DX, DY, mode: INTEGER);
VAR xpos, ypos: INTEGER;
BEGIN
IF (W<=0) OR (H<=0) THEN RETURN END;
xpos:=0; ypos:=0;
IF SY<DY THEN INC(SY,H-1);INC(DY, H-1); ypos:=128 END;
IF SX<DX THEN INC(SX,W-1);INC(DX, W-1)ELSE xpos:=32 END;
SY:=Height-SY;DY:=Height-DY;
DEC(W); DEC(H);
WaitFIFOempty;
IF mode = invert THEN SYSTEM.PORTOUT(FGmix, LONG(65H))
ELSE SYSTEM.PORTOUT(FGmix, LONG(67H)) END; (*Fig mix register*)
SYSTEM.PORTOUT(MFcont, SHORT(pixctrl1)); (*Function control*)
SYSTEM.PORTOUT(curX, SX);                (*Source X*)
SYSTEM.PORTOUT(curY, SY);                (*Source Y*)

```

```

SYSTEM.PORTOUT(diaStep, DX);          (*Destination X*)
SYSTEM.PORTOUT(axStep, DY);          (*Destination Y*)
SYSTEM.PORTOUT(majAxis, W);          (*Width*)
SYSTEM.PORTOUT(MFcont, H);          (*Height*)
SYSTEM.PORTOUT(cmdReg, SHORT(copycmd)+xpos+ypos);    (*Command*)
END CopyBlok;

```

## Глава 8. Арифметика с плавающей запятой

Не все процессоры Intel имеют блок плавающей запятой (FPU) на чипе. Для некоторых (80386SX, 80486SX) требуется специальная микросхема (80x87) содержащая такой FPU. Хотя базовая система Oberon не выполняет вычисления в формате REAL, желательно иметь в распоряжении операции с плавающей запятой. По этой причине был написан эмулятор для 80x87. Он основан на эмуляторе MIPS R2010 с плавающей запятой. Непортабельные части (процедуры кода с регистровыми соглашениями) были адаптированы для реализации в DOS. Когда необходимо выполнить инструкцию с плавающей запятой, но не установлен FPU, система отлавливает его. Идея заключается в том, чтобы подключить к вызову эмулятора FPU в качестве процедуры обработки прерывания. После обработки прерывания программа продолжает выполнение нормально. Обратите внимание, что эмуляция также включает стек FPU, который должен быть обновлен.

### Эмулируемые инструкции

Эмулятор знает только инструкции ADD, SUB, DIV и MULT, а также ABS и NEG. Все остальные инструкции (например, тригонометрические функции) должны быть приведены к этой базе. (См. также ниже: Математика модули).

### Декодирование адресов

Итак, эмулятор вызывается по ловушке 7 (без сопроцессора). Задача эмулятора плавающей точки теперь состоит в следующем: (1) декодировать адрес и (2) интерпретировать функцию. Интерпретация функции уже была реализована, но декодирование адреса пришлось переделать. Для этого для этого мы использовали один трюк. Предположим, что нужное РЕАЛЬНОЕ число находится по адресу месте вне[Reg1+Reg2\*4]. Тогда абсолютный адрес может быть вычислить с помощью LEA Reg, off[Reg1+Reg2\*4]. Таким образом, все декодирование выполняется собственными средствами. Все, что нам нужно сделать, это поместить LEA (Load effective address) перед адресной частью инструкции с плавающей запятой и выполнить эту инструкцию. Этот способ реализации самомодифицирующегося кода оказался очень полезным, элегантным и даже безопасным! Следующий код показывает, как это делается в деталях:

```

PROCEDURE Adr*; (* Пустая процедура; освобождает место для 10 байт*)
BEGIN HALT(32) END Adr;
PROCEDURE GetAdr(start, len:LONGINT);
VAR Padr:LONGINT; nop:INTEGER; code:SHORTINT; P:Proc;
BEGIN
  P:=Adr;
  Padr:=S.VAL(LONGINT, P); (*адрес процедуры*)

```

```

nop:=9090H; (*90H, 90H*)
S.PUT(Padr+1, nop);S.PUT(Padr+3, nop); (*Вход с NOP*)
S.PUT(Padr+9, nop);
IF Len # 0 THEN S.MOVE(start+1, Padr+3, len) END; (*Ввод кода *)
S.GET(start, code);
S.PUT(Padr, 90X);S.PUT(Padr+1, 8DX); (*Put NOP, LEA*)
S.PUT(Padr+2, code MOD 8+code DIV 64x64); (*Put dest to EAX*)
S.PUTREG(0,EAX); S.PUTREG(1,ECX); (*Установка регистров*)
S.PUTREG(2,EDX); S.PUTREG(3,EBX);
S.PUTREG(6,ESI); S.PUTREG(7,EDI);
S.PUTREG(5,oldebp); S.PUTREG(5,EBP); (*установить старый EBP*)
Adr; S.GETREG(0,adr); (*Получение адреса*)
S.PUTREG(5,oldebp); (*Сброс EBP*)
END GetAdr;

```

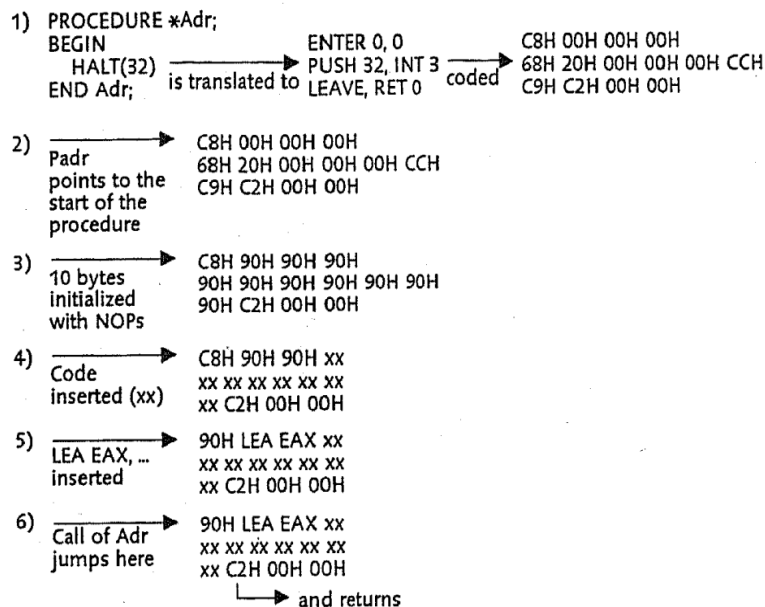


Рис 1: Процесс модификации кода

Все регистры должны быть сохранены в начале процедуры обработки прерывания и должны быть восстановлены снова, прежде чем адрес может быть вычислен. Указатель кадра (EBP) сохраняется перед вызовом Adr и восстанавливается после него.

## 8.1 Математические модули

Как уже упоминалось ранее, эмулятор не знает других операций с плавающей запятой, кроме ADD, SUB, MULT и DIV. Две математические библиотеки,

Math.Obj и MathL.Obj, должны быть адаптированы и оптимизированы для эмуляции, поскольку 80387 имеет инструкции для SORT, логарифмические и тригонометрические функции и многое другое. Наличие математического сопроцессора проверяется во время запуска и флаг DOS.CoAvail устанавливается соответствующим образом. В математических модулях реализованы два варианта, один для сопроцессора, а другой для эмулятора. (См. ниже)

Пример для функции Sin(x):

```
PROCEDURE -Sin(x:REAL):REAL
```

```
    0C8H, 0,0,0 (*ИНТЕР 0,0*)
```

```
    0D9H, 45H, 08H, (*FLD ST(0), 8[EBP]*)
```

```
    0D9H, 0FEH, (*FSIN ST(0)*)
```

```
    0DDH, 0D9H, (*FSTP ST(1)*)
```

```
    0C9H, 0C2H, 4, 0; (*ОСТАВИТЬ, RET 4*)
```

```
PROCEDURE sin*(x:REAL):REAL;
```

```
VAR n: LONGINT; y,yy, f:REAL;
```

```
BEGIN
```

```
    IF DOS.CoAvail THEN (*Сопроцессор существует, используйте инструкции с плавающей точкой*)
```

```
        IF x<0.0 THEN RETURN -Sin(-x) ELSE RETURN Sin(x) END
```

```
ELSE (*Нет сопроцессора, эмулируйте функцию*)
```

```
    y:=c31*x; n:=ENTIER(y+0.5); (*c31=2/pi*)
```

```
    y:=2*(y-n); yy:=y*y;
```

```
    IF~ODD(n) THEN f:=(p33*yy+p32)*yy+p31/(p30+yy)*y
```

```
    ELSE f:=(q33*yy+q32)*yy+q31/(q31+yy) END;
```

```
    IF ODD(nDIV2) THEN f:=-f END;
```

```
RETURN f
```

```
END
```

```
END sin;
```

c31, p33, p32, p31, p30, q33, q32, q31 - это РЕАЛЬНЫЕ константы, используемые для эмуляции.

Эта эмуляция все еще оставляет место для улучшения точности, но она не для обычных требований она подходит. Другие процедуры в модуле Math реализованы аналогичным образом.

Определение Math.Obj:

```
DEFINITION Math;
```

```
    CONST e=2.7182817E+00;
```

```
    VAR pi:REAL;
```

```
    PROCEDURE arctan(x:REAL):REAL;
```

```
    PROCEDURE cos(x:REAL):REAL;
```

```
    PROCEDURE exp(x:REAL):REAL;
```



```

PROCEDURE In (x:REAL) :REAL;
PROCEDURE sin (x:REAL) :REAL;
PROCEDURE sqrt (x:REAL) :REAL;
END Math.

```

Определение MathL.Obj:

```

DEFINITION MathL;
  CONST e=2.71828182845905D+000;
  VAR pi:LONGREAL;
  PROCEDURE arctan (x:LONGREAL) :LONGREAL;
  PROCEDURE cos (x:LONGREAL) :LONGREAL;
  PROCEDURE exp (x:LONGREAL) :LONGREAL;
  PROCEDURE ln (x:LONGREAL) :LONGREAL;
  PROCEDURE sin (x:LONGREAL) :LONGREAL;
  PROCEDURE sqrt (x:LONGREAL) :LONGREAL;
END MathL.

```

Существует множество других функций, предоставляемых сопроцессором, но они не используются для реализации стандартного интерфейса Oberon с плавающей точкой.

## Глава 9. Файловая система

Как показано в [Wir92], файловая система Oberon очень гибкая и предлагает новые перспективы по сравнению с обычными файловыми системами и, в частности, с файловой системой DOS. Хотя в файловой системе Oberon нет подкаталогов, 32-символьные имена файлов обеспечивают достаточную гибкость для создания групп файлов в одном каталоге, т. е. файлы с одинаковым префиксом или суффиксом относятся к одной группе. DOS не позволяет использовать имена файлов длиннее 8+3 символов. Также файлы не могут быть многократно открыты с независимыми указателями на чтение и запись. DOS файлы с разными указателями обязательно ссылаются на одну и ту же позицию чтения/записи. Основой для файловой системы DOS послужила реализация для UNIX. Отсутствие длинных имен файлов делает необходимыми два отдельных каталога. Первый – это обычный каталог, предоставляемый операционной системой DOS. Второй – каталог Oberon, отображающий имена файлов Oberon на имена файлов DOS. Каталог Oberon реализован в виде специального файла под названием FILENAME.TEX. Он считывается в память при запуске. Его представление в памяти – простой линейный список в алфавитном порядке с маркером окончания Sentinel (Рис 1).

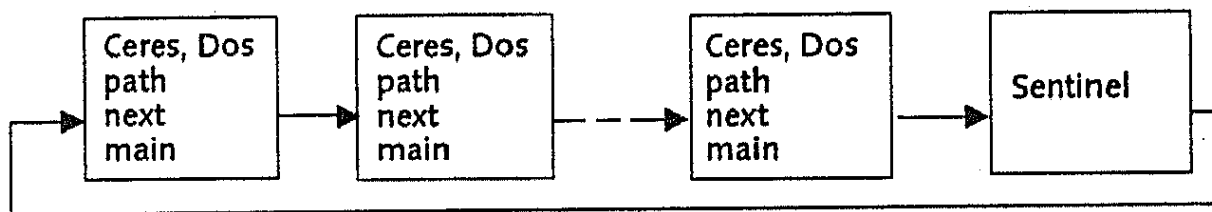


Рис 1: Каталог Oberon

Каждая запись в каталоге Oberon выглядит следующим образом:

```

NodePtr=POINTER TO Node;
Node=RECORD
  Crees, Dos: FileName;
                                (*32-символьное имя, соответствующее 8+3 DOS-имени*)
  path:Path;                    (* Подкаталог, в котором находится файл*)
  next: NodePtr;                (*Указатель на следующий узел*)
  main: BOOLEAN;                (*Главный или текущий каталог*)
END;

```

### Таблица перевода

Когда пользователь выходит из Oberon, таблица перевода записывается обратно на диск. Если непредвиденные обстоятельства препятствуют корректному завершению работы системы Oberon, каталог Oberon не может быть записан на диск. Если при запуске системы не найден FILENAME TEX, то все файлы в каталоге Oberon DOS открываются и проверяются по отдельности. Первые два байта содержат тег Oberon. Последующие 32 байта дают имя Oberon, и соответствующая пара Oberon/DOS может быть вставлена в каталог. Если метка отсутствует, имя Oberon по определению совпадает с именем DOS. Аналогичный алгоритм используется, если файл был добавлен или удален под контролем DOS в каталоге Oberon. Если есть два разных файла с одинаковым именем Oberon, то более старый из них удаляется. Таким образом, таблица трансляции гарантированно сохраняется в любой момент времени.

### Дескриптор файла

Каждый файл внутренне представлен как указатель на файловый дескриптор, т.е. На полный набор информации о файле.

```

File=POINTER TO Handle;
Handle=RECORD
  origName,                      (*Оригинальное 32-символьное имя*)
  workName,                      (*Текущее имя под DOS*)
  registerName                   (*Зарегистрированное имя DOS, пустое при создании
  нового файла*)
  :FileName;
  path: FileDir.Path             (*Подкаталог, в котором находится файл*)
  fd,                           (*Номер файлового хэндла*)
  len,                          (*Длина файла*)
  pos:LONGINT;                  (*Текущая позиция файла*)
  buf                             (*Файлы размером до 16 КБ хранятся в памяти*)

```

```

:ARRAY nofbufs OF Buffer;
swapper,                (*Указывает, какой буфер будет выгружен
следующим*)
state: INTEGER;          (*Состояние: create(файл находится в буферах),
open и close*)
dospath: BOOLEAN;        (*Файл находится в текущем/основном каталоге или
в любом подкаталоге*)
offset: SHORTINT;        (*Файлы Oberon начинаются со смещения 34*)
END;

```

## Новые файлы

Каждый раз, когда новый файл создается с помощью `Files.New(name)`, выделяется новый файловый хэндл, тег и имя вставляются в первый буфер, а имя файла копируется в файл `origName`. Никаких обращений к диску в этот момент не происходит. Каждый файл имеет 4 буфера памяти размером 4 КБ каждый. Таким образом, небольшие файлы могут храниться полностью в памяти, что гарантирует быстрый доступ. Если все буферное пространство используется, временный файл создается под DOS с именем `1.xxx`, где `xxx` обозначает порядковый номер. В файле записи `origName` сохраняется оригинальное имя файла Oberon имя файла. Хотя файл создается на диске сейчас, он не вставляется в каталог Oberon. Это и переименование в `origName` (насколько позволяет ограничение в 8+3 символа) впервые выполняется, когда файл регистрируется командой `Files.Register(f)`. Другие файлы с таким же именем Oberon может открывать одновременно. Затем создаются временные файлы с разными именами (`1.yyy`, см. выше) создаются, но все файлы имеют одинаковое имя `origName`. Все временные файлы, которые были созданы, но не переименованы, т. е. не были зарегистрированы в Oberon, удаляются при следующем запуске системы.

## Старые файлы

Если файл открыт с помощью `Files.Old(имя)`, то соответствующее имя DOS будет найдено в таблице сопоставления, и доступ к файлу будет осуществляться в DOS под своим DOS-именем. В противном случае, если имя отсутствует в каталоге Oberon будет возвращен `NIL`. Каждый открытый файл помещается в кэш, так что второй вызов `Files.Old(name)` найдет уже открытый файл в кэше и вернет тот же файловый хэндл.

```

CONST cacheSize=64;
VAR cache: ARRAY cacheSize OF LONGINT (*=File*)

```

## Регистрация файлов

`Files.Register(file)` промывает все файловые буферы и вставляет пару Oberon/DOS пару имен в каталог Oberon. Если файл уже был создан ранее под временным именем (например, `1.xxx`), он должен быть закрыт, переименовать в соответствующее DOS-имя и снова открыть с теперь уже именем DOS. Это навязано DOS, которая не позволяет переименовывать открытых файлов. Небольшой файл, который может храниться в буферах, может быть создан непосредственно под правильным именем DOS. Поскольку временный файл не был создан, переименование не требуется. Все зарегистрированные файлы также помещаются в кэш, что делает их доступными для будущего доступа.

## Файлы DOS

Теперь можно получить доступ к файлам DOS во всех других каталогах DOS, даже на дискете. Однако для этого сканер должен допускать двоеточие ":" и обратную косую черту "\" в качестве допустимых символов в именах. Если нужно открыть файл в каталоге, например:

a:\mydir\test.mod, файловый модуль проверяет имя файла на наличие "\" в имени, что указывает на путь DOS. Файл test.mod затем ищется в каталоге a:\mydir вместо текущего каталога. Аналогично, если файл сохранен и его имя содержит любое "\", данный путь будет взят вместо текущего пути по умолчанию. Сохраненный таким образом файл не содержит тег и имя Oberon, и он не вставляется ни в одну таблицу отображения. Это позволяет сохранять файлы непосредственно на DOS-диск из Oberon. Это техника доступа к файлам, внешним по отношению к Oberon и может заменить System.CopyToDOS и команды экспорта и импорта System.CopyFromDOS.

## Подкаталоги

Еще один способ поместить файлы в разные каталоги - это использовать подкаталоги. Для этого модуль System предлагает четыре новые команды:

```
System.MakeDir~
    Создает новый подкаталог,
    например, System.MakeDirectory C:\WORK\NEWDIR

System.RemoveDir~
    Удалить указанный подкаталог,
    например, System.RemoveDirectory C:\WORK\NEWDIR
    Перед удалением подкаталог должен быть освобожден.

System.CureentDir~
    Показывает текущий каталог,

System.ChangeDir~
    Устанавливает текущий каталог на указанный,
    например, System.ChangeDirectory C:\WORK\SOURCES~.
```

На самом деле, Oberon поддерживает два каталога, называемые main и current. После загрузки системы оба каталога устанавливаются в реальный текущий каталог в DOS. Если переменная окружения для Oberon была объявлена в файле AUTOEXEC.BAT, то она будет принята за текущий каталог. Все файлы будут располагаться в текущем каталоге, который можно изменить с помощью команды System.ChangeDir. Нужный файл сначала ищется в текущем каталоге, а затем в главном каталоге. Файлы в main не могут быть удалены. Таким образом, каталог main предопределен хранить стабильную версию, в то время как новые реализации размещаются в специальном подкаталоге. Передача данных между текущим и основным каталогами может быть с помощью System.CopyFiles.

```
System.CopyFiles C:\WORK\TEST\NEW.OBJ=>New.Obj~
(*Текущий каталог установлен на C:\WORK*) или
System.CopyFiles New.Obj => C:\WORK\NEW.OBJ~
(*Текущий каталог установлен в C:\WORK\TESTS*)
```

Если в этих командах каталог назначения является либо текущим, либо main, то обновляется каталог Oberon DOS, в противном случае предполагается, что местом назначения является обычный каталог DOS без таблицы сопоставления. Каждое изменение текущего каталога заставляет

резервировать таблицу отображения на диск, поэтому каждый подкаталог, к которому когда-либо обращался Oberon, содержит файл с именем FILENAME.TEX. Это обеспечивает быстрый доступ, когда каталог будет использован позже.

## Ссылки

- [Bro91] Ralf Broun, Jim Kyle: PC Interrupts  
A programmer's reference to BIOS, DOS and third-party calls  
(c) 1991 Addison Wesley Publishing Company, Inc.  
ISBN 0-201-57797-6
- [Dun89] Ray Duncan: Programmierleitfaden für MS-DOS  
Funktionen  
übersetzt von Peter Riswick  
Redmond, Washington: Microsoft Press;  
(c) 1989 Vieweg, Braunschweig  
ISBN 3-528-04650-3
- [Dun90] Ray Duncan: DOS ohne Schranken  
Orig.: Extending DOS, Programming MS-DOS for the  
1990s  
(c) 1990 Addison-Wesley (Deutschland) GmbH  
ISBN 3-89319-300-6
- [Dun90a] Ray Duncan: Programmierhandbuch MS-DOS  
Orig.: Programming in the MS-DOS Environment  
2nd Section of the MS-DOS Encyclopedia  
(c) 1990 Fiedr. Vieweg & Sohn Verlagsgesellschaft mbH  
Braunschweig  
ISBN 3-528-04631-7
- [Els88] Jürgen Elsing  
MS-DOS Assembler Programmierung: praktische  
Anwendungen  
von DOS-Aufrufen in Maschinensprache  
(c) 1988 IWT Verlag GmbH, Vaterstetten bei München  
ISBN 3-88322-8

- [Fer91] Richard F. Ferraro  
Programmer's Guide to the EGA and VGA Cards  
Second Edition, March 1991  
(c) 1990 by Richard F. Ferraro  
Addison-Wesley  
ISBN 0-201-57025-4
- [Mue90] John Mueller and Wallace WANG  
Microsoft Macro Assembler 5.1  
Programming in the 80386 Environment  
(c) 1990 Windcrest books, Division of TAB BOOKS inc.  
ISBN 0-8306-3179-8
- [NS83] NS16000 Instruction Set Reference Manual  
(c) 1983 National Semiconductor Corporation  
California  
Order No. 420010099-001
- [Not92] Semesterarbeit  
Hintergrundbitmaps für DOS-Oberon  
Thomas Notter, Sommersemester 1992
- [Ott92] Semesterarbeit  
VGA Rasteroperationen für DOS-Oberon  
Hans-Werner Ott, Sommersemester 1992
- [Pet91] Semesterarbeit  
i386-Assembler für DOS-Oberon,  
Harald E. Peter, Wintersemester 1991/92
- [Pho89] Phoenix Technical Reference Series  
system BIOS for IBM PC/XT/AT Computers and  
Compatibels  
(c) 1989 Phoenix Technologis, Ltd.  
ISBN 0-201-51806-6
- [Smi87] Bud E. Smith, Mark T. Johnson: Programming the Intel  
80386  
(c) 1987 Scott, Foresman and Company, Glenview,

Illinois

ISBN 0-673-18568-0

[S392] 86C801/86C805 GUI Accelerator

August 1992

S3 Incorporated

2880 San Tomas Expressway

Santa Clara, CA 95051-0981

[Wir92] Niklaus Wirth, Jurg Gutknecht: Project Oberon

The Design of an Operating System and Compiler

(c) 1992 ACM Press

ISBN 0-201-54428-8