Федеральное агентство по образованию  
Государственное образовательное учреждение  
высшего профессионального образования  
«Санкт-Петербургский государственный  
инженерно-экономический университет»

**РЕФЕРАТ**

**«Оперативная память ПК, виды, назначение и основные характеристики»***по дисциплине «Информатика»*

Руководитель А. А. Кантарович  
Исполнитель В. Калмыков  
Группа 2601

Санкт-Петербург  
2010

Содержание.

[1.Оперативная память. 3](#_Toc283222408)

[2.Виды оперативной памяти, их назначение и основные характеристики. 3](#_Toc283222409)

[2.1 Статическая память 3](#_Toc283222410)

[3. Устройство матрицы статической памяти 3](#_Toc283222411)

[4. Типы статической памяти 3](#_Toc283222412)

[4.2 Динамическая память. 3](#_Toc283222413)

[5. Процедуры и функции для работы с динамической памятью. 3](#_Toc283222414)

[Список литературы 3](#_Toc283222415)

# 1.Оперативная память.

Оперативная память - это, в отечественной научной терминологии, "оперативное запоминающее устройство" или ОЗУ, а в западной - RAM, то есть "Random Access Memory" ("память с произвольным доступом"). ОЗУ представляет собой область временного хранения данных, при помощи которой обеспечивается функционирование программного обеспечения. Память состоит из ячеек, каждая из которых предназначена для хранения определенного объема данных, как правило, одного или четырех бит. Чипы памяти работают синхронно с системной шиной. Компьютерная оперативная память является динамической (отсюда - DRAM или Dynamic RAM) - для хранения данных в такой памяти требуется постоянная подача электрического тока, при отсутствии которого ячейки опустошаются. Пример энергонезависимой или постоянной памяти (ПЗУ или ROM - Read Only Memory) памяти - флэш-память, в которой электричество используется лишь для записи и чтения, в то время как для самого хранения данных источник питания не нужен. Ячейки памяти в микросхемах представляют собой конденсаторы, которые заряжаются в случае необходимости записи логической единицы, и разряжаются при записи нуля. Опустошение памяти в случае отсутствия электроэнергии осуществляется именно за счет утечки токов из конденсаторов. (Рис.1)

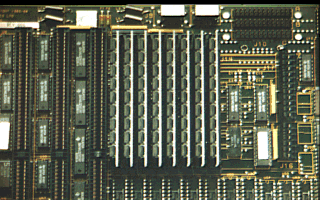
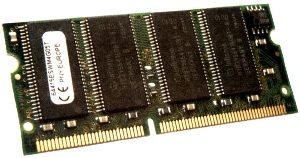


Рис.1



Принцип работы оперативной памяти можно представить следующим образом. Поскольку ячейки организованы в виде двумерной матрицы, для получения доступа к той или иной ячейке необходимо указать адрес соответствующих строки и столбца. Для выбора адреса применяются импульсы RAS# (Row Access Strobe - стробирующий импульс доступа к строке) и CAS# (Column Acess Strobe - стробирующий импульс доступа к столбцу) при которых уровень сигнала (точнее, напряжение) изменяется с высокого на низкий. Эти импульсы синхронизированы с тактирующим импульсом, поэтому оперативная память также называется синхронной (SDRAM). Сначала подается сигнал активации необходимой строки, после чего - импульс RAS#, а затем - CAS#. При операции записи происходит то же самое, за исключением того, что в этом случае подается специальный импульс разрешения записи WE# (Write Enable), который также должен измениться с высокого на низкий. После завершения работы со всеми ячейками активной строки выполняется команда Precharge, позволяющая перейти к следующей строке. Существуют и другие сигналы, но в контексте данной статьи их можно не упоминать, чтобы неоправданно не усложнять материал.

Важнейшая характеристика памяти, от которой зависит производительность - это пропускная способность, которая выражается как произведение частоты системной шины на объем данных, передаваемых за каждый такт. В случае с памятью SDRAM мы имеет шину шириной 64 бита или 8 байт. Следовательно, к примеру, пропускная способность памяти типа DDR333 составляет 333 МГц х 8 Байт = 2,7 Гбайта в секунду или 2700 Мбайт в секунду. Отсюда, кстати, и другое название памяти - PC2700, по ее пропускной способности в мегабайтах в секунду. В последнее время часто используется двухканальное подключение памяти, при котором теоретическая пропускная способность удваивается. То есть, в случае с двумя модулями DDR333 мы получим максимально возможную скорость обмена данных 5,4 Гбайта/с.

Тем не менее, частота работы памяти и, следовательно, ее теоретическая пропускная способность не являются единственными параметрами, отвечающими за производительность. В действительности не менее важную роль играют и латентность памяти, то есть значения задержек между подачей команды и ее выполнением. Эти значения принято называть таймингами, которые выражаются в тактах, прошедших между поступлением какой-либо команды и ее реальным исполнением.Четыре важнейших тайминга, которые всегда используются при описании тех или иных модулей памяти - tRCD, tCL, tRP, tRAS (иногда дополнительно указывается и Command rate), причем записываются они обычно в этой же последовательности в виде 4-4-4-12-(1T) (цифры в данном случае произвольные). Аббревиатура tRCD расшифровывается как timе of RAS# to CAS# Delay - тайминг задержки между импульсами RAS# и CAS#. Сокращение tCL означает timе of CAS# Latency - тайминг задержки относительно импульса CAS# после подачи команды записи или чтения. tRP - это timе of Row Precharge: тайминг между завершением обработки строки и перехода к новой строке. Значение tRAS (time of Active to Precharge Delay) считается одним из основных параметров, поскольку он описывает время задержки между активацией строки и подачей команды Precharge, которой заканчивается работа с этой строкой. Наконец, параметр Command rate означает задержку между командой выбора конкретного чипа на модуле и командой активации строки; обычно эта задержка составляет не более одного-двух тактов.

Общее правило гласит: чем меньше тайминги при одной тактовой частоте, тем быстрее память. Более того, в целом ряде случаев быстрее оказывается память с меньшими таймингами, работающая даже на более низкой тактовой частоте. Все дело в том, что, как мы уже упоминали, оперативная память работает синхронно с системной шиной, поэтому память с частотой, не кратной частоте системной шины и с пропускной способностью, превышающей пропускную способность системной шины никаких преимуществ перед более дешевой не имеет. К примеру, системная шина современных процессоров Pentium 4 работает на частоте 800 МГц, что при ширине шины 64 бит обеспечивает максимальную пропускную способность в 6,4 Гбайта в секунду. Оптимальным выбором для таких чипов является двухканальная память DDR2 400 с аналогичной пропускной способностью в те же 6,4 Гбайта в секунду. Использование в двухканальном режиме более дорогих модулей типа DDR2 533/677 реальной прибавки в производительности вряд ли даст. Более того, в иных случаях есть смысл снизить рабочую частоту таких модулей, но добиться более низких таймингов. Это положительно скажется на производительности - чтобы убедиться в этом, достаточно "прогнать" различные тестовые программы.

## 2.Виды оперативной памяти, их назначение и основные характеристики.

Существует много различных видов оперативной памяти, но их все можно подразделить на две основные подгруппы - статическая память (Static RAM) и динамическая память (Dynamic RAM).

Эти два типа памяти отличаются, прежде всего, различной в корне технологической реализацией - SRAM будет хранить записанные данные до тех пор, пока не запишут новые или не отключат питание, а DRAM может хранить данные лишь небольшое время, после которого данные нужно восстановить (регенерировать), иначе они будут потеряны.

Рассмотрим достоинства и недостатки SRAM и DRAM:

Память типа DRAM, в силу своей технологии, имеет большую плотность размещения данных, чем SRAM.

DRAM гораздо дешевле SRAM, но последняя, производительнее и надежнее, поскольку всегда готова к считыванию.

### 2.1 Статическая память

Статическая память, или SRAM (Statistic RAM) является наиболее производительным типом памяти. Микросхемы SRAM применяются для кэширования оперативной памяти, в которой используются микросхемы динамической памяти, а также для кэширования данных в механических устройствах хранения информации, в блоках памяти видеоадаптеров и т. д. Фактически, микросхемы SRAM используются там, где необходимый объем памяти не очень велик, но высоки требования к быстродействию, а раз так, то оправдано использование дорогостоящих микросхем. В персональных компьютерах с процессорами, у которых не было интегрированной на кристалле кэш-памяти второго уровня, всегда использовались микросхемы SRAM внешнего кэша. Для удешевления системных плат и возможности их модернизации производители системных плат с процессорами 486 и первых поколений Pentium устанавливали специальные кроватки (разъемы для микросхем с DIP-корпусом), в которые можно было устанавливать различные микросхемы SRAM, отличающиеся как по быстродействию и объему памяти, так и различной разрядностью. Для конфигурирования памяти на системной плате предусматривался набор джамперов. Для справки прямо на системной плате краской наносилась информация об установке джамперов, например, как показано в табл.(в колонках JS1 и JS2 указаны номера контактов, которые надо замкнуть перемычками).

Пример таблицы конфигурирования кэш-памяти на системной плате:

|  |  |  |  |
| --- | --- | --- | --- |
| Size | SRAM | JS1 | JS2 |
| 256 К | 32x8 | 1-2 | 1-2 |
| 512 К | 64x8 | 2-3 | 1—2 |
| 1 М | 128x8 | 2-3 | 2-3 |

Отметим, что изменением конфигурации кэш-памяти занимались только тогда, когда выходила из строя какая-либо микросхема кэш-памяти. В остальных случаях изменять положение джамперов не рекомендовалось. В дальнейшем, по мере разработки более совершенных микросхем SRAM, они непосредственно припаивались на системную плату в количестве 1, 2 или 4 штук. На системных платах, которые выпускаются в настоящее время, микросхемы SRAM используются, в основном, только для кэширования ввода/вывода и других системных функций.

### 3. Устройство матрицы статической памяти

Подобно ячейкам динамической, триггеры объединяются в единую матрицу, состоящую из строк (row) и столбцов (column), последние из которых так же называются битами (bit).

В отличии от ячейки динамической памяти, для управления которой достаточно всего одного ключевого транзистора, ячейка статической памяти управляется как минимум двумя. Это не покажется удивительным, если вспомнить, что триггер, в отличии от конденсатора, имеет раздельные входы для записи логического нуля и единицы соответственно. Таким образом, на ячейку статической памяти расходуется целых восемь транзисторов (см. рис.2) - четыре идут, собственно, на сам триггер и еще два - на управляющие "защелки".

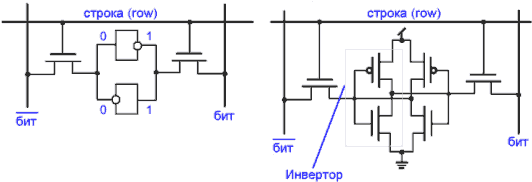


Рис. 2. Устройство 6-транзистроной одно-портовой ячейки SRAM-памяти

Причем, шесть транзисторов на ячейку - это еще не предел! Существуют и более сложные конструкции! Основной недостаток шести транзисторной ячейки заключается в том, что в каждый момент времени может обрабатываться всего лишь одна строка матрицы памяти. Параллельное чтение ячеек, расположенных в различных строках одного и того же банка невозможно, равно как невозможно и чтение одной ячейки одновременно с записью другой.

Этого ограничения лишена многопортовая память. Каждая ячейка многопортовой памяти содержит один-единственный триггер, но имеет несколько комплектов управляющих транзисторов, каждый из которых подключен к "своим" линиям ROW и BIT, благодаря чему различные ячейки матрицы могут обрабатываться независимо. Такой подход намного более прогрессивен, чем деление памяти на банки. Ведь, в последнем случае параллелизм достигается лишь при обращении к ячейкам различных банков, что не всегда выполнимо, а много портовая память допускает одновременную обработку любых ячеек, избавляя программиста от необходимости вникать в особенности ее архитектуры.

Наиболее часто встречается двух - портовая память, устройство ячейки которой изображено на рис. 3. (внимание! это совсем не та память которая, в частности, применяется в кэше первого уровня микропроцессоров Intel Pentium). Нетрудно подсчитать, что для создания одной ячейки двух - портовой памяти расходуется восемь транзисторов. Пусть емкость кэш памяти составляет 32 Кб, тогда только на одно ядро уйдет свыше двух миллионов транзисторов!

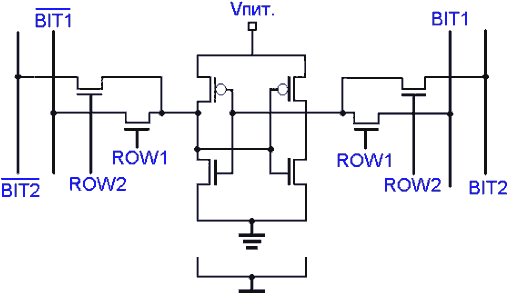


Рис. 3. Устройство 8-транзистроной двух портовой ячейки SRAM-памяти

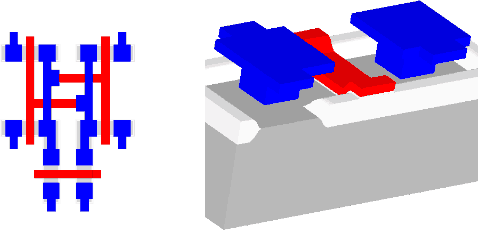


Рис. 3. Ячейка динамической памяти воплощенная в кристалле

### 4. Типы статической памяти

Существует как минимум три типа статической памяти: асинхронная, синхронная и конвейерная. Все они практически ничем не отличаются от соответствующих им типов динамической памяти.

Асинхронная статическая память

Асинхронная статическая память работает независимо от контроллера и потому, контроллер не может быть уверен, что окончание цикла обмена совпадет с началом очередного тактового импульса. В результате, цикл обмена удлиняется по крайней мере на один такт, снижая тем самым эффективную производительность. "Благодаря" последнему обстоятельству, в настоящее время асинхронная память практически нигде не применяется (последними компьютерами, на которых она еще использовались в качестве кэша второго уровня, стали "трешки" - машины, построенные на базе процессора Intel 80386).

Синхронная статическая память

Синхронная статическая память выполняет все операции одновременно с тактовыми сигналами, в результате чего время доступа к ячейке укладывается в один-единственный такт. Именно на синхронной статической памяти реализуется кэш первого уровня современных процессоров.

Конвейерная статическая память

Конвейерная статическая память представляет собой синхронную статическую память, оснащенную специальными "защелками", удерживающими линии данных, что позволяет читать (записывать) содержимое одной ячейки параллельно с передачей адреса другой.

Так же, конвейерная память может обрабатывать несколько смежных ячеек за один рабочий цикл. Достаточно передать лишь адрес первой ячейки пакета, а адреса остальных микросхема вычислит самостоятельно, - только успевай подавать (забирать) записывание (считанные) данные!

За счет большей аппаратной сложности конвейерной памяти, время доступа к первой ячейке пакета увеличивается на один такт, однако, это практически не снижает производительности, т.к. все последующие ячейки пакета обрабатываются без задержек.

Конвейерная статическая память используется в частности в кэше второго уровня микропроцессоров Pentium-II и ее формула выглядит так: 2-1-1-1.

### 4.2 Динамическая память.

Все переменные, объявленные в программе, размещаются в одной непрерывной области оперативной памяти, которая называется сегментом данных. Длина сегмента данных определяется архитектурой микропроцессоров 8086 и составляет 65 536 байт, что может вызвать известные затруднения при обработке больших массивов данных.

С другой стороны, объем памяти ПК (обычно не менее 640 Кбайт) достаточен для успешного решения задач с большой размерностью данных. Выходом из положения может служить использование так называемой динамической памяти.

Динамическая память — это оперативная память ПК, предоставляемая программе при ее работе, за вычетом сегмента данных F4 Кбайт), стека (обычно 16 Кбайт) и собственно тела программы.

Размер динамической памяти можно варьировать в широких пределах (см. приложение 1). По умолчанию этот размер определяется всей доступной памятью ПК и, как правило, составляет не менее 200—300 Кбайт. Динамическая память — это фактически единственная возможность обработки массивов данных большой размерности. Без динамической памяти трудно или невозможно решить многие практические задачи.

Такая необходимость возникает, например, при разработке систем автоматизированного проектирования (САПР): размерность математических моделей, используемых в САПР, может значительно отличаться в разных проектах; статическое (т. е. на этапе разработки САПР) распределение памяти в этом случае, как правило, невозможно. Наконец, динамическая память широко применяется для временного запоминания данных при работе с графическими и звуковыми средствами ПК.

Динамическое размещение данных означает использование динамической памяти непосредственно при работе программы. В отличие от этого, статическое размещение осуществляется компилятором Турбо Паскаля в процессе компиляции программы. При динамическом размещении заранее не известны ни тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным.

Оперативная память ПК представляет собой совокупность элементарных ячеек для хранения информации — байтов, каждый из которых имеет собственный номер. Эти номера называются адресами, они позволяют обращаться к любому байту памяти.

Турбо Паскаль предоставляет в распоряжение программиста гибкое средство управления динамической памятью — так называемые указатели. Указатель — это переменная, которая в качестве своего значения содержит адрес байта памяти.

В ПК адреса задаются совокупностью двух шестнадцатиразрядных слов, которые называются сегментом и смещением. Сегмент — это участок памяти, имеющий длину 65 536 байт F4 Кбайт) и начинающийся с физического адреса, кратного 16 (т. е. О, 16, 32, 48 и т. д.). Смещение указывает, сколько байтов от начала сегмента необходимо пропустить, чтобы обратиться к нужному адресу. Адресное пространство ПК составляет 1 Мбайт (речь идет о так называемой стандартной памяти ПК; на современных компьютерах с процессорами 80386 и выше адресное пространство составляет 4 Гбайт, однако в Турбо Паскале нет средств, поддерживающих работу с дополнительной памятью; при использовании среды Borland Pascal with Objects 7.0 такая возможность имеется).

Для адресации в пределах 1 Мбайт нужно 20 двоичных разрядов, которые получаются из двух шестнадцатиразрядных слов (сегмента и смещения) следующим образом (рис. 6.1): содержимое сегмента смещается влево на 4 разряда, освободившиеся правые разряды заполняются нулями, результат складывается с содержимым смещения. Фрагмент памяти в 16 байт называется параграфом, поэтому можно сказать, что сегмент адресует память с точностью до параграфа, а смещение — с точностью до байта. Каждому сегменту соответствует непрерывная и отдельно адресуемая область памяти. Сегменты могут следовать в памяти один за другим без промежутков или с некоторым интервалом, или, наконец, перекрывать друг друга. Таким образом, по своей внутренней структуре любой указатель представляет собой совокупность двух слов (данных типа WORD), трактуемых как сег- 154.

С помощью указателей можно размещать в динамической памяти любой из известных в Турбо Паскале типов данных. Лишь некоторые из них (BYTE, CHAR, SHORTINT, BOOLEAN) занимают во внутреннем пред- представлении один байт, остальные — несколько смежных. Поэтому на самом деле указатель адресует лишь первый байт данных. 6.3. Объявление указателей.

Как правило, в Турбо Паскале указатель связывается с некоторым типом данных. Такие указатели будем называть типизированными. Для объявления типизированного указателя используется значок л, который помещается перед соответствующим типом, например: var pi AInteger; р2 : "Real; type PerconPomter = "PcrconRecord; PerconRecord = record Name : String; Job : String; Next : PerconPomter end; Обратите внимание: при объявлении типа PerconPointer мы сослались на тип PerconRecord, который предварительно в программе объявлен не был. Как уже отмечалось, в Турбо Паскале последовательно проводится в жизнь принцип, в соответствии с которым перед использованием какого-либо идентификатора он должен быть описан. Исключение сделано только для указателей, которые могут ссылаться на еще не объявленный тип данных. Это сделано не случайно.

Динамическая память дает возможность реализовать широко применяемую в некоторых программах организацию данных в виде списков. Каждый элемент списка имеет в своем составе указатель на соседний элемент (рис. 6.2), что обеспечивает возможность просмотра и коррекции списка. Если бы в Турбо Паскале не было этого исключения, реализация списков была бы значительно затруднена. В Турбо Паскале можно объявлять указатель и не связывать его при этом с каким-либо конкретным типом данных. Для этого служит стандартный тип POINTER, например: var р: pointer.

1-й элемент списка Указатель — 2-й элемент списка Последний элемент списка NIL Рис. 6.2. Списочная структура данных Указатели такого рода будем называть нешипизированными. Поскольку нети- пизированные указатели не связаны с конкретным типом, с их помощью удобно динамически размещать данные, структура и тип которых меняются в ходе работы программы.

Как уже говорилось, значениями указателей являются адреса переменных в памяти, поэтому следовало бы ожидать, что значение одного указателя можно передавать другому. На самом деле это не совсем так. В Турбо Паскале можно передавать значения только между указателями, связанными с одним и тем же типом данных. Если, например, объявлены переменные pl,p2; "Integer; рЗ : лЯоа1; рр : pointer; то присваивание pl := р2; вполне допустимо, в то время как присваивание pl := рЗ; запрещено, поскольку Р1 и РЗ указывают па разные типы данных. Это огра- ограничение, однако, не распространяется на нетипизированные указатели, по- поэтому мы могли бы записать рр :- рЗ, pl := рр; и тем самым достичь нужного результата.

Читатель вправе задать вопрос, стоило ли вводить ограничения и тут же давать средства для их обхода. Все дело в том, что любое ограничение, с одной стороны, вводится для повышения надежности программ, а с другой — уменьшает мощность языка, делает его менее пригодным для каких-то Применений.

В Турбо Паскале немногочисленные исключения в отношении типов данных придают языку необходимую гибкость, но их использование требует от программиста дополнительных усилий и таким образом свиде- свидетельствует о вполне осознанном действии.

Выделение и освобождение динамической памяти Вся динамическая память в Турбо Паскале рассматривается как сплошной массив байтов, который называется кучей. Физически куча располагается в старших адресах сразу за областью памяти, которую занимает тело программы. Начало кучи хранится в стандартной переменной HeapOrg (рис. 6.3), ко- конец — в переменной HeapEnd. Текущую границу незанятой динамической памяти содержит переменная Heapptr. Памяхь под любую динамически размещаемую переменную выделяется процедурой NEW. Параметром обращения к этой процедуре является типизированный указатель. В результате обращения указатель приобретает значение, соответствующее динамическому адресу, начиная с которого можно разместить данные, например: var i, j : "Integer; г : AReal; begin New(i); end.

После выполнения этого фрагмента указатель 1 приобретет значение, которое перед этим имел указатель кучи HEAPPTR, а сам HEAPPTR увеличит свое значение на 2, т. к. длина внутреннего представления типа INTEGER, с кото- которым связан указатель I, составляет 2 байта (на самом деле это не совсем гак: память под любую переменную выделяется порциями, кратными 8 байтам — см. разд. 6.7). Оператор new (г) ; вызовет еще раз смещение указателя HEAPPTR, но теперь уже на 6 байт, потому что такова длина внутреннего представления типа REAL Аналогичным образом выделяется память и для переменной любого другого типа. После того как указатель приобрел некоторое значение, т. е. стал указывать на конкретный физический байт памяти, по этому адресу можно разместить любое значение соответствующего типа. Для этого сразу за указателем без каких-либо пробелов ставится значок л, например: i = 2, (В область памяти i помещено значение 2} гл = 2\*pi; {В область памяти г помещено значение 6.28}\

Расположение кучи в памяти ПК Таким образом, значение, на которое указывает указатель, т. е. собственно данные, размещенные в куче, обозначаются значком Л, который ставится сразу за указателем. Если за указателем нет значка, то имеется в виду ад- адрес, по которому размещены данные. Имеет смысл еще раз задуматься над только что сказанным: значением любого указателя является адрес, а чтобы указать, что речь идет не об адресе, а о тех данных, которые размещены по этому адресу, за указателем ставится Л. Если вы четко уясните себе это, у вас не будет проблем при работе с динамической памятью. Динамически размещенные данные можно использовать в любом месте программы, где это допустимо для констант и переменных соответствую- соответствующего типа, например: гЛ :<\* sqr (rA) + \Л - 17; Разумеется, совершенно недопустим оператор г := sqr(rA) + iA - 17; т. к. указателю r нельзя присвоить значение вещественного выражения. Точно так же недопустим оператор гл := sqr (г) ; поскольку значением указателя r является адрес, и его (в отличие от того значения, которое размещено по этому адресу) нельзя возводить в квадрат. Ошибочным будет и такое присваивание: =х; 158 Ядро Турбо Паскаля т. к. вещественным данным, на которые указывает R, нельзя присвоить значение указателя (адрес).

Динамическую память можно не только забирать из кучи, но и возвращать обратно. Для этого используется процедура DISPOSE. Например, операторы disposed) ; dispose(i); вернут в кучу 8 байт, которые ранее были выделены указателям 1 и R (см. выше). Отметим, что процедура dtspose (PTR) не изменяет значения указателя PTR, а лишь возвращает в кучу память, ранее связанную с этим указателем. Од- Однако повторное применение процедуры к свободному указателю приведет к возникновению ошибки периода исполнения. Освободившийся указатель программист может пометить зарезервированным словом NIL. Помечен ли какой-либо указатель или нет, можно проверить следующим образом: const р: 4lGal = NIL; begin if p = NIL then new(p); dispose(p) ; p : NIL; end. Никакие другие операции сравнения над указателями не разрешены. Приведенный выше фрагмент иллюстрирует предпочтительный способ объявления указателя в виде типизированной константы (см. главу 7) с одновременным присвоением ему значения NIL. Следует учесть, что начальное значение указателя (при его объявлении в разделе переменных) может быть произвольным. Использование указателей, которым не присвоено значение процедурой NEW или другим способом, не контролируется системой и может привести к непредсказуемым результатам.

Чередование обращений к процедурам NEW и DISPOSE обычно приводит к "ячеистой" структуре памяти. Дело в том, что все операции с кучей выполняются под управлением особой подпрограммы, которая называется администратором кучи. Она автоматически пристыковывается к вашей программе компоновщиком Турбо Паскаля и ведет учет всех свободных фрагментов в куче. При очередном обращении к процедуре NEW эта подпрограмма отыскивает наименьший свободный фрагмент, в котором еще может разместиться.

Адрес начала найденного фрагмента возвращается в указателе, а сам фрагмент или его часть нужной длины помечается как занятая часть кучи. (Подробнее о работе администратора кучи см. в разд. 6.7). Другая возможность состоит в освобождении целого фрагмента кучи. С этой целью перед началом выделения динамической памяти текущее значение указателя HEAPPTR запоминается в переменной-указателе с помощью процедуры MARK Теперь можно в любой момент освободить фрагмент кучи, на- начиная от того адреса, который запомнила процедура MARK, и до конца дина- динамической памяти.

Для этого используется процедура RELEASE Например: var p,pl ,р?, рЗ,р4,р5 : ЛInteger; begin new(pi); new (p.°); mark(p); new (pi) ; new(p4); new(p5) release (p) f end. В этом примере процедурой MARK{P) в указатель р было помещено текущее значение HEAPPTR, однако память под переменную не резервировалась. Обращение RELEASE (P) освободило динамическую память от помеченного места до конца кучи. На рис. 6.4 проиллюстрирован механизм работы процедур NEW—DISPOSE И NEW—MARK—RELEASE ДЛЯ рассмотренного Примера и для случая, когда вместо оператора RELEASE (Р) используется, например, DISPOSE (РЗ). Следует учесть, что вызов RELEASE уничтожает список свободных фрагментов в куче, созданных до этого процедурой DISPOSE, поэтому совместное использование обоих механизмов освобождения памяти в рамках одной программы не рекомендуется.

Как уже отмечалось, параметром процедуры NEW может быть только типизированный указатель. Для работы с нетипизированными указателями служат процедуры: GETMEM (P, SIZE) — резервирование памяти; О FREEMEM(P, SIZE) освобождение памяти.

Понятно, что наличие нетипизированных указателей в Турбо Паскале (в стандартном Паскале их нет) открывает широкие возможности неявного преобразования типов. К сожалению, трудно обнаруживаемые ошибки в программе, связанные с некорректно используемыми обращениями к про- процедурам NEW и DISPOSE, также могут привести к нежелательному преобразо- преобразованию типов. В самом деле, пусть имеется программа: {i :- HeapOrg; HeapPtr ;-- HeapOrg + 2} {j :- HeapOrg) {HeapPtr ;- HeapOrg} {r := HeapOrg; HeapPtr -•¦» HeapOrg + бу! i ~i \* r : " begin new(i); j := i; Y := 2 dispose new(r) ; Integer; Real; r (i); гл := pi; end.

Что будет выведено на экран дисплея? Чтобы ответить на этот вопрос, про- проследим за значениями указателя HEAPPTR.

Перед исполнением программы этот указатель имел значение адреса начала кучи HEAPORG, которое и было передано указателю I, а затем и J. После выполнения DISPOSE (I) указатель кучи вновь приобрел значение HEAPORG, этот адрес передан указателю R в процедуре NEW(R).

После того как по адресу R разместилось вещественное число я = 3,14159, первые 2 байта кучи оказались заняты под часть внутрен- внутреннего представления этого числа. В то же время j все еще сохраняет адрес HEAPORG, поэтому оператор WRITELN(OT) будет рассматривать 2 байта числа л как внутреннее представление целого числа (ведь j — это указатель на тип INTEGER) и выведет 8578. 6.5. Использование указателей Подведем некоторые итоги. Итак, динамическая память составляет 200— 300 Кбайт или больше, ее начало хранится в переменной HEAPORG, а конец соответствует адресу переменной HEAPEND.

Текущий адрес свободного участка динамической памяти хранится в указателе HEAPPTR Посмотрим, как можно использовать динамическую память для размещения крупных массивов данных. Пусть, например, требуется обеспечить доступ к элементам прямоугольной матрицы 100x200 типа EXTENDED Для размещения такого массива требуется память размером 200 000 байт A00x200x10). Казалось бы, эту проблему можно решить следующим образом: var i,j : Integer; PtrArr : array [1..100, 1..200] of AReal; begin for l := 1 to 100 do for j := 1 to 200 do new(PtiArr[i,]l); and. Теперь к любому элементу вновь созданного динамического массива можно обратиться по адресу, например: PtrArr[1,11Л :- 0; if PtrArr[I,j\*2JA > 1 then Вспомним, однако, что длина внутреннего представления указателя состав- составляет 4 байта, поэтому для размещения массива PTRARR потребуется ЮОх х200х4 = 80 000 байт, что превышает размер сегмента данных F5 536 байт), доступный, как уже отмечалось, программе для статического размещения данных.

Выходом из положения могла бы послужить адресная арифметика, т. е. арифметика над указателями, потому что в этом случае можно было бы от- отказаться от создания массива указателей ptrarr и вычислять адрес любого элемента прямоугольной матрицы непосредственно перед обращением к нему.

Однако в Турбо Паскале над указателями не определены никакие операции, кроме операций присваивания и отношения. Тем не менее, решить указанную задачу все-таки можно. Как мы уже знаем, любой указатель состоит из двух слов типа WORD, в которых хранятся сегмент и смещение. В Турбо Паскале определены две встроенные функции типа WORD, позволяющие получить содержимое этих слов: ? SEG(X) — возвращает сегментную часть адреса; ? OFS (X) — возвращает смещение. Аргументом х при обращении к этим функциям может служить любая пере- переменная, в том числе и та, на которую указывает указатель Например, если имеем р : begin new(p); pn := 3.14; end. то функция seg(P) вернет сегментную часть адреса, по которому располага- располагается 4-баЙтный указатель р, в то время как seg(P") — сегмент 6-байтного участка кучи, в котором хранится число 3,14.

Таким образом, возможна такая последовательность действий. Вначале процедурой getmem из кучи забираются несколько фрагментов подходящей дли- длины (напомню, что за одно обращение к процедуре можно зарезервировать не более 65 521 байт динамической памяти). Для рассматриваемого примера удобно резервировать фрагменты такой длины, чтобы в них могли, напри- например, разместиться строки прямоугольной матрицы, т. е. 200x10 = 2000 байт. Начало каждого фрагмента, т. е. фактически начало размещения в памяти каждой строки, запоминается в массиве PTRSTR, состоящем из 100 указателей. Теперь для доступа к любому элементу строки нужно вычислить смещение этого элемента от начала строки и сформировать соответствующий указатель: var Integer; array A , .100J of pointer; PtrStr pr const SizeOfReal begin for l := 1 to 100 do GetMem(PtrStr[i]rSizeOfReal\*200); 6; /Обращение к элементу матрицы [i,j]} pr := ptr(seg(PtrStr[i]"), ofs(PtrStr[i.]") + [j-l) \*SizeOtReal) ; if pr" > 1 then end.

Поскольку оператор вычисления адреса PR : = PTR.. . будет, судя по всему, использоваться в программе неоднократно, полезно ввести вспомогательную функцию GETR, возвращающую значение элемента матрицы, и процедуру PUTR, устанавливающую новое значение элемента (правила объявления про- процедур и функций изложены в главе 8).

Каждая из них, в свою очередь, обращается к функции ADDRR для вычисления адреса. В примере 6.1 приводится программа, создающая в памяти матрицу из NxM случайных чисел и вычисляющая их среднее значение. const SizeOfReal = 6; N = 100; М = 200; var l, ] : Integer; PtrStr: array [1..N] of pointer; s : Reel; (Длина переменной типа REAL} {Количество столбцов} {Количество строк} /64 type RealPomt = ~Rcal; {— --) Function AddrR (i,j : word} : RealPoint; {По сегменту i и смещению j выдает адрес вещественной перемерной} begin AddrR := ptr (seg (PtrStr [1] л) , ofs (PtrStr [i]'v) + (j-l) \SizeOfReal) end {AddrR I; { 1 Function GetRA,j: Integer): Real; {Выдает ¦значение вещественной переменной по сегменту i и смещению j ее адреса} begin GetR := AddrR (i.,])A end I GetR); {- ¦-- I Procepure PiitR(i,j - Integer; x- Rpal); {Помещает в переменную, адрес которой имеет сегмент i и смещение j, вещественное значение у] begin AddrR (x, j) " -.= х end {PutR); {- } begin (Main! for i :=] to N do begin GetMem(PtrStr[i] ,M^SizeOfRea!) ; for j •= 1 to M do PutR(i, j. Random) end; s := 0; fcr l := 1 to N do for j := 1 to M do s := s + GetR(i, j) ; WriteLn(S / (N \* M) : 12:10) end {Mam}.

В рассмотренном примере предполагается, что каждая строка размещается в куче, начиная с границы параграфа, и смещение для каждого указателя PTRSTR равно нулю. В действительности при последовательных обращениях к процедуре GF.TMEM начало очередного фрагмента следует сразу за концом предыдущего и может не попасть на границу сегмента. В результате, при размещении фрагментов максимальной длины F5 521 байт) может возникнуть переполнение при вычислении смещения последнего байта.

### 5. Процедуры и функции для работы с динамической памятью.

Ниже приводится описание как уже рассмотренных процедур и функций, так и некоторых других, которые могут оказаться полезными при обращении к динамической памяти. Функция addr. Возвращает результат типа POINTER, в котором содержится адрес аргумента. Обращение: ADDR (X) Здесь х — любой объект программы (имя любой переменной, процедуры, функции). Возвращаемый адрес совместим с указателем любого типа. Отметим, что аналогичный результат возвращает операция.

Функция CSEG. Возвращает значение, хранящееся в регистре cs микропроцессора (в начале работы программы в регистре содержится сегмент начала кода программы). Обращение: CSEG Результат возвращается в слове типа WORD Процедура DISPOSE. Возвращает в кучу фрагмент динамической памяти, который ранее был зарезервирован за типизированным указателем. Обращение: DISPOSE(TP) Здесь ТР типизированный указатель. При повторном использовании про- процедуры применительно к уже освобожденному фрагменту возникает ошибка периода исполнения. При освобождении динамических объектов можно указывать вторым параметром обращения к DISPOSE имя деструктора.

Функция DSEG. Возвращает значение, хранящееся в регистре DS микропроцессора (в начале работы программы в регистре DS содержится сегмент на- начала данных программы). Обращение: DSEG Результат возвращается в слове типа WORD 16й Часть I Ядро Турбо Паскаля Процедура ffeemem. Возвращает в кучу фрагмент динамической памяти, кото- который ранее был зарезервирован за нетипизированным указателем. Обращение: FREEMEM (P, SIZE) Здесь: ? р — нетипизированный указатель; ? SIZE — длина в байтах освобождаемого фрагмента.

При повторном использовании процедуры применительно к уже освобожденному фрагменту возникает ошибка периода исполнения. Процедура getmem. Резервирует за нетипизированным указателем фрагмент динамической памяти требуемого размера. Обращение: GFTMFM (P, ST7F) За одно обращение к процедуре можно зарезервировать не более 65 521 байт динамической памяти. Если нет свободной памяти требуемого размера, воз- возникает ошибка периода исполнения.

Если память не фрагментирована, последовательные обращения к процедуре будут резервировать последователь- последовательные участки памяти, так что начало следующего будет располагаться сразу за концом предыдущего. Процедура MARK Запоминает текущее значение указателя кучи HEAPPTR Об- Обращение: MARK (PTR) Здесь PTR — указатель любого типа, в котором будет возвращено текущее значение HEAPPTR Используется совместно с процедурой RELEASE для освобождения части кучи.

Функция maxavail. Возвращает размер в байтах наибольшего непрерывного участка кучи. Обращение: MAXAVAIL Результат имеет тип LONGINT За один вызов процедуры NEW или GETMEM нельзя зарезервировать памяти больше, чем значение, возвращаемое этой функцией. Функция mema VAIL. Возвращает размер в байтах общего свободного пространства кучи. Обращение: MEMAVAIL Результат имеет тип LONGINT Процедура NEW. Резервирует фрагмент кучи для размещения переменной. Обращение: NEW (TP) Здесь тр — типизированный указатель.

За одно обращение к процедуре можно зарезервировать не более 65 521 байт динамической памяти. Если нет свободной памяти требуемого размера, возникает ошибка периода исполнения. Если память не фрагментирована, последовательные обращения к процедуре будут резервировать последователь – последовательные участки памяти, так что начало следующего будет располагаться сразу за концом предыдущего. Процедура NEW может вызываться как функция. В этом случае параметром обращения к ней является тип переменной, размещаемой в куче, а функция NEW возвращает значение типа "указатель". Например: type Pint -ЛInteger; var р: Pint; begin p = New (Pint) ; end.

При размещении в динамической памяти объекта разрешается в качестве второго параметра обращения к NEW указывать имя конструктора (см. главу 10). Функция OFS. Возвращает значение типа WORD, содержащее смещение адреса указанного объекта. Вызов: OFS (X) Здесь х — выражение любого типа или имя процедуры. Функция PTR. Возвращает значение типа POINTER по заданному сегменту SEG и смещению OFS. Вызов: PTR (SEG, OFS) Здесь: О SEG — выражение типа WORD, содержащее сегмент; ? OFS — выражение типа WORD, содержащее смещение.

Значение, возвращаемое функцией, совместимо с указателем любого типа. Процедура RELEASE. Освобождает участок кучи. Обращение: RELEASE (PTR) Здесь PTR — указатель любого типа, в котором предварительно было сохранено процедурой MARK значение указателя кучи. Освобождается участок кучи от адреса, хранящегося в PTR, до конца кучи. Одновременно уничтожается список всех свободных фрагментов, которые, возможно, были созданы процедурами DISPOSE ИЛИ FREEMEM. Функция SEG.

Возвращает значение типа WORD, содержащее сегмент адреса указанного объекта. Вызов: SEG (X) Здесь х — выражение любого типа или имя процедуры. Функция SIZEOF. Возвращает длину в байтах внутреннего представления указанного объекта. Вызов: SIZEOF (X) Здесь х — имя переменной, функции или типа. Например, везде в программе из примера 6.1 вместо константы SIZEOFREAL можно было бы использовать обращение SIZEOF (REAL) 6.7. Администратор кучи

Как уже отмечалось, администратор кучи — это служебная подпрограмма, которая обеспечивает взаимодействие пользовательской программы с кучей. Администратор кучи обрабатывает запросы процедур NEW, getmem, DISPOSE, FREEMEM и др. и изменяет значения указателей HEAPPTR и FREELIST. Указа- Указатель HEAPPTR содержит адрес нижней границы свободной части кучи, а ука- указатель FREELIST — адрес описателя первого свободного блока. В модуле SYSTEM указатель FREELIST описан как POINTER, однако фактически он указывает на следующую структуру данных: type PFreeRec = "TFieeRac; TFreeRec - record Next : pointer; Size : pointer end, Эта списочная структура предназначена для описания всех свободных блоков па- памяти, которые расположены ниже границы HEAPPTR.

Происхождение блоков связано со случайной последовательностью использования процедур NEW—DISPOSE или GETMEM— FREEMEM ("ячеистая" структура кучи). Поле NEXT в записи TFREEREC содержит адрес описателя следующего по списку свободного блока кучи или ад- адрес, совпадающий с HEAPEND, если этот участок последний в списке. Поле SIZE содержит ненормализованную длину свободного блока или 0, если ниже адреса, содержащегося в HEAPPTR, нет свободных блоков., Ненормализованная длина оп- определяется так: в старшем слове этого поля содержится количество свободных параграфов, а в младшем — количество свободных байтов в диапазоне 0—15.

Следующая функция преобразует значение поля SIZE в фактическую длину свобод- свободного блока: Function BlockSizelSize: pointer): Longint; {Функция преобразует ненормализованную длину свободного блока в байты} type PtrRec = record Lo, Hi : word end; var LengthBlock: Longmt; begin BlockSize := Longlnt(PtrRec(Size).Hi)\*16 + PtrRec(S.ze).Lo end; Сразу после загрузки программы указатели heabptr и E'REELIST содержат один и тот же адрес, который совпадает с началом кучи (этот адрес содер- содержится в указателе HEAPORG) При этом в первых 8 байтах кучи хранится за- запись, соответствующая типу TFREEREC (поле NEXT содержит адрес, совпа- совпадающий со значением HEAPEND, а поле SIZE — ноль, что служит дополнительным признаком отсутствия "ячеек" в динамической памяти). При работе с кучей указатели HEAPPTR и FREELIST будут иметь одинаковые значения до тех пор, пока в куче не образуется хотя бы один свободный блок ниже границы, содержащейся в указателе HEAPPTR.

Как только это произойдет, указатель FREELIST станет ссылаться на начало этого блока, а в первых 8 байтах освобожденного участка памяти будет размещена запись TFREEREC Используя FREELIST как начало списка, программа пользователя всегда сможет просмотреть весь список свободных блоков и при необходимости модифицировать его. Описанный механизм вскрывает один не очень существенный недостаток, связанный с работой администратора кучи, а именно: в любой освободившийся блок администратор должен поместить описатель этого блока, а это означает, что длина блока не может быть меньше 8 байт. Администратор кучи всегда выцеляет память блоками, размер которых кратен размеру записи TFREEREC, т. е. кратен 8 байтам. Даже если программа запросит 1 байт, администратор выделит ей фактически 8 байт. Те же 8 байт будут выделены при запросе 2, 3, ..., 8 байт; при запросе 9 байт будет выделен блок в 16 байт и т. д.

Это обстоятельство следует учитывать, если вы хотите минимизировать возможные потери динамической памяти. Если запрашиваемый размер не кратен 8 байтам, в куче образуется "дырка" размером от 1 до 7 байт, при- 17Q Часть I. Ядро Турбо Паскаля чем она не может использоваться ни при каком другом запросе динамической памяти вплоть до того момента, когда связанная с ней переменная не будет удалена из кучи. Если при очередном обращении к функции NEW или GETMEM администратор не может найти в куче нужный свободный блок, он обращается к функции, адрес которой содержит переменная HEAPERROR

Эта функция соответствует следующему процедурному типу: type HeapErrorFun = function (Size : word) : Integer; Здесь SIZE — размер той переменной, для которой нет свободной динамической памяти. Стандартная функция, адрес которой при запуске программы содержит переменная heaperror, возвращает 0, что приводит к останову программы по ошибке периода счета с кодом 203 (см. приложение 3). Вы можете переопределить эту функцию и таким образом блокировать останов программы. Для этого необходимо написать собственную функцию и поместить ее адрес в указатель HEAPERROR. Например: Function HeapFunc(SlZe: Word) : Integer; far; begin HeapFunc := 1 end; begin {Основная программа) HeapError := @HeapFunc; end.

Отметим, что функция типа HEAPERRORFUN вызывается только в том случае, когда обращение с требованием выделения динамической памяти было не- неуспешным. Она может возвращать одно из трех значений: ? 0 — прекратить работу программы; 0 1 — присвоить соответствующему указателю значение NIL и продолжить работу программы; 0 2 — повторить выделение памяти; разумеется, в этом случае внутри функции типа HEAPERRORFUN необходимо освободить память нужного размера.

### Список литературы

1. Бройдо В.Л. Теория информации. Учебное пособие.- 2-е изд.- СПб: Питер, 2005.- 702с.
2. Под ред. А.Д.Урсула. История Прикладная теория информации Учебное пособие. СПб: РАГС, 2006.- 433 с.
3. Под ред. А.Д.Хомоненко Основы современных компьютерных технологий: Учебник/ - СПб: Корона принт, 2009.- 672 с.
4. М.: ФАИР-ПРЕСС Организация информационного производства: Учебное пособие.-, 2005.- 247 с..-(Специальный издательский проект для библиотек)
5. Петров Ю.П. Оперативная память. / Ю.П.Петров.- СПб: БХВ-Петербург, 2005.- 441 с.-(Учебное пособие)