Содержание

1а 1. Информатика. Информация. Представление и обработка информации. Системы счисления

Информатика занимается формализованным представлением объектов и структур их взаимосвязей в различных областях науки, техники, производства. Для моделирования объектов и явлений используются различные формальные средства, например логические формулы, структуры данных, языки программирования и др.

В информатике такое фундаментальное понятие, как **информация** имеет различные значения:

1) формальное представление внешних форм информации;
2) абстрактное значение информации, ее внутреннее содержание, семантика;
3) отношение информации к реальному миру.

Но, как правило, под информацией понимают ее абстрактное значение — **семантику**. Если мы хотим обмениваться информацией, нам необходимы согласованные представления, чтобы не нарушалась правильность интерпретации. Для этого интерпретацию представления информации отождествляют с некоторыми математическими структурами. В этом случае обработка информации может быть выполнена строгими математическими методами.

Одно из математических описаний информации — это представление ее в виде функции

$$y = f(x,t)$$

где t — время,

x — точка некоторого поля,

в которой измеряется значение y. В зависимости от параметров функции x и t информацию можно классифицировать.

2а 2. Представление чисел в ЭВМ. Формализованное понятие алгоритма

32-разрядные процессоры могут работать с оперативной памятью емкостью до 232-1, а адреса могут записываться в диапазоне 00000000 — FFFFFFFF. Однако в реальном режиме процессор работает с памятью до 220-1, а адреса попадают в диапазон 00000 — FFFFF. Байты памяти могут объединяться в поля как фиксированной, так и переменной длины. **Словом** называется поле фиксированной длины, состоящее из 2 байтов, **двойным словом** — поле из 4 байтов. Адреса полей бывают **четные** и **нечетные**, при этом для четных адресов операции выполняются быстрее.

Числа с фиксированной точкой в ЭВМ представляются как целые двоичные числа, и занимаемый ими объем может составлять 1, 2 или 4 байта.

Целые двоичные числа представляются в дополнительном коде. Дополнительный код положительного числа равен самому числу, а дополнительный код отрицательного числа может быть получен по такой формуле:

$$x = 10n - |x|,$$

где n — разрядность числа.

В двоичной системе счисления дополнительный код получается путем инверсии разрядов, т. е., заменой единиц нулями и наоборот, и прибавлением единицы к младшему разряду.

Количество битов мантиссы определяет точность представления чисел, количество битов машинного порядка определяет диапазон представления чисел с плавающей точкой.

Формализованное понятие алгоритма

Алгоритм может существовать только тогда, когда в то же самое время существует некоторый матема-

3а 3. Введение в язык Pascal

Основные символы языка — буквы, цифры и специальные символы — составляют его алфавит. Язык Pascal включает следующий набор основных символов:

1) 26 латинских строчных и 26 латинских прописных букв:
2) _ (знак подчеркивания);
3) 10 цифр: 0 1 2 3 4 5 6 7 8 9;
4) знаки операций:
   + — O / = <> < > <= >= := @ ;
5) ограничители: . , ' ( ) [ ] (. .) { } (* *) .. : ;
6) спецификаторы: ^ # $;
7) служебные (зарезервированные) слова:

ABSOLUTE, ASSEMBLER, AND, ARRAY, ASM, BEGIN, CASE, CONST, CONSTRUCTOR, DESTRUCTOR, DIV, DO, DOWNTO, ELSE, END, EXPORT, EXTERNAL, FAR, FILE, FOR, FORWARD, FUNCTION, GOTO, IF, IMPLEMENTATION, IN, INDEX, INHERITED, INLINE, INTERFACE, INTERRUPT, LABEL, LIBRARY, MOD, NAME, NIL, NEAR, NOT, OBJECT, OF, OR, PACKED, PRIVATE, PROCEDURE, PROGRAM, PUBLIC, RECORD, REPEAT, RESIDENT, SET, SHL, SHR, STRING, THEN, TO, TYPE, UNIT, UNTIL, USES, VAR, VIRTUAL, WHILE, WITH, XOR.

Кроме перечисленных, в набор основных символов входит пробел.

В языке Pascal существует правило: тип явно задается в описании переменной или функции, которое предшествует их использованию. Концепция типа языка Pascal имеет следующие основные свойства:

1) любой тип данных определяет множество значений, к которому принадлежит константа, которые может принимать переменная или выражение либо вырабатывать операция или функция;
2) тип значения, задаваемого константой, переменной или выражением, можно определить по их виду или описанию;

4а 4. Стандартные процедуры и функции

Арифметические функции

1. Function Abs(X); возвращает абсолютное значение параметра.
2. Function ArcTan(X: Extended): Extended; возвращает арктангенс аргумента.
3. Function Exp(X: Real): Real; возвращает экспоненту.
4. Function Frac(X: Real): Real; возвращает дробную часть аргумента.
5. Function Int(X: Real): Real; возвращает целочисленную часть аргумента.
6. Function Ln(X: Real): Real; возвращает натуральный логарифм (Ln e = 1) выражения X вещественного типа.
7. Function Pi: Extended; возвращает значение Pi, которое определено как 3.1415926535.
8. Function Sin(X: Extended): Extended; возвращает синус аргумента.
9. Function Sqr(X: Extended): Extended; возвращает квадрат аргумента.
10. Function Sqrt(X: Extended): Extended; возвращает квадратный корень аргумента.

Процедуры и функции преобразования величин

1. Procedure Str(X [: Width [: Decimals]]; var S); преобразовывает число X в строковое представление.
2. Function Chr(X: Byte): Char; возвращает символ с порядковым номером X в ASCII-таблице.
3. Function High(X); возвращает наибольшее значение в диапазоне параметра.
4. Function Low(X); возвращает наименьшее значение в диапазоне параметра.
5. Function Ord(X): LongInt; возвращает порядковое значение выражения перечислимого типа.

2б тический объект. Формализованное понятие алгоритма связано с понятием рекурсивных функций, нормальных алгоритмов Маркова, машин Тьюринга.

В математике функция называется однозначной, если для любого набора аргументов существует закон, по которому определяется единственное значение функции. В качестве такого закона может выступать алгоритм; в этом случае функция называется вычислимой.

Рекурсивные функции — это подкласс вычислимых функций, а алгоритмы, определяющие вычисления, называются сопутствующими алгоритмами рекурсивных функций. Сначала фиксируются базовые рекурсивные функции, для которых сопутствующий алгоритм тривиален, однозначен; затем вводятся три правила — операторы подстановки, рекурсии и минимизации, при помощи которых на основе базовых функций получаются более сложные рекурсивные функции.

Базовыми функциями и их сопутствующими алгоритмами могут выступать:

1) функция n независимых переменных, тождественно равная нулю. Тогда, если знаком функции является φ n, то независимо от количества аргументов значение функции следует положить равным нулю;
2) тождественная функция n независимых переменных вида Ψ ni. Тогда, если знаком функции является Ψ ni, то значением функции следует взять значение i-го аргумента, считая слева направо;
3) λ-функция одного независимого аргумента. Тогда, если знаком функции является λ, то значением функции следует взять значение, следующее за значением аргумента.

1б Если параметры — скалярные величины, принимающие непрерывный ряд значений, то полученная таким образом информация называется непрерывной (или *аналоговой*). Если же параметрам придать некоторый шаг изменений, то информация называется *дискретной*. Дискретная информация считается универсальной.

Дискретную информацию обычно отождествляют с *цифровой* информацией, которая является частным случаем символьной информации алфавитного представления. **Алфавит** — конечный набор символов любой природы. Очень часто в информатике возникает ситуация, когда символы одного алфавита надо представить символами другого, т. е. провести операцию *кодирования*.

Как показала практика, наиболее простым алфавитом, позволяющим кодировать другие алфавиты, является двоичный, состоящий из двух символов, которые обозначаются, как правило, через 0 и 1. С помощью n символов двоичного алфавита можно закодировать 2n символов, а этого достаточно, чтобы закодировать любой алфавит.

Величина, которая может быть представлена символом двоичного алфавита, называется минимальной единицей информации или **битом**. Последовательность из 8 бит — **байт**. Алфавит, содержащий 256 различных 8-битных последовательностей, называется *байтовым*.

Под **системой счисления** подразумевается набор правил наименования и записи чисел. Различают позиционные и непозиционные системы счисления.

Система счисления называется *позиционной*, если значение цифры числа зависит от местоположения цифры в числе. В противном случае она называется *непозиционной*. Значение числа определяется по положению этих цифр в числе.

4б 6. Function Round(X: Extended): LongInt; округляет значение вещественного типа до целого.

7. Function Trunc(X: Extended): LongInt; усекает значение вещественного типа до целого.

8. Procedure Val(S; var V; var Code: Integer); преобразовывает число из строкового значения S в числовое представление V.

Процедуры и функции работы с порядковыми величинами

1. Procedure Dec(var X [; N: LongInt]); вычитает единицу или N из переменной X.

2. Procedure Inc(var X [; N: LongInt]); прибавляет единицу или N к переменной X.

3. Function Odd(X: LongInt): Boolean; возвращает True, если X — нечетное число, и False — в противном случае.

4. Function Pred(X); возвращает предыдущее значение параметра.

5. Function Succ(X); возвращает следующее значение параметра.

3б 3) каждая операция или функция требуют аргументов фиксированного типа и выдают результат фиксированного типа.

В языке Pascal существуют скалярные и структурированные типы данных. К скалярным типам относятся стандартные типы и типы, определяемые пользователем. Стандартные типы включают целые, действительные, символьный, логические и адресный типы.

Целые типы определяют константы, переменные и функции, значения которых реализуются множеством целых чисел, допустимых в данной ЭВМ.

В языке Pascal принят следующий приоритет операций:

Тип	Диапазон значений	Требуемая память (байт)
Byte	0...255	1
Word	0...65535	2
Shortint	–128...127	1
Integer	–32768...32767	2
Longint	–2147483648...2147483647	4

Тип	Диапазон значений	Кол-во цифр мантиссы	Требуемая память (байт)
Real	2.9e – 39...1.7e + 38	11—12	6
Single	1.5e – 45...3.4e + 38	7—8	4
Double	5.0e – 324...1.7e + 308	15—16	8
Extended	3.4e – 4932...1.1e + 4932	19—20	10
Comp	–9.2e + 18...9.2e + 18	19—20	2

1) вычисления в круглых скобках;
2) вычисления значений функций;
3) унарные операции;
4) операции * / div mod and;
5) операции + — or xor;
6) операции отношения = <> < > <= >=.

5а 5. Операторы языка Pascal

Условный оператор

Формат полного условного оператора определяется следующим образом:

If B then S1 else S2;

где *B* — условие разветвления (принятия решения), логическое выражение или отношение;

S1, *S2* — один выполняемый оператор, простой или составной.

При выполнении условного оператора сначала вычисляется выражение *B*, затем анализируется его результат: если *B* — истинно, то выполняется оператор *S1* — ветвь *then*, а оператор *S2* пропускается; если *B* — ложно, то выполняется оператор *S2* — ветвь *else*, а оператор *S1* — пропускается.

Оператор выбора

Структура оператора имеет следующий вид:

```
case S of
c1: insruction1;
c2: insruction2;
...
cn: insructionN;
else instruction
end;
```

где *S* — выражение порядкового типа, значение которого вычисляется;

*c*1, *c*2,..., *cn* — константы порядкового типа, с которыми сравниваются выражения *S*; *instruction*1,..., *instructionN* — операторы, из которых выполняется тот, с константой которого совпадает значение выражения *S*;

instruction — оператор, который выполняется, если значение выражения *S* не совпадает ни с одной из констант *c*1, *c*2, ... *cn*.

Оператор цикла с параметром

Когда начинает выполняться оператор *for*, начальное и конечное значения определяются один раз,

6а 6. Понятие вспомогательного алгоритма

Алгоритм решения задачи проектируется путем декомпозиции всей задачи в отдельные подзадачи. Обычно подзадачи реализуются в виде подпрограмм.

Подпрограмма — это некоторый вспомогательный алгоритм, многократно использующийся в основном алгоритме с различными значениями некоторых входящих величин, называемых параметрами.

Подпрограмма в языках программирования — это последовательность операторов, которые определены и записаны только в одном месте программы, однако их можно вызвать для выполнения из одной или нескольких точек программы. Каждая подпрограмма определяется уникальным именем.

В языке Pascal существуют два типа подпрограмм — процедуры и функции. Процедура и функция — это именованная последовательность описаний и операторов. При использовании процедур или функций программа должна содержать текст процедуры или функции и обращение к процедуре или функции. Параметры, указанные в описании, называются формальными, указанные в обращении подпрограммы — фактическими. Все формальные параметры можно разбить на следующие категории:

1) параметры-переменные;
2) параметры-константы;
3) параметры-значения;
4) параметры-процедуры и параметры-функции, т. е. параметры процедурного типа;
5) нетипизированные параметры-переменные.

Тексты процедур и функций помещаются в раздел описаний процедур и функций.

7а 7. Процедуры и фукции в Pascal

Процедуры в Pascal

Описание процедуры состоит из заголовка и блока, который, за исключением раздела подключения модулей, не отличаются от блока программы. Заголовок состоит из ключевого слова *Procedure*, имени процедуры и необязательного списка формальных параметров в круглых скобках:

Procedure <имя> [(<список формальных параметров>)];

Для каждого формального параметра должен быть определен его тип. Группы параметров в описании процедуры разделяются точкой с запятой.

По структуре процедура почти полностью аналогична программе. Однако в блоке процедуре отсутствует раздел подключения модулей. Блок состоит из двух частей: описательной и исполнительной. В описательной части содержится описание элементов процедуры. А в исполнительной части указываются действия с доступными процедуре элементами программы (например, глобальные переменные и константы), позволяющие получить требуемый результат. Раздел инструкций процедуры отличается от раздела инструкций программы только тем, что после ключевого слова *End*, завершающего этот раздел, ставится точка с запятой, а не точка.

Для обращения к процедуре используется инструкция вызова процедуры. Она состоит из имени процедуры и списка аргументов, заключенного в круглые скобки. Операторы, которые должны выполняться при запуске процедуры, содержатся в операторной части модуля процедуры.

Иногда требуется, чтобы процедура вызывала сама себя. Такой способ вызова называется рекурсией. Рекурсия полезна в случаях, когда основную задачу можно разбить на подзадачи, каждая из которых реализуется по алгоритму, совпадающему с основным.

Функции в Pascal

Описание функции определяет часть программы, в которой вычисляется и возвращается значение. Описание функции состоит

8а 8. Опережающие описания и подключение подпрограмм. Директива

В программе может содержаться несколько подпрограмм, т. е. структура программы может быть усложнена. Однако эти подпрограммы могут располагаться на одном уровне вложенности, поэтому сначала должно идти описание подпрограммы, а затем обращение к ней, если только не используется специальное опережающее описание.

Описание процедуры, содержащее вместо блока операторов директиву *forward*, называется опережающим описанием. В каком-либо месте после этого описания с помощью определяющего описания процедура должна определяться. Определяющее описание — это описание, в котором используется тот же идентификатор процедуры, но опущен список формальных параметров, и в которое включен блок операторов. Описание *forward* и определяющее описание должны присутствовать в одной и той же части описания процедуры и функции. Между ними могут описываться другие процедуры и функции, которые могут обращаться к процедуре с опережающим описанием. Таким образом, возможна взаимная рекурсия.

Опережающее описание и определяющее описание представляют собой полное описание процедуры. Процедура считается описанной с помощью опережающего описания.

Если в программе будет содержаться довольно много подпрограмм, то программа перестанет быть наглядной, в ней будет тяжело ориентироваться. Во избежание этого некоторые подпрограммы хранят в виде исходных файлов на диске, а при необходимости они подключаются к основной программе

6б **Передача имен процедур и функций в качестве параметров**

Во многих задачах, особенно в задачах вычислительной математики, необходимо передавать имена процедур и функций в качестве параметров. Для этого в TURBO PASCAL введен новый тип данных — процедурный, или функциональный, в зависимости от того, что описывается. (Описание процедурных и функциональных типов приводится в разделе описания типов.)

Функциональный и процедурный тип определяется как заголовок процедуры и функции со списком формальных параметров, но без имени. Можно определить функциональный, или процедурный тип без параметров, например:

```
type
Proc = Procedure;
```

После объявления процедурного, или функционального, типа его можно использовать для описания формальных параметров — имен процедур и функций. Кроме того, необходимо написать те реальные процедуры или функции, имена которых будут передаваться как фактические параметры.

5б и эти значения сохраняются на протяжении всего выполнения оператора *for*. Оператор, который содержится в теле оператора *for*, выполняется один раз для каждого значения в диапазоне между начальным и конечным значением. Счетчик цикла всегда инициализируется начальным значением.

Оператор цикла с предусловием

While B do S;

где *B* — логическое условие, истинность которого проверяется (оно является условием завершения цикла)$;
S — тело цикла — один оператор.

Выражение, с помощью которого осуществляется управление повторением оператора, должно иметь логический тип. Вычисление его производится до того, как внутренний оператор будет выполнен. Внутренний оператор выполняется повторно до тех пор, пока выражение принимает значение True. Если выражение с самого начала принимает значение False, то оператор, содержащийся внутри оператора цикла с предусловием, не выполняется.

Оператор цикла с постусловием

repeat S until B;

где *B* — логическое условие, истинность которого проверяется (оно является условием завершения цикла);
S — один или более операторов тела цикла.

Результат выражения должен быть логического типа. Операторы, заключенные между ключевыми словами *repeat* и *until*, выполняются последовательно до тех пор, пока результат выражения не примет значение True. Последовательность операторов выполнится, по крайней мере, один раз, поскольку вычисление выражения производится после каждого выполнения последовательности операторов.

8б на этапе компиляции при помощи **директивы компиляции**.

Директива — это специальный комментарий, который может быть размещен в любом месте программы, там, где может находиться и обычный комментарий. Однако они различаются тем, что у директивы имеется специальная форма записи: сразу после закрывающей скобки без пробела записывается знак $, а затем, опять же без пробела, указывается директива.

Пример:

1) {$E+} — эмулировать математический сопроцессор;
2) {$F+} — формировать дальний тип вызова процедур и функций;
3) {$N+} — использовать математический сопроцессор;
4) {$R+} — проверять выход за границы диапазонов.

Некоторые ключи компиляции могут содержать параметр, например:

{$I имя файла} — включить в текст компилируемой программы названный файл.

7б из заголовка и блока. Заголовок содержит ключевое слово Function, имя функции, необязательный список формальных параметров, заключенный в круглые скобки, и тип возвращаемого функцией значения.

Общий вид заголовка функции следующий:

Function <имя> [(<список формальных параметров>)]: <тип возвращаемого результата>;

В реализации *Turbo Pascal 7.0* фирмы *Borland* возвращаемое функцией значение не может иметь составной тип. А язык *Object Pascal*, используемый в интегрированных средах разработки *Borland Delphi*, допускает любой тип возвращаемого результата, кроме файлового типа.

Блок функции представляет собой локальный блок, по структуре аналогичный блоку процедуры. В теле функции должна быть хотя бы одна инструкция присваивания, в левой части которой стоит имя функции. Именно она и определяет значение, возвращаемое функцией. Если таких инструкций несколько, то результатом функции будет значение последней выполненной инструкции присваивания.

Функция активизируется при вызове функции. При вызове функции указывается идентификатор функции и какие-либо параметры, необходимые для вычисления функции. Вызов функции может включаться в выражения в качестве операнда. Когда выражение вычисляется, функция выполняется и значением операнда становится значение, возвращаемое функцией.

В операторной части блока функции задаются операторы, которые должны выполняться при активизации функции. В модуле должен содержаться, по крайней мере, один оператор присваивания, в котором идентификатору функции присваивается значение. Результатом функции является последнее присвоенное значение. Если такой оператор присваивания отсутствует или он не был выполнен, то значение, возвращаемое функцией, не определено.

Если идентификатор функции используется при вызове функции внутри модуля - функции, то функция выполняется рекурсивно.

9a

9. Параметры подпрограмм

В описании процедуры или функции задается список формальных параметров. Каждый параметр, описанный в списке формальных параметров, является локальным по отношению к описываемой процедуре или функции, и в модуле, связанным с данной процедурой или функцией, на него можно ссылаться по его идентификатору.

Существует три типа параметров: значение, переменная и нетипизированная переменная. Они характеризуются следующим:

1. Группа параметров без предшествующего ключевого слова является списком параметров-значений.
2. Группа параметров, перед которыми следует ключевое слово *const* и за которыми следует тип, является списком параметров-констант.
3. Группа параметров, перед которыми стоит ключевое слово *var* и за которыми следует тип, является списком параметров-переменных.

Параметры-значения

Формальный параметр-значение обрабатывается, как локальная по отношению к процедуре или функции переменная, за исключением того, что он получает свое начальное значение из соответствующего фактического параметра при активизации процедуры или функции. Изменения, которые претерпевает формальный параметр-значение, не влияют на значение фактического параметра. Соответствующее фактическое значение параметра-значения должно быть выражением, и его значение не должно иметь файловый тип или какой-либо структурный тип, содержащий в себе файловый тип.

Фактический параметр должен иметь тип, совместимый по присваиванию с типом формального параметра-значения. Если параметр имеет строковый тип, то формальный параметр будет иметь атрибут размера, равный 255.

10a

10. Типы параметров подпрограмм

Параметры-значения

Формальный параметр-значение обрабатывается как локальная переменная, он получает свое начальное значение из соответствующего фактического параметра при активизации процедуры или функции. Изменения, которые претерпевает формальный параметр-значение, не влияют на значение фактического параметра. Соответствующее фактическое значение параметра-значения должно быть выражением, и его значение не должно иметь файловый тип.

Параметры-константы

Формальные параметры-константы получают свое значение при активизации процедуры или функции. Присваивания формальному параметру-константе не допускаются. Формальный параметр-константа не может передаваться в качестве фактического параметра другой процедуре или функции.

Параметры-переменные

Параметр-переменная используется, когда значение должно передаваться из процедуры или функции вызывающей программе. При активизации формальный параметр-переменная замещается фактической переменной, изменения формального параметра-переменной отражаются на фактическом параметре.

Нетипизированные параметры

Когда формальный параметр является нетипизированным параметром-переменной, то соответствующий фактический параметр может представлять собой ссылку на переменную или константу. Нетипизированный параметр, описанный с ключевым словом var, может модифицироваться, а нетипизированный параметр, описанный с ключевым словом const, доступен только по чтению.

11a

11. Строковый тип в Pascal. Процедуры и функции для переменных строкового типа

Последовательность символов определенной длины называется строкой. Переменные строкового типа определяются путем указания имени переменной, зарезервированного слова string, и возможно, но не обязательно указания максимального размера, т. е. длины строки, в квадратных скобках. Если не задавать максимальный размер строки, то по умолчанию он будет равен 255, т. е. строка будет состоять из 255 символов.

К каждому элементу строки можно обратиться по его номеру. Однако ввод и вывод строк осуществляются целиком, а не поэлементно, как это происходит в массивах. Число введенных символов не должно превышать указанного в максимальном размере строки, так если такое превышение будет иметь место, то «лишние» символы будут проигнорированы.

Процедуры и функции для переменных строкового типа

1. Function Copy(S: String; Index, Count: Integer): String;

Возвращает подстроку строки. S — выражение типа String. Index и Count — выражения целого типа. Функция возвращает строку, содержащую Count символов, начинающихся с позиции Index. Если Index больше, чем длина S, функция возвращает пустую строку.

2. Procedure Delete(var S: String; Index, Count: Integer);

Удаляет подстроку символов длиной Count из строки S, начиная с позиции Index. S — переменная типа String. Index и Count — выражения целого типа. Если Index больше, чем длина S, символы не удаляются.

12a

12. Записи

Запись представляет собой совокупность ограниченного числа логически связанных компонент, принадлежащих к разным типам. Компоненты записи называются полями, каждое из которых определяется именем. Поле записи содержит имя поля, вслед за которым через двоеточие указывается тип этого поля. Поля записи могут относиться к любому типу, допустимому в языке Pascal, за исключением файлового типа.

Описание записи в языке Pascal осуществляется с помощью служебного слова RECORD, вслед за которым описываются компоненты записи. Завершается описание записи служебным словом END.

Например, записная книжка содержит фамилии, инициалы и номера телефона, поэтому отдельную строку в записной книжке удобно представить в виде следующей записи:

```
type Row = Record
 FIO: String[20];
 TEL: String[7];
end;
var str: Row;
```

Описание записей возможно и без использования имени типа, например:

```
var str: Record
 FIO: String[20];
 TEL: String[7];
end;
```

10б **Процедурные переменные**

После определения процедурного типа появляется возможность описывать переменные этого типа. Такие переменные называют процедурными переменными. Процедурной переменной можно присвоить значение процедурного типа.

Процедура или функция при присваивании должна быть:

1) не стандартной;
2) не вложенной;
3) не процедурой типа *inline*;
4) не процедурой прерывания (*interrupt*).

Параметры процедурного типа

Поскольку процедурные типы допускается использовать в любом контексте, то можно описывать процедуры или функции, которые воспринимают процедуры и функции в качестве параметров. Параметры процедурного типа особенно полезны в том случае, когда над множеством процедур или функций нужно выполнить какие-то общие действия.

Если процедура или функция должны передаваться в качестве параметра, они должны удовлетворять тем же правилам совместимости типа, что и при присваивании. То есть, такие процедуры или функции должны компилироваться с директивой *far*, они не могут быть встроенными функциями, не могут быть вложенными и не могут описываться с атрибутами *inline* или *interrupt*.

9б **Параметры-константы**

В теле подпрограммы значение параметра-константы изменить нельзя. Параметрами-константами можно оформить те параметры, изменения которых в подпрограмме нежелательно и должно быть запрещено.

Параметры-переменные

Параметр-переменная используется в случаях, когда значение должно быть передано из подпрограммы в вызывающий блок. В этом случае при вызове подпрограммы формальный параметр замещается аргументом-переменной, и любые изменения формального параметра отражаются на аргументе.

Процедурные переменные

После определения процедурного типа появляется возможность описывать переменные этого типа. Такие переменные называют процедурными переменными. Как и целая переменная, которой можно присвоить значение целого типа, процедурной переменной можно присвоить значение процедурного типа. Таким значением может быть, конечно, другая процедурная переменная, но оно может также представлять собой идентификатор процедуры или функции. В таком контексте описания процедуры или функции можно рассматривать как описание особого рода константы, значением которой является процедура или функция.

Как и при любом другом присваивании, значения переменной в левой и в правой части должны быть совместимы по присваиванию. Процедурные типы, чтобы они были совместимы по присваиванию, должны иметь одно и то же число параметров, а параметры на соответствующих позициях должны быть одинакового типа. Имена параметров в описании процедурного типа никакого действия не вызывают.

Кроме того, для обеспечения совместимости по присваиванию, процедура или функция, если ее нужно присвоить процедурной переменной, не должна быть стандартной или вложенной.

12б Обращение к записи в целом допускается только в операторах присваивания, где слева и справа от знака присваивания используются имена записей одинакового типа. Во всех остальных случаях оперируют отдельными полями записей. Чтобы обратиться к отдельной компоненте записи, необходимо задать имя записи и через точку указать имя нужного поля. Такое имя называется составным. Компонентой записи может быть также запись, в таком случае составное имя будет содержать не два, а большее количество имен.

Обращение к компонентам записей можно упростить, если воспользоваться оператором присоединения with. Он позволяет заменить составные имена, характеризующие каждое поле, просто на имена полей, а имя записи определить в операторе присоединения.

Иногда содержимое отдельной записи зависит от значения одного из ее полей. В языке Pascal допускается описание записи, состоящей из общей и вариантной частей. Вариантная часть задается с помощью *конструкции case P of, где P — имя поля из общей части записи*. Возможные значения, принимаемые этим полем, перечисляются так же, как и в операторе варианта. Однако вместо указания выполняемого действия, как это делается в операторе варианта, указываются поля варианта, заключенные в круглые скобки. Описание вариантной части завершается служебным словом end. Тип поля P можно указать в заголовке вариантной части. Инициализация записей осуществляется с помощью типизированных констант.

11б 3. Procedure Insert(Source: String; var S: String; Index: Integer);

Объединяет подстроку в строку, начиная с определенной позиции. Source — выражение типа String. S — переменная типа String любой длины. Index — выражение целочисленного типа. Insert вставляет Source в S, начиная с позиции S[Index].

4. Function Length(S: String): Integer;

Возвращает число символов, фактически используемое в строке S. Обратите внимание: при использовании строк с нуль-окончанием, число символов не обязательно равно числу байтов.

5. Function Pos(Substr: String; S: String): Integer;

Ищет подстроку в строке. Pos ищет Substr внутри S и возвращает целочисленное значение, которое является индексом первого символа Substr внутри S. Если Substr не найден, Pos возвращает нуль.

13a

13. Множества

Понятие множества в языке Pascal основывается на математическом представлении о множествах: это ограниченная совокупность различных элементов. Для построения конкретного множественного типа используется перечисляемый или интервальный тип данных. Тип элементов, составляющих множество, называется базовым типом.

Множественный тип описывается с помощью служебных слов Set of, например:

```
type M = Set of B;
```

здесь M — множественный тип, B — базовый тип.

Принадлежность переменных к множественному типу может быть определена прямо в разделе описания переменных.

Константы множественного типа записываются в виде заключенной в квадратные скобки последовательности элементов или интервалов базового типа, разделенных запятыми.

К переменным и константам множественного типа применимы операции присваивания (:=), объединения (+), пересечения (*) и вычитания (–). Результат выполнения этих операций есть величина множественного типа:

```
1) ['A','B'] + ['A','D']      даст      ['A','B','D'];
2) ['A'] * ['A','B','C']      даст      ['A'];
3) ['A','B','C'] — ['A','B']  даст      ['C'].
```

К множественным величинам применимы операции: тождественность (=), нетождественность (<>), содержится в (<=), содержит (>=). Результат выполнения этих операций имеет логический тип:

```
1) ['A','B'] = ['A','C']      даст      FALSE ;
2) ['A','B'] <> ['A','C']     даст      TRUE;
```

14a

14. Файлы. Операции с файлами

Файловый тип данных определяет упорядоченную совокупность однотипных компонент.

При работе с файлами выполняются операции ввода-вывода. Операция ввода — это перепись данных с внешнего устройства в память, операция вывода — пересылка данных из памяти на внешнее устройство.

Текстовые файлы

Для описания таких файлов имеется тип Text:

```
var TF1, TF2: Text;
```

Компонентные файлы

Компонентный, или типизированный файл, — это файл с объявленным типом его компонент.

```
type M = File Of T;
```

где M — имя файлового типа;
T — тип компоненты.

Операции производятся с помощью процедур.

Write(f, X_1, X_2,... XK)

Бестиповые файлы

Бестиповые файлы позволяют записывать на диск произвольные участки памяти ЭВМ и считывать их.

```
var f: File;
```

1. Procedure Assign(var F; FileName: String);
Она сопоставляет имя файла с переменной.
2. Procedure Close(var F);
Она разрывает связь между файловой переменной и внешним дисковым файлом и закрывает файл.
3. Function Eof(var F): Boolean;
{Типизированные или нетипизированные файлы}
Function Eof[(var F: Text)]: Boolean;
{Текстовые файлы}
Проверяет на конец файла.

15a

15. Модули. Виды модулей

Модуль(UNIT) в Pascal — это особым образом оформленная библиотека подпрограмм. Модуль, в отличие от программы, не может быть запущен на выполнение самостоятельно, он может только участвовать в построении программ и других модулей.

Модуль в Pascal представляет собой отдельно хранимую и независимо компилируемую программную единицу.

Все программные элементы модуля можно разбить на две части:

1) программные элементы, предназначенные для использования другими программами или модулями, такие элементы называют видимыми вне модуля;
2) программные элементы, необходимые только для работы самого модуля, их называют невидимыми (или скрытыми).

```
unit <имя модуля>; {заголовок модуля}
interface
{описание видимых программных элементов модуля}
implementation
{описание скрытых программных элементов модуля}
begin
{операторы инициализации элементов модуля}
end.
```

Для обращения к переменной, описанной в модуле, необходимо применить составное имя, состоящее из имени модуля и имени переменной, разделенных точкой.

Рекурсивное использование модулей запрещено.

Перечислим, какие бывают виды модулей.

16a

16. Ссылочный тип данных. Динамическая память. Динамические переменные. Работа с динамической памятью

Статической переменной (статически размещенной) называется описанная явным образом в программе переменная, обращение к ней осуществляется по имени. Место в памяти для размещения статических переменных определяется при компиляции программы. В отличие от таких статических переменных в программах, написанных на языке Pascal, могут быть созданы динамические переменные. Основное свойство динамических переменных заключается в том, что они создаются, и память для них выделяется во время выполнения программы.

Размещаются динамические переменные в динамической области памяти (heap-области). Динамическая переменная не указывается явно в описаниях переменных, и к ней нельзя обратиться по имени. Доступ к таким переменным осуществляется с помощью указателей и ссылок.

Ссылочный тип (указатель) определяет множество значений, которые указывают на динамические переменные определенного типа, называемого базовым типом. Переменная ссылочного типа содержит адрес динамической переменной в памяти. Если базовый тип является еще не описанным идентификатором, то он должен быть описан в той же самой части описания типов, что и тип-указатель.

Зарезервированное слово nil обозначает константу со значением указателя, которая ни на что не указывает.

Приведем пример описания динамических переменных.

```
var p1, p2 : ^real;
  p3, p4 : ^integer;
        ...
```

14б 4. Procedure Erase(var F);
Удаляет внешний файл, связанный с F.
5. Function FileSize(var F): Integer;
Возвращает размер в байтах файла F.
6. Function FilePos(var F): LongInt;
Возвращает текущую позицию внутри файла.
7. Procedure Reset(var F [: File; RecSize: Word]);
Открывает существующий файл.
8. Procedure Rewrite(var F: File [; Recsize: Word]);
Создает и открывает новый файл.
9. Procedure Seek(var F; N: LongInt);
Перемещает текущую позицию файла к определенному компоненту.
10. Procedure Append(var F: Text);
Дозапись.
11. Function Eoln[(var F: Text)]: Boolean;
Проверяет на конец строки.
12. Procedure Read(F, V1 [, V2, ..., Vn]);
{Типизированные и нетипизированные файлы}
Procedure Read([var F: Text;] V1 [, V2, ..., Vn]);
{Текстовые файлы}
Читает компонент файла в переменную.
13. Procedure Readln([var F: Text;] V1 [, V2, ..., Vn]);
Считывает строку символов в файле, включая маркер конца строки, и переходит к началу следующей.
14. Function SeekEof[(var F: Text)]: Boolean;
Возвращает признак конца файла. Используется только для открытых текстовых файлов.
15. Procedure Writeln([var F: Text;] [P1, P2, ..., Pn]);
{Текстовые файлы}
Выполняет операцию Write, затем помещает метку конца строки в файл.

13б 3) ['B'] <= ['B','C'] даст TRUE;
4) ['C','D'] >= ['A'] даст FALSE.

Кроме этих операций, для работы с величинами множественного типа используется операция *in*, проверяющая принадлежность элемента базового типа, стоящего слева от знака операции, множеству, стоящему справа от знака операции. Результат выполнения этой операции — булевский.

Величины множественного типа не могут быть элементами списка ввода-вывода. В каждой конкретной реализации транслятора с языка Pascal количество элементов базового типа, на котором строится множество, ограниченно.

16б **Процедуры и функции работы с динамической памятью**

1. Процедура *New*(*var p:Pointer*).
Выделяет место в динамической области памяти для размещения динамической переменной *p^*, и ее адрес присваивает указателю p.
2. Процедура *Dispose*(*var p:Pointer*).
Освобождает участок памяти, выделенный для размещения динамической переменной процедурой New, и значение указателя p становится неопределенным.
3. Процедура *GetMem*(*var p:Pointer*; *size:Word*).
Выделяет участок памяти в *heap*-области, присваивает адрес его начала указателю p, размер участка в байтах задается параметром *size*.
4. Процедура *FreeMem*(*var p:Pointer*; *size:Word*).
Освобождает участок памяти, адрес начала которого определен указателем *p*, а размер — параметром *size*. Значение указателя *p* становится неопределенным.
5. Процедура *Mark*(*var p*:*Pointer*) записывает в указатель *p* адрес начала участка свободной динамической памяти на момент ее вызова.
6. Процедура *Release*(*var p:Pointer*) освобождает участок динамической памяти, начиная с адреса, записанного в указатель *p* процедурой *Mark*, т. е. очищает ту динамическую память, которая была занята после вызова процедуры *Mark*.
7. Функция *MaxAvail*:*Longint* возвращает длину в байтах самого длинного свободного участка динамической памяти.
8. Функция *MemAvail*:*Longint* возвращает полный объем свободной динамической памяти в байтах.
9. Вспомогательная функция *SizeOf(X)*:*Word* возвращает объем в байтах, занимаемый *X*, причем *X* может быть либо именем переменной любого типа, либо именем типа.

15б 1. Модуль SYSTEM.
Модуль SYSTEM реализует поддерживающие подпрограммы нижнего уровня для всех встроенных средств, таких как ввод-вывод, работа со строками, операции с плавающей точкой и динамическое распределение памяти.
2. Модуль DOS.
Модуль Dos реализует многочисленные процедуры и функции Pascal, которые эквивалентны наиболее часто используемым вызовам DOS, как, например, GetTime, SetTime, DiskSize и так далее.
3. Модуль CRT.
Модуль CRT реализует ряд мощных программ, предоставляющих полную возможность управления средствами компьютера PC, такими, как управление режимом экрана, расширенные коды клавиатуры, цвета, окна и звуковые сигналы.
4. Модуль GRAPH.
С помощью процедур и функций, входящих в этот модуль, можно создавать различные графические изображения на экране.
5. Модуль OVERLAY.
Модуль OVERLAY позволяет уменьшить требования к памяти программы DOS реального режима.

17a 17. Абстрактные структуры данных

Структурированные типы данных, такие как массивы, множества, записи, представляют собой статические структуры, так как их размеры неизменны в течение всего времени выполнения программы.

Часто требуется, чтобы структуры данных меняли свои размеры в ходе решения задачи. Такие структуры данных называются динамическими. К ним относятся стеки, очереди, списки, деревья и др.

Описание динамических структур с помощью массивов, записей и файлов приводит к неэкономному использованию памяти ЭВМ и увеличивает время решения задач.

Каждая компонента любой динамической структуры представляет собой запись, содержащую, по крайней мере, два поля: одно поле типа «указатель», а второе — для размещения данных. В общем случае запись может содержать не один, а несколько указателей и несколько полей данных. Поле данных может быть переменной, массивом, множеством или записью.

Если в указующей части содержится адрес одного элемента списка, то список называется однонаправленным (или односвязным). Если же он содержит две компоненты, то двусвязным. Над списками можно проводить различные операции, например:

1) добавление элемента к списку;
2) удаление элемента из списка с заданным ключом;
3) поиск элемента с заданным значением ключевого поля;
4) сортировка элементов списка;
5) деление списка на два и более списков;
6) объединение двух и более списков в один;
7) другие операции.

18a 18. Стеки

Стеком называется динамическая структура данных, добавление компоненты в которую и исключение компоненты из которой производится из одного конца, называемого вершиной стека. Стек работает по принципу *LIFO*(*Last-In*, *First-Out*) — «Поступивший последним, обслуживается первым».

Обычно над стеками выполняется три операции:

1) начальное формирование стека (запись первой компоненты);
2) добавление компоненты в стек;
3) выборка компоненты (удаление).

Для формирования стека и работы с ним необходимо иметь две переменные типа «указатель», первая из которых определяет вершину стека, а вторая — вспомогательная.

Пример. Составить программу, которая формирует стек, добавляет в него произвольное количество компонент, а затем читает все компоненты.

```
Program STACK;
uses Crt;
type
 Alfa = String[10];
 PComp = ^Comp;
 Comp = Record
 sD : Alfa;
 pNext : PComp
end;
var
 pTop : PComp;
 sC : Alfa;

Procedure CreateStack(var pTop : PComp; var sC :
Alfa);
```

19a 19. Очереди

Очередью называется динамическая структура данных, добавление компоненты в которую производится в один конец, а выборка осуществляется с другого конца. Очередь работает по принципу *FIFO* (*First-In*, *First-Out*) — «Поступивший первым, обслуживается первым».

Пример. Составить программу, которая формирует очередь, добавляет в нее произвольное количество компонент, а затем читает все компоненты.

```
Program QUEUE;
uses Crt;
type
 Alfa = String[10];
 PComp = ^Comp;
 Comp = record
 sD : Alfa;
 pNext : PComp;
end;
var
 pBegin, pEnd : PComp;
 sC : Alfa;

Procedure CreateQueue(var pBegin,pEnd:PComp; var
sC:Alfa);
 begin
  New(pBegin);
  pBegin^.pNext := NIL;
  pBegin^.sD := sC;
  pEnd := pBegin;
 end;
```

20a 20. Древовидные структуры данных

Древовидной структурой данных называется конечное множество элементов-узлов, между которыми существуют отношения — связь исходного и порожденного.

Если использовать рекурсивное определение, предложенное *Н*. Виртом, то древовидная структура данных с базовым типом *t* — это либо пустая структура, либо узел типа *t*, с которым связано конечное множество древовидных структур с базовым типом *t*, называемых поддеревьями.

Далее дадим определения, используемые при оперировании древовидными структурами.

Если узел *y* находится непосредственно под узлом *x*, то узел *y* называется непосредственным потомком узла *x*, а *x* — непосредственным предком узла *y*, т. е., если узел *x* находится на *i*-ом уровне, то соответственно узел *y* находится на (*i* + 1)-ом уровне.

Максимальный уровень узла дерева называется высотой или глубиной дерева. Предка не имеет только один узел дерева — его корень.

Узлы дерева, у которых не имеется потомков, называются терминальными узлами (или листами дерева). Все остальные узлы называются внутренними узлами. Количество непосредственных потомков узла определяет степень этого узла, а максимально возможная степень узла в данном дереве определяет степень дерева.

Предков и потомков нельзя поменять местами, т. е. связь исходного и порожденного действует только в одном направлении.

Если пройти от корня дерева к некоторому конкретному узлу, то количество ветвей дерева, которое при

18б
```
begin
 New(pTop);
pTop^.pNext := NIL;
pTop^.sD := sC;
end;
Procedure AddComp(var pTop : PComp; var sC : Alfa);
var pAux : PComp;
begin
 NEW(pAux);
 pAux^.pNext := pTop;
 pTop := pAux;
 pTop^.sD := sC;
end;

Procedure DelComp(var pTop : PComp; var sC : ALFA);
begin
 sC := pTop^.sD;
 pTop := pTop^.pNext;
end;

begin
 Clrscr;
 writeln(' ВВЕДИ СТРОКУ ');
 readln(sC);
 CreateStack(pTop, sC);
 repeat
 writeln(' ВВЕДИ СТРОКУ ');
 readln(sC);
 AddComp(pTop, sC);
 until sC = 'END';
```

17б Однако, как правило, необходимости во всех операциях при решении различных задач не возникает. Поэтому в зависимости от основных операций, которые необходимо применить, существуют различные виды списков. Наиболее популярные из них — это стек и очередь.

20б этом будет пройдено, называется длиной пути для этого узла. Если все ветви (узлы) у дерева упорядочены, то дерево называется упорядоченным.

Частным случаем древовидных структур являются бинарные деревья. Это деревья, в которых каждый потомок имеет не более двух потомков, называемых левым и правым поддеревьями. Таким образом, бинарное дерево — это древовидная структура, степень которой равна двум.

Упорядоченность бинарного дерева определяется по следующему правилу: каждому узлу соответствует свое ключевое поле, и для каждого узла значение ключа больше всех ключей в его левом поддереве и меньше всех ключей в его правом поддереве.

Дерево, степень которого больше двух, называется сильноветвящимся.

19б
```
Procedure AddQueue(var pEnd : PComp; var sC :
Alfa);
var pAux : PComp;
begin
 New(pAux);
 pAux^.pNext := NIL;
 pEnd^.pNext := pAux;
 pEnd := pAux;
 pEnd^.sD := sC;
end;

Procedure DelQueue(var pBegin : PComp; var sC :
Alfa);
begin
 sC := pBegin^.sD;
 pBegin := pBegin^.pNext;
end;

begin
 Clrscr;
 writeln(' ВВЕДИ СТРОКУ ');
 readln(sC);
 CreateQueue(pBegin, pEnd, sC);

 repeat
 writeln(' ВВЕДИ СТРОКУ ');
 readln(sC);
 AddQueue(pEnd, sC);
 until sC = 'END';
```

21a 21. Операции над деревьями

Далее будем рассматривать все операции применительно к бинарным деревьям.

I. Построение дерева.

Приведем алгоритм построения упорядоченного дерева.

1. Если дерево пусто, то данные переносятся в корень дерева. Если же дерево не пусто, то осуществляется спуск по одной из его ветвей таким образом, чтобы упорядоченность дерева не нарушалась. В результате новый узел становится очередным листом дерева.

2. Чтобы добавить узел в уже существующее дерево, можно воспользоваться вышеприведенным алгоритмом.

3. При удалении узла из дерева следует быть внимательным. Если удаляемый узел является листом, или же имеет только одного потомка, то операция проста. Если же удаляемый узел имеет двух потомков, то необходимо будет найти узел среди его потомков, который можно будет поставить на его место. Это нужно в силу требования упорядоченности дерева.

Можно поступить таким образом: поменять удаляемый узел местами с узлом, имеющем самое большое значение ключа в левом поддереве, или с узлом, имеющем самое малое значение ключа в правом поддереве, а затем удалить искомый узел как лист.

II. Поиск узла с заданным значением ключевого поля.

При осуществлении этой операции необходимо совершить обход дерева. Необходимо учитывать различные формы записи дерева: префиксную, инфиксную и постфиксную.

22a 22. Примеры реализации операций

1. Построить дерево из n узлов минимальной высоты, или идеально сбалансированное дерево (количество узлов левого и правого поддеревьев такого дерева должны отличаться не более чем на единицу).

Рекурсивный алгоритм построения:

1) первый узел берется в качестве корня дерева;
2) тем же способом строится левое поддерево из *nl* узлов;
3) тем же способом строится правое поддерево из *nr* узлов;

$$nr = n - nl - 1$$

В качестве информационного поля будем брать номера узлов, вводимые с клавиатуры. Рекурсивная функция, реализующая данное построение, будет выглядеть следующим образом:

```
Function Tree(n : Byte) : TreeLink;
Var t : TreeLink; nl,nr,x : Byte;
Begin
 If n = 0 then Tree := nil
 Else
 Begin
 nl := n div 2;
 nr = n – nl – 1;
 writeln('Введите номер вершины ');
 readln(x);
 new(t);
t^.inf := x;
t^.left := Tree(nl);
t^.right := Tree(nr);
Tree := t;
End;
{Tree}
End.
```

23a 23. Понятие графа. Способы представления графа

Граф — пара $G = (V,E)$, где V — множество объектов произвольной природы, называемых вершинами, а E — семейство пар $ei = (vi1, vi2)$, $vijOV$, называемых **ребрами**. В общем случае множество V и (или) семейство E могут содержать бесконечное число эле-ментов, но мы будем рассматривать только конечные графы, т. е. графы, у которых как V, так и E конечны. Если порядок элементов, входящих в ei, имеет значение, то граф называется **ориентированным**, сокращенно — орграф, иначе — **неориентированным**. Ребра орграфа называются **дугами**.

Если $e = <u,v>$, то вершины v и u называются концами ребра. При этом говорят, что ребро e является смежным (инцидентным) каждой из вершин v и u. Вершины v и u также называются **смежными (инцидентными)**. В общем случае допускаются ребра вида $e = <v, v>$; такие ребра называются **петлями**.

Степень вершины графа — это число ребер, инцидентных данной вершине, причем петли учитываются дважды.

Вес вершины — число (действительное, целое или рациональное), поставленное в соответствие данной вершине (интерпретируется как стоимость, пропускная способность и т. д.).

Путем в графе (или маршрутом в орграфе) называется чередующаяся последовательность вершин и ребер (или дуг — в орграфе) вида $v0, (v0,v1), v1, \ldots, (vn-1,vn), vn$. Число n называется длиной пути. Путь без повторяющихся ребер называется цепью, без повторяющихся вершин — простой цепью. Замкнутый путь без повторяющихся ребер называется **циклом** (или

24a 24. Различные представления графа

Для реализации графа в виде списка инцидентности можно использовать следующий тип:

```
Type List = ^S;
  S = record;
  inf : Byte;
  next : List;
  end;
```

Тогда граф задается следующим образом:

```
Var Gr : array[1..n] of List;
```

Теперь обратимся к процедуре обхода графа. Это вспомогательный алгоритм, который позволяет просмотреть все вершины графа, проанализировать все информационные поля. Если рассматривать обход графа в глубину, то существуют два типа алгоритмов: рекурсивный и нерекурсивный.

На языке Pascal процедура обхода в глубину будет выглядеть следующим образом:

```
Procedure Obhod(gr : Graph; k : Byte);
Var g : Graph; l : List;
Begin
nov[k] := false;
g := gr;
While g^.inf <> k do
g := g^.next;
l := g^.smeg;
While l <> nil do begin
If nov[l^.inf] then Obhod(gr, l^.inf);
l := l^.next;
End;
End;
```

22б 2. В бинарном упорядоченном дереве найти узел с заданным значением ключевого поля. Если такого элемента в дереве нет, то добавить его в дерево.

```
Procedure Search(x : Byte; var t : TreeLink);
Begin
If t = nil then
Begin
New(t);
t^inf := x;
t^.left := nil;
t^.right := nil;
End
Else if x < t^.inf then
Search(x, t^.left)
Else if x > t^.inf then
Search(x, t^.right)
Else
Begin
{обработка найденного элемента}
...
End;
End.
```

21б Возникает вопрос: каким образом представить узлы дерева, чтобы было наиболее удобно работать с ними? Можно представлять дерево с помощью массива, где каждый узел описывается величиной комбинированного типа, у которой информационное поле символьного типа и два поля ссылочного типа. Но это не совсем удобно, так как деревья имеют большое количество узлов, заранее не определенное. Поэтому лучше всего при описании дерева использовать динамические переменные. Тогда каждый узел представляется величиной одного типа, которая содержит описание заданного количества информационных полей, а количество соответствующих полей должно быть равно степени дерева. Логично отсутствие потомков определять ссылкой nil. Тогда на языке Pascal описание бинарного дерева может выглядеть следующим образом:

```
TYPE TreeLink = ^Tree;
Tree = record;
Inf : <тип данных>;
Left, Right : TreeLink;
End.
```

24б **Представление графа списком списков**

Граф можно определить с помощью списка списков следующим образом:

```
Type List = ^Tlist;
Tlist = record
inf : Byte;
next : List;
 end;
Graph = ^TGpaph;
TGpaph = record
inf : Byte;
smeg : List;
next : Graph;
end;
```

При обходе графа в ширину мы выбираем произвольную вершину и просматриваем сразу все вершины, смежные с ней.

Приведем процедуру обхода графа в ширину на псевдокоде:

```
Procedure Obhod2(v);
Begin
queue = O;
queue <= v;
nov[v] = False;
While queue <> O do
Begin
p <= queue;
For u in spisok(p) do
If nov[u] then
Begin
nov[u] := False;
queue <= u;
End;
End;
End;
```

23б контуром в орграфе); без повторяющихся вершин (кроме первой и последней) — простым циклом.

Граф называется связным, если существует путь между любыми двумя его вершинами, и несвязным — в противном случае.

Существуют различные способы представления графов.

1. Матрица инцидентности.

Это прямоугольная матрица размерности $n \times m$, где n — количество вершин, а m — количество ребер.

2. Матрица смежности.

Это квадратная матрица размерности $n \times n$, где n — количество вершин.

3. Список смежности (инцидентности).

Представляет собой структуру данных, которая для каждой вершины графа хранит список смежных с ней вершин. Список представляет собой массив указателей, i-ый элемент которого содержит указатель на список вершин, смежных с i-ой вершиной.

4. Список списков.

Представляет собой древовидную структуру данных, в которой одна ветвь содержит списки вершин, смежных для каждой.

25a

25. Объектный тип в Pascal. Понятие объекта, его описание и использование

Объектно-ориентированный язык программирования характеризуется тремя основными свойствами:

1) инкапсуляцией. Комбинирование записей с процедурами и функциями, манипулирующими полями этих записей, формирует новый тип данных — объект;
2) наследованием.Определение объекта и его дальнейшее использование для построения иерархии порожденных объектов с возможностью для каждого порожденного объекта, относящегося к иерархии, доступа к коду и данным всех порождающих объектов;
3) полиморфизмом. Присваивание действию одного имени, которое затем совместно используется вниз и вверх по иерархии объектов, причем каждый объект иерархии выполняет это действие способом, именно ему подходящим.

Говоря об объекте, мы вводим в рассмотрение новый тип данных — объектный. Объектный тип является структурой, состоящей из фиксированного числа компонентов. Каждый компонент является либо полем, содержащим данные строго определенного типа, либо методом, выполняющим операции над объектом.

Объектный тип может наследовать компоненты другого объектного типа. Если тип T_2 наследует от типа T_1, то тип T_2 является потомком типа T_1, а сам тип T_1 является родителем типа T_2.

Следующий исходный код приводит пример описания объектного типа.

26a

26. Наследование

Наследование — это процесс порождения новых типов-потомков от существующих типов-родителей, при этом потомок получает (наследует) от родителя все его поля и методы.

Тип-потомок, при этом, называется наследником или порожденным (дочерним) типом. А тип, которому наследует дочерний тип, называется порождающим (родительским) типом.

Наследуемые поля и методы можно использовать в неизменном виде или переопределять (модифицировать).

Н. Вирт в своем языке Паскаль стремился к максимальной простоте, поэтому он не стал его усложнять введением отношения наследования. Поэтому типы в Паскале не могут наследовать.

Однако *Turbo Pascal 7.0* расширяет этот язык для поддержки наследования. Одним из таких расширений является новая категория структуры данных, связанная с записями, но значительно более мощная. Типы данных в этой новой категории определяются с помощью нового зарезервированного слова *Object*. Синтаксис при этом очень похож на синтаксис определения записей:

```
Type
 <имя типа> = Object [(<имя типа родителя>)]
([<область действия>]
<описание полей и методов>)+
end;
```

Знак "+" после синтаксической конструкции в круглых скобках означает, что эта конструкция должна встречаться один или более раз в данном описании.

27a

27. Создание экземпляров объектов

Экземпляр объекта создается посредством описания переменной или константы объектного типа или путем применения стандартной процедуры New к переменной типа «указатель на объектный тип». Результирующий объект называется экземпляром объектного типа.

Если объектный тип содержит виртуальные методы, то экземпляры этого объектного типа должны инициализироваться посредством вызова конструктора перед вызовом любого виртуального метода.

Присваивание экземпляра объектного типа не подразумевает инициализации экземпляра. Объект инициализируется кодом, генерируемым компилятором, который выполняется между вызовом конструктора и моментом когда выполнение фактически достигает первого оператора в блоке кода конструктора.

Если экземпляр объекта не инициализируется и проверка диапазона включена (директивой {$*R*+}), то первый вызов виртуального метода экземпляра объекта дает ошибку этапа выполнения. Если проверка диапазона выключена (директивой {$*R*–}), то первый вызов виртуального метода неинициализированного объекта может привести к непредсказуемому поведению.

Правило обязательной инициализации применимо также к экземплярам, которые являются компонентами структурных типов. Например:

```
var
 Comment: array [1..5] of TStrField;
 I: integer;
begin
 for I := 1 to 5 do
```

28a

28. Компоненты и область действия

Область действия идентификатора компоненты простирается за пределы объектного типа. Более того, область действия идентификатора компонента простирается сквозь блоки процедур, функций, конструкторов и деструкторов, которые реализуют методы объектного типа и его наследников. Исходя из этих соображений, идентификатор компоненты должен быть уникальным внутри объектного типа и внутри всех его наследников, а также внутри всех его методов.

В описании объектного типа заголовок метода может задавать параметры описываемого объектного типа, даже если описание еще не полное.

Рассмотрим следующую схему описания типа, содержащего компоненты всех допустимых областей действия:

```
Type
 <имя типа> = Object [(<имя типа родителя>)]
Private
<частные описания полей и методов>
Protected
<защищенные описания полей и методов>
Public
<общедоступные описания полей и методов>
 end;
```

Поля и методы, описанные в разделе *Private*, могут быть использованы только внутри модуля, содержащего их описания и нигде более.

Защищенные поля и методы, то есть описанные в разделе *Protected*, видимы в модуле, где определяется тип, и потомкам данного типа.

26б Область действия есть одно из следующих ключевых слов:

— *Private*;
— *Protected*;
— *Public*.

Область действия характеризует, каким участкам программы будут доступны компоненты, описания которых следуют за ключевым словом, именующим данную область действия.

Подробнее об областях действия компонент рассказано в вопросе № 28.

Наследование — это мощный инструмент, используемый при разработке программ. Оно позволяет реализовать на практике объектно-ориентированную декомпозицию задачи, средствами языка выражать отношения между объектами типов, образующих иерархию, а также способствует повторному использованию программного кода.

25б

```
type
  Point = object
 X, Y: integer;
 end;

 Rect = object
 A , B: TPoint;
 procedure Init(XA, YA, XB, YB: Integer);
 procedure Copy(var R: TRectangle);
 procedure Move(DX, DY: Integer);
 procedure Grow(DX, DY: Integer);
 procedure Intersect(var R: TRectangle);
 procedure Union(var R: TRectangle);
 function Contains(P: Point): Boolean;
 end;
```

В отличие от других типов объектные типы могут описываться только в разделе описаний типов, находящемся на самом внешнем уровне области действия программы или модуля. Таким образом, объектные типы не могут описываться в разделе описаний переменных или внутри блока процедуры, функции или метода.

Тип компоненты файлового типа не может иметь объектный тип или любой структурный тип, содержащий компоненты объектного типа.

28б Поля и методы из раздела Public не имеют ограничений на использование и могут быть задействованы в любом месте программы, которое имеет доступ к объекту данного типа.

Область действия идентификатора компонента, описанного в части *private* описания типа, ограничивается модулем (программой), которая содержит описание объектного типа. Другими словами, частные (*private*) компоненты-идентификаторы действуют, как обычные общедоступные идентификаторы в рамках модуля, который содержит описание объектного типа, а вне модуля любые частные компоненты и идентификаторы неизвестны и недоступны. Поместив в один модуль связанные типы объектов, можно сделать так, что эти объекты смогут обращаться к частным компонентам друг друга, и эти частные компоненты будут неизвестны другим модулям.

27б

```
 Comment [I].Init (1, I + 10, 40, 'первое_имя');
   .
   .
   .
 for I := 1 to 5 do Comment [I].Done;
end;
```

Для динамических экземпляров инициализация, как правило, связана с размещением, а очистка — с удалением, что достигается благодаря расширенному синтаксису стандартных процедур New и Dispose. Например:

```
 var
 SP: StrFieldPtr;
begin
 New (SP, Init (1, 1, 25, 'первое_имя');
 SP^.Put ('Владимир');
 SP^.Display;
  .
  .
  .
 Dispose (SP, Done);
end.
```

Указатель на объектный тип является совместимым по присваиванию с указателем на любой родительский объектный тип, поэтому во время выполнения программы указатель на объектный тип может указывать на экземпляр этого типа или на экземпляр любого дочернего типа.

29a

29. Методы

Описание метода внутри объектного типа соответствует опережающему описанию метода (forward). Таким образом, где-нибудь после описания объектного типа, но внутри той же самой области действия, что и область действия описания объектного типа, метод должен реализоваться путем определения его описания.

Для процедурных и функциональных методов определяющее описание имеет форму обычного описания процедуры или функции с тем исключением, что в этом случае идентификатор процедуры или функции рассматривается как идентификатор метода.

В определяющем описании метода всегда присутствует неявный параметр с идентификатором Self, соответствующий формальному параметру-переменной, обладающему объектным типом. Внутри блока метода Self представляет экземпляр, компонент метода которого был указан для активизации метода. Таким образом, любые изменения значений полей Self отражаются на экземпляре.

Виртуальные методы

По умолчанию методы являются статическими, однако они могут, за исключением конструкторов, быть виртуальными (посредством включения **директивы virtual** в описание метода). Компилятор разрешает ссылки на вызовы статических методов во время процесса компиляции, тогда как вызовы виртуальных методов разрешаются во время выполнения. Это иногда называют поздним связыванием.

Переопределение статического метода не зависит от изменения заголовка метода. В противоположность этому, переопределение виртуального метода должно сохранять порядок, типы и имена параметров, а также

30a

30. Конструкторы и деструкторы

Конструкторы и деструкторы являются специализированными формами методов. Используемые в связи с расширенным синтаксисом стандартных процедур New и Dispose конструкторы и деструкторы обладают способностью размещения и удаления динамических объектов. Кроме того, конструкторы имеют возможность выполнить требуемую инициализацию объектов, содержащих виртуальные методы. Как и все другие методы, конструкторы и деструкторы могут наследоваться, а объекты могут содержать любое число конструкторов и деструкторов.

Конструкторы используются для инициализации вновь созданных объектов. Обычно инициализация основывается на значениях, передаваемых конструктору в качестве параметров. Конструктор не может быть виртуальным, так как механизм диспетчеризации виртуального метода зависит от конструктора, который первым совершил инициализацию объекта.

Приведем несколько примеров конструкторов:

```
constructor Field.Copy(var F: Field);
begin
Self := F;
end;
```

Главным действием конструктора порожденного (дочернего) типа почти всегда является вызов соответствующего конструктора его непосредственного родителя для инициализации наследуемых полей объекта. После выполнения этой процедуры конструктор инициализирует поля объекта, которые принадлежат только порожденному типу.

31a

31. Деструкторы

Borland Pascal предоставляет специальный тип метода, называемый сборщиком мусора (или деструктором) для очистки и удаления динамически размещенного объекта. Деструктор объединяет шаг удаления объекта с какими-либо другими действиями или задачами, необходимыми для данного типа объекта. Для единственного типа объекта можно определить несколько деструкторов.

Деструкторы можно наследовать, и они могут быть либо статическими, либо виртуальными. Поскольку различные программы завершения, как правило, требуют различные типы объектов, обычно рекомендуется, чтобы деструкторы всегда были виртуальными, благодаря чему для каждого типа объекта будет выполнен правильный деструктор.

Зарезервированное слово **destructor** не требуется указывать для каждого метода очистки, даже если определение типа объекта содержит виртуальные методы. Деструкторы в действительности работают только с динамически размещенными объектами.

При очистке динамически размещенного объекта деструктор осуществляет специальные функции: он гарантирует, что в динамически распределяемой области памяти всегда будет освобождаться правильное число байтов. Не может быть никаких опасений по поводу использования деструктора применительно к статически размещенным объектам; фактически, не передавая типа объекта деструктору, программист лишает объект данного типа полных преимуществ управления динамической памятью в Borland Pascal.

Деструкторы в действительности становятся самими собой тогда, когда должны очищаться полиморфические объекты и когда должна освобождаться занимаемая ими память.

32a

32. Виртуальные методы

Метод становится виртуальным, если за его объявлением в типе объекта стоит новое зарезервированное слово **virtual**. Если объявляется метод в родительском типе как virtual, то все методы с аналогичными именами в дочерних типах также должны объявляться виртуальными во избежание ошибки компилятора.

Ниже приведены объекты из примера платежной ведомости, должным образом виртуализированные:

```
type
 PEmployee = ^TEmployee;
 TEmployee = object
 Name, Title: string[25];
 Rate: Real;
 constructor Init (AName, ATitle: String; ARate: Real);
 function GetPayAmount : Real; virtual;
 function GetName : String;
 function GetTitle : String;
 function GetRate : Real;
 procedure Show; virtual;
end;
 PHourly = ^THourly;
 THourly = object(TEmployee);
 Time: Integer;
 constructor Init (AName, ATitle: String; ARate: Real;
Time: Integer);
 function GetPayAmount : Real; virtual;
 function GetTime : Integer;
 end;
 PSalaried = ^TSalaried;
```

30б Деструкторы являются противоположностями конструкторови используются для очистки объектов после их использования.Обычно очистка состоит в удалении всех полей указателей в объекте.

Примечание

Деструктор может быть виртуальным и часто является таковым. Деструктор редко имеет параметры.

Приведем несколько примеров деструкторов:

```
destructor Field.Done;
begin
 FreeMem(Name, Length (Name^) + 1);
end;

destructor StrField.Done;
begin
 FreeMem(Value, Len);
 Field.Done;
end;
```

Деструктор дочернего типа, такой как указанный выше TStrField. Done, обычно сначала удаляет введенные в порожденном типе поля указателей, а затем в качестве последнего действия вызывает соответствующий сборщик-деструктор непосредственного родителя для удаления унаследованных полей указателей объекта.

29б типы результатов функций, если таковые имеются. Более того, переопределение опять же должно включать **директиву virtual**.

Динамические методы

Borland Pascal поддерживает дополнительные методы с поздним связыванием, которые называются динамическими методами. Динамические методы отличаются от виртуальных только характером их диспетчеризации на этапе выполнения. Во всех других отношениях динамические методы считаются эквивалентными виртуальным.

Описание динамического метода эквивалентно описанию виртуального метода, но описание динамического метода должно включать в себя индекс динамического метода, который указывается непосредственно за ключевым словом virtual. Индекс динамического метода должен быть целочисленной константой в диапазоне от 1 до 656535 и должен быть уникальным среди индексов других динамических методов, содержащихся в объектном типе или его предках. Например:

```
procedure FileOpen(var Msg: TMessage); virtual 100;
```

Переопределение динамического метода должно соответствовать порядку, типам и именам параметров и точно соответствовать типу результата функции порождающего метода. Переопределение также должно включать в себя **директиву virtual**, за которой следует тот же индекс динамического метода, который был задан в объектном типе предка.

32б
```
  TSalaried = object(TEmployee);
  function GetPayAmount : Real; virtual;
end;

PCommissioned = ^TCommissioned;
TCommissioned = object(Salaried);
Commission : Real;
SalesAmount : Real;
constructor Init (AName, ATitle: String; ARate,
                  ACommission, ASalesAmount: Real);
function GetPayAmount : Real; virtual;
end;
```

Конструктор является специальным типом процедуры, которая выполняет некоторую установочную работу для механизма виртуальных методов. Более того, конструктор должен вызываться перед вызовом любого виртуального метода. Вызов виртуального метода без предварительного вызова конструктора может привести к блокированию системы, а у компилятора нет способа проверить порядок вызова методов.

Каждый тип объекта, имеющий виртуальные методы, обязан иметь конструктор.

Конструктор должен вызываться перед вызовом любого другого виртуального метода. Вызов виртуального метода без предыдущего обращения к конструктору может вызвать блокировку системы, и компилятор не сможет проверить порядок, в котором вызываются методы.

31б Полиморфические объекты — это те объекты, которые были присвоены родительскому типу благодаря правилам совместимости расширенных типов Borland Pascal. Термин «полиморфический» является подходящим, так как код, обрабатывающий объект, «не знает» точно во время компиляции, какой тип объекта ему придется в конце концов обработать. Единственное, что он знает, — это то, что этот объект принадлежит иерархии объектов, являющихся потомками указанного типа объекта.

Сам по себе метод деструктора может быть пуст и выполнять только эту функцию:

```
destructor AnObject.Done;
begin
end;
```

То, что делается полезного в этом деструкторе, не является достоянием его тела, однако при этом компилятором генерируется код эпилога в ответ на зарезервированное слово destructor. Это напоминает модуль, который ничего не экспортирует, но который осуществляет некоторые невидимые действия за счет выполнения своей секции инициализации перед стартом программы. Все действия происходят «за кулисами».

33a 33. Поля данных объекта и формальные параметры метода

Выводом из того факта, что методы и их объекты разделяют общую область действия, является то, что формальные параметры метода не могут быть идентичными любому из полей данных объекта. Это является не каким-то новым ограничением, налагаемым объектно-ориентированным программированием, а скорее теми же самыми старыми правилами области действия, которые Паскаль имел всегда. Это то же самое, что и запрет для формальных параметров процедуры быть идентичными локальным переменным этой процедуры. Рассмотрим пример, иллюстрирующий эту ошибку для процедуры:

```
procedure CrunchIt(Crunchee: MyDataRec, Crunchby,
          ErrorCode: integer);
var
    A, B: char;
    ErrorCode : integer;
begin
 .
 .
 .
end;
```

На строчке, содержащей объявление локальной переменной *ErrorCode*, возникает ошибка. Это происходит потому, что идентификаторы формального параметра и локальной переменной совпадают.

Локальные переменные процедуры и ее формальные параметры совместно используют общую область действия и поэтому не могут быть идентичными. Бу-

34a 34. Инкапсуляция

Объединение в объекте кода и данных называется инкапсуляцией. В принципе, возможно предоставить достаточное количество методов, благодаря которым пользователь объекта никогда не будет обращаться к полям объекта непосредственно. Некоторые другие объектно-ориентированные языки, например Smalltalk, требуют обязательной инкапсуляции, однако в Borland Pascal имеется выбор.

Например, объекты TEmployee и THourly написаны таким образом, что совершенно исключена необходимость прямого обращения к их внутренним полям данных:

```
type
 TEmployee = object
 Name, Title: string[25];
 Rate: Real;
 procedure Init (AName, ATitle: string; ARate: Real);
 function GetName : String;
 function GetTitle : String;
 function GetRate : Real;
 function GetPayAmount : Real;
end;

 THourly = object(TEmployee)
 Time: Integer;
 procedure Init(AName, ATitle: string; ARate:
              Real, Atime: Integer);
 function GetPayAmount : Real;
end;
```

Здесь присутствуют только четыре поля данных: Name, Title, Rate и Time. Методы GetName и GetTitle

35a 35. Расширяющиеся объекты

Если определен порожденный тип, то методы порождающего типа наследуются, однако при желании они могут переопределяться. Для переопределения наследуемого метода попросту описывается новый метод с тем же именем, что и наследуемый метод, но с другим телом и (при необходимости) с другим множеством параметров.

Определим дочерний по отношению к TEmployee тип, представляющий работника, которому платится часовая ставка, в следующем примере:

```
const
PayPeriods = 26;  { периоды выплат }
OvertimeThreshold = 80; { на период выплаты }
OvertimeFactor = 1.5;  { почасовой коэффициент }

type
THourly = object(TEmployee)
Time: Integer;
procedure Init(AName, ATitle: string; ARate:
   Real, Atime: Integer);
function GetPayAmount : Real;
end;

procedure THourly.Init(AName, ATitle: string;
                       ARate: Real, Atime: Integer);
begin
TEmployee.Init(AName, ATitle, ARate);
Time := ATime;
end;
```

36a 36. Совместимость типов объектов

Наследование до некоторой степени изменяет правила совместимости типов в *Borland Pascal*. Потомок наследует совместимость типов всех своих предков.

Эта расширенная совместимость типов принимает три формы:

1) между реализациями объектов;
2) между указателями на реализации объектов;
3) между формальными и фактическими параметрами.

Совместимость типов расширяется только от потомка к родителю.

Например, *TSalaried* является потомком *TEmployee*, а *TCommissioned* — потомком *TSalaried*. Рассмотрим следующие описания:

```
var
  AnEmployee: TEmployee;
  ASalaried: TSalaried;
  PCommissioned: TCommissioned;
  TEmployeePtr: ^TEmployee;
  TSalariedPtr: ^TSalaried;
  TCommissionedPtr: ^TCommissioned;
```

При данных описаниях справедливы следующие операторы присваивания:

```
  AnEmployee :=ASalaried;
  ASalaried := ACommissioned;
  TCommissionedPtr := ACommissioned;
```

В общем случае правило совместимости типов формулируется так: источник должен быть в состоянии полностью заполнить приемник. Порожденные типы содержат все, что содержат их порождающие типы благодаря свойству наследования. Поэтому порожденный тип имеет размер не меньший размера родителя. Присвоение порождающего объекта порожденному могло бы оставить некоторые поля порожденного объекта неопределенными, что опасно и поэтому недопустимо.

34б выводят фамилию работающего и его должность соответственно. Метод GetPayAmount использует Rate, а в случае работающего THourly и Time для вычисления суммы выплат работающему. Здесь уже нет необходимости обращаться непосредственно к этим полям данных.

Предположив существование экземпляра AnHourly типа THourly, мы могли бы использовать набор методов для манипулирования полями данных AnHourly, например:

```
with AnHourly do
begin
 Init (Aleksandr Petrov, Fork lift operator' 12.95, 62);
  {Выводит на экран фамилию, должность и сумму выплат}
 Show;
end;
```

Следует обратить внимание, что доступ к полям объекта осуществляется не иначе, как только с помощью методов этого объекта.

33б дет получено сообщение *"Error 4: Duplicate identifier"* (Ошибка 4; Повторение идентификатора), если попытаться компилировать что-либо подобное. Та же ошибка возникает при попытке присвоить формальному параметру метода имени поля объекта, которому данный метод принадлежит.

Обстоятельства несколько отличаются, так как помещение заголовка подпрограммы внутрь структуры данных является намеком на новшество в *Turbo Pascal*, но основные принципы области действия Паскаля не изменились.

По-прежнему необходимо соблюдать определенную культуру при выборе идентификаторов переменных и параметров. Некоторые стили программирования предлагают способы именования полей типов, позволяющие снизить риск возникновения дублирующихся идентификаторов. Например, венгерская нотация предлагает имена полей начинать с префикса "*m_*".

36б В операторах присваивания из источника в приемник будут копироваться только поля, являющиеся общими для обоих типов. В операторе присваивания:

```
AnEmployee := ACommissioned;
```

Только поля *Name*, *Title* и *Rate* из *ACommissioned* будут скопированы в *AnEmployee*, так как только эти поля являются общими для *TCommissioned* и *TEmployee*. Совместимость типов работает также между указателями на типы объектов и подчиняется тем же общим правилам, что и для реализаций объектов. Указатель на потомка может быть присвоен указателю на родителя. Если дать предыдущие определения, то следующие присваивания указателей будут допустимыми:

```
TSalariedPtr := TCommissionedPtr;
TEmployeePtr := TSalariedPtr;
TEmployeePtr := PCommissionedPtr;
```

Формальный параметр (либо значение, либо параметр-переменная) данного объектного типа может принимать в качестве фактического параметра объект своего же типа или объекты всех дочерних типов. Если определить заголовок процедуры следующим образом:

```
procedure CalcFedTax(Victim: TSalaried);
```

то допустимыми типами фактических параметров могут быть *TSalaried* или *TCommissioned*, но не тип *TEmployee*. *Victim* также может быть параметром-переменной. При этом выполняются те же правила совместимости.

Параметр-значение является указателем на действительный, посылаемый в качестве параметра объект, а параметр-переменная является копией фактического параметра. Эта копия включает только те поля, которые входят в тип формального параметра-значения. Это означает, что фактический параметр преобразуется к типу формального параметра.

35б
```
function THourly.GetPayAmount: Real;
 var
  Overtime: Integer;
  begin
  Overtime := Time – OvertimeThreshold;
  if Overtime > 0 then
  GetPayAmount := RoundPay(OvertimeThreshold * Rate
+
                          Rate OverTime * OvertimeFactor
* Rate)
  else
  GetPayAmount := RoundPay(Time * Rate)
  end;
```

Вызывая переопределяемый метод, необходимо быть уверенным в том, что порожденный тип объекта включает функциональность родителя. Кроме того, любое изменение в родительском методе автоматически оказывает влияние на все порожденные.

Важное замечание: хотя методы могут быть переопределены, поля данных переопределяться не могут. После того как было определено поле данных в иерархии объекта, никакой дочерний тип не может определить поле данных в точности с таким же именем.

37a

37. Об ассемблере

Когда-то ассемблер был языком, без знания которого нельзя было заставить компьютер сделать что-либо полезное. Постепенно ситуация менялась. Появлялись более удобные средства общения с компьютером. Но в отличие от других языков ассемблер не умирал, более того, он не мог сделать этого в принципе. Почему? В поисках ответа попытаемся понять, что такое язык ассемблера вообще.

Если коротко, то язык ассемблера — это символическое представление машинного языка. Все процессы в машине на самом низком, аппаратном уровне приводятся в действие только командами (инструкциями) машинного языка. Отсюда понятно, что, несмотря на общее название, язык ассемблера для каждого типа компьютера свой. Это касается и внешнего вида программ, написанных на ассемблере, и идей, отражением которых этот язык является.

По-настоящему решить проблемы, связанные с аппаратурой (или, даже более того, зависящие от аппаратуры, как, к примеру, повышение быстродействия программы), невозможно без знания ассемблера.

Программист или любой другой пользователь могут использовать любые высокоуровневые средства вплоть до программ построения виртуальных миров и, возможно, даже не подозревать, что на самом деле компьютер выполняет не команды языка, на котором написана его программа, а их трансформированное представление в форме скучной и унылой последовательности команд совсем другого языка — машинного. А теперь представим, что у такого пользователя возникла нестандартная проблема. К примеру, его программа должна работать с некоторым необычным

38a

38. Программная модель микропроцессора

На современном компьютерном рынке наблюдается большое разнообразие различных типов компьютеров. Поэтому возможно предположить возникновение у потребителя вопроса — как оценить возможности конкретного типа (или модели) компьютера и его отличительные особенности от компьютеров других типов (моделей).

Рассмотрения для этого одной лишь только структурной схемы компьютера недостаточно, так как она принципиально мало чем различается у разных машин: у всех компьютеров есть оперативная память, процессор, внешние устройства.

Различными являются способы, средства и используемые ресурсы, с помощью которых компьютер функционирует как единый механизм.

Чтобы собрать воедино все понятия, характеризующие компьютер с точки зрения его функциональных программно-управляемых свойств, существует специальный термин — архитектура ЭВМ.

Впервые понятие архитектура ЭВМ стало упоминаться с появлением машин 3-го поколения для их сравнительной оценки.

К изучению языка Ассемблера любого компьютера имеет смысл приступать только после выяснения того, какая часть компьютера оставлена видимой и доступной для программирования на этом языке. Это так называемая программная модель компьютера, частью которой является программная модель микропроцессора, которая содержит 32 регистра в той или иной мере доступных для использования программистом.

39a

39. Пользовательские регистры

Как следует из названия, пользовательскими регистры называются потому, что программист может использовать их при написании своих программ. К этим регистрам относятся:

1) восемь 32-битных регистров, которые могут использоваться программистами для хранения данных и адресов (их еще называют регистрами общего назначения (РОН)):
   — eax/ax/ah/al;
   — ebx/bx/bh/bl;
   — edx/dx/dh/dl;
   — ecx/cx/ch/cl;
   — ebp/bp;
   — esi/si;
   — edi/di;
   — esp/sp.
2) шесть регистров сегментов:
   — cs;
   — ds;
   — ss;
   — es;
   — fs;
   — gs;
3) регистры состояния и управления:
   — регистр флагов eflags/flags;
   — регистр указателя команды eip/ip.

На следующем рисунке показаны основные регистры микропроцессора:

40a

40. Регистры общего назначения

Все регистры этой группы позволяют обращаться к своим «младшим» частям. Использовать для самостоятельной адресации можно только младшие 16- и 8-битные части этих регистров. Старшие 16 бит этих регистров как самостоятельные объекты недоступны.

Перечислим регистры, относящиеся к группе регистров общего назначения. Так как эти регистры физически находятся в микропроцессоре внутри арифметико-логического устройства (АЛУ), то их еще называют регистрами АЛУ:

1) eax/ax/ah/al (Accumulator register) — аккумулятор.
   Применяется для хранения промежуточных данных. В некоторых командах использование этого регистра обязательно;
2) ebx/bx/bh/bl (Base register) — базовый регистр.
   Применяется для хранения базового адреса некоторого объекта в памяти;
3) ecx/cx/ch/cl (Count register) — регистр-счетчик.
   Применяется в командах, производящих некоторые повторяющиеся действия. Его использование зачастую неявно и скрыто в алгоритме работы соответствующей команды.
   К примеру, команда организации цикла loop, кроме передачи управления команде, находящейся по некоторому адресу, анализирует и уменьшает на единицу значение регистра ecx/cx;
4) edx/dx/dh/dl (Data register) — регистр данных.
   Так же, как и регистр eax/ax/ah/al, он хранит промежуточные данные. В некоторых командах его использование обязательно; для некоторых команд это происходит неявно.

38б Данные регистры можно разделить на две большие группы:
1) 16 пользовательских регистров;
2) 16 системных регистров.

В программах на языке Ассемблера регистры используются очень интенсивно. Большинство регистров имеют определенное функциональное назначение.

Помимо перечисленных выше регистров, фирмы-разработчики процессоров внедряют в программную модель дополнительные регистры, предназначенные для оптимизации определенных классов вычислений. Так, в семействе процессоров *Pentium Pro* (*MMX*) корпорации *Intel* было внедрено *MMX* расширение от *Intel*. Оно включает в себя 8 (*MM0-MM7*) 64-битных регистров и позволяет производить целочисленные операции над парами нескольких новых типов данных:
1) восемь упакованных байт;
2) четыре упакованных слова;
3) два двойных слова;
4) учетверенное слово;

Другими словами, одной инструкцией *MMX* расширения программист может, например, сложить между собой два двойных слова. Физически никаких новых регистров добавлено не было. *MM0-MM7* это мантиссы (младшие 64 бита) стека 80 битных *FPU* (*floating point unit* — сопроцессор) регистров.

Кроме того, на данный момент существуют следующие расширения программной модели — *3DNOW!* от *AMD*; *SSE*, *SSE2*, *SSE3*, *SSE4*. Последние 4 расширения поддерживаются как процессорами фирмы *AMD*, так и процессорами корпорации *Intel*.

37б устройством или выполнять другие действия, требующие знания принципов работы аппаратуры компьютера. Каким бы хорошим ни был язык, на котором программист написал свою программу, без знания ассемблера ему не обойтись. И не случайно практически все компиляторы языков высокого уровня содержат средства связи своих модулей с модулями на ассемблере либо поддерживают выход на ассемблерный уровень программирования.

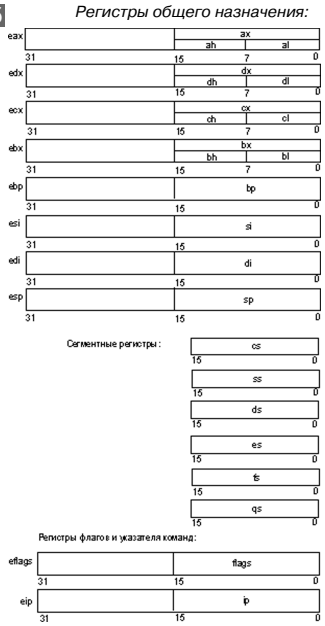
Компьютер составлен из нескольких физических устройств, каждое из которых подключено к одному блоку, называемому системным

40б Следующие два регистра используются для поддержки так называемых цепочечных операций, т. е. операций, производящих последовательную обработку цепочек элементов, каждый из которых может иметь длину 32, 16 или 8 бит:
1) esi/si (Source Index register) — индекс источника. Этот регистр в цепочечных операциях содержит текущий адрес элемента в цепочке-источнике;
2) edi/di (Destination Index register) — индекс приемника (получателя). Этот регистр в цепочечных операциях содержит текущий адрес в цепочке-приемнике.

В архитектуре микропроцессора на программно-аппаратном уровне поддерживается такая структура данных, как стек. Для работы со стеком в системе команд микропроцессора есть специальные команды, а в программной модели микропроцессора для этого существуют специальные регистры:
1) esp/sp (Stack Pointer register) — регистр указателя стека. Содержит указатель вершины стека в текущем сегменте стека.
2) ebp/bp (Base Pointer register) — регистр указателя базы кадра стека. Предназначен для организации произвольного доступа к данным внутри стека.

Использование жесткого закрепления регистров для некоторых команд позволяет более компактно кодировать их машинное представление. Знание этих особенностей позволит при необходимости хотя бы на несколько байт сэкономить память, занимаемую кодом программы.

39б *Регистры общего назначения:*



41a

41. Сегментные регистры

В программной модели микропроцессора имеется шесть сегментных регистров: cs, ss, ds, es, gs, fs.

Их существование обусловлено спецификой организации и использования оперативной памяти микропроцессорами Intel. Она заключается в том, что микропроцессор аппаратно поддерживает структурную организацию программы в виде трех частей, называемых сегментами. Соответственно такая организация памяти называется сегментной.

Для того чтобы указать на сегменты, к которым программа имеет доступ в конкретный момент времени, и предназначены сегментные регистры. Фактически (с небольшой поправкой) в этих регистрах содержатся адреса памяти, с которых начинаются соответствующие сегменты. Логика обработки машинной команды построена так, что при выборке команды, доступе к данным программы или к стеку неявно используются адреса во вполне определенных сегментных регистрах.

Микропроцессор поддерживает следующие типы сегментов.

1. Сегмент кода. Содержит команды программы. Для доступа к этому сегменту служит **регистр cs** (code segment register) — сегментный регистр кода. Он содержит адрес сегмента с машинными командами, к которому имеет доступ микропроцессор (т. е. эти команды загружаются в конвейер микропроцессора).

2. Сегмент данных. Содержит обрабатываемые программой данные. Для доступа к этому сегменту служит **регистр ds** (data segment register) — сегментный регистр данных, который хранит адрес сегмента данных текущей программы.

42a

42. Регистры состояния и управления

В микропроцессор включены несколько регистров, которые постоянно содержат информацию о состоянии как самого микропроцессора, так и программы, команды которой в данный момент загружены на конвейер. К этим регистрам относятся:

1) регистр флагов eflags/flags;
2) регистр указателя команды eip/ip.

Используя эти регистры, можно получать информацию о результатах выполнения команд и влиять на состояние самого микропроцессора. Рассмотрим подробнее назначение и содержимое этих регистров

1. eflags/flags (flag register) — регистр флагов. Разрядность eflags/flags — 32/16 бит. Отдельные биты данного регистра имеют определенное функциональное назначение и называются флагами. Младшая часть этого регистра полностью аналогична регистру flags для i8086.

Исходя из особенностей использования флаги регистра eflags/flags можно разделить на три группы:

1) восемь флагов состояния.
   Эти флаги могут изменяться после выполнения машинных команд. Флаги состояния *регистра eflags* отражают особенности результата исполнения арифметических или логических операций. Это дает возможность анализировать состояние вычислительного процесса и реагировать на него с помощью команд условных переходов и вызовов подпрограмм.
2) один флаг управления.
   Обозначается df (Directory Flag). Он находится в 10-м бите регистра eflags и используется цепочечными командами. Значение флага df опреде-

43a

43. Системные регистры микропроцессора

Само название этих регистров говорит о том, что они выполняют специфические функции в системе. Использование системных регистров жестко регламентировано. Именно они обеспечивают работу защищенного режима. Их также можно рассматривать как часть архитектуры микропроцессора, которая намеренно оставлена видимой для того, чтобы квалифицированный системный программист мог выполнить самые низкоуровневые операции.

Системные регистры можно разделить на три группы:

1) четыре регистра управления;
   В группу регистров управления входят 4 регистра:
   — cr0;
   — cr1;
   — cr2;
   — cr3;
2) четыре регистра системных адресов (которые также называются регистрами управления памятью);
   К регистрам системных адресов относятся следующие регистры:
   — регистр таблицы глобальных дескрипторов *gdtr*;
   — регистр таблицы локальных дескрипторов *ldtr*;
   — регистр таблицы дескрипторов прерываний *idtr*;
   — 16-битовый регистр задачи *tr*;
3) восемь регистров отладки.
   К их числу относятся:
   — dr0;
   — dr1;
   — dr2;
   — dr3;
   — dr4;

44a

44. Регистры управления

В группу регистров управления входят четыре регистра: cr0, cr1, cr2, cr3. Эти регистры предназначены для общего управления системой. Регистры управления доступны только программам с уровнем привилегий 0.

Хотя микропроцессор имеет четыре регистра управления, доступными являются только три из них — исключается cr1, функции которого пока не определены (он зарезервирован для будущего использования).

Регистр cr0 содержит системные флаги, управляющие режимами работы микропроцессора и отражающие его состояние глобально, независимо от конкретных выполняющихся задач.

Назначение системных флагов:

1) pe (Protect Enable), бит 0 — разрешение защищенного режима работы. Состояние этого флага показывает, в каком из двух режимов — реальном (pe = 0) или защищенном (pe = 1) — работает микропроцессор в данный момент времени;
2) mp (Math Present), бит 1 — наличие сопроцессора. Всегда 1;
3) ts (Task Switched), бит 3 — переключение задач. Процессор автоматически устанавливает этот бит при переключении на выполнение другой задачи;
4) am (Alignment Mask), бит 18 — маска выравнивания.
   Этот бит разрешает (am = 1) или запрещает (am = 0) контроль выравнивания;
5) cd (Cache Disable), бит 30 — запрещение кеш-памяти.
   С помощью этого бита можно запретить (cd = 1) или разрешить (cd = 0) использование внутренней кеш-памяти (кеш-памяти первого уровня);

42б ляет направление поэлементной обработки в этих операциях: от начала строки к концу (df = 0) либо наоборот, от конца строки к ее началу (df = 1). Для работы с флагом df существуют специальные команды: cld (снять флаг df) и std (установить флаг df).

Применение этих команд позволяет привести флаг df в соответствие с алгоритмом и обеспечить автоматическое увеличение или уменьшение счетчиков при выполнении операций со строками.

3) пять системных флагов.

Управляют вводом-выводом, маскируемыми прерываниями, отладкой, переключением между задачами и виртуальным режимом 8086. Прикладным программам не рекомендуется модифицировать без необходимости эти флаги, так как в большинстве случаев это приведет к прерыванию работы программы.

2. eip/ip (Instraction Pointer register) — регистр-указатель команд. Регистр eip/ip имеет разрядность 32/16 бит и содержит смещение следующей подлежащей выполнению команды относительно содержимого сегментного регистра cs в текущем сегменте команд. Этот регистр непосредственно недоступен программисту, но загрузка и изменение его значения производятся различными командами управления, к которым относятся команды условных и безусловных переходов, вызова процедур и возврата из процедур. Возникновение прерываний также приводит к модификации регистра eip/ip .

41б 3. Сегмент стека. Этот сегмент представляет собой область памяти, называемую стеком. Работу со стеком микропроцессор организует по следующему принципу: последний записанный в эту область элемент выбирается первым. Для доступа к этому сегменту служит **регистр ss** (stack segment register) — сегментный регистр стека, содержащий адрес сегмента стека.

4. Дополнительный сегмент данных. Не явно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в сегментном регистре ds. Если программе недостаточно одного сегмента данных, то она имеет возможность использовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в сегментном **регистре ds**, при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных префиксов переопределения сегментов в команде. Адреса дополнительных сегментов данных должны содержаться в **регистрах es**, **gs**, **fs** (extension data segment registers).

44б 6) pg (PaGing), бит 31 — разрешение (pg = 1) или запрещение (pg = 0) страничного преобразования.

Флаг используется при страничной модели организации памяти.

Регистр cr2 используется при страничной организации оперативной памяти для регистрации ситуации, когда текущая команда обратилась по адресу, содержащемуся в странице памяти, отсутствующей в данный момент времени в памяти.

В такой ситуации в микропроцессоре возникает исключительная ситуация с номером 14, и линейный 32-битный адрес команды, вызвавшей это исключение, записывается в регистр cr2. Имея эту информацию, обработчик исключения 14 определяет нужную страницу, осуществляет ее подкачку в память и возобновляет нормальную работу программы;

Регистр cr3 также используется при страничной организации памяти. Это так называемый регистр каталога страниц первого уровня. Он содержит 20-битный физический базовый адрес каталога страниц текущей задачи. Этот каталог содержит 1024 32-битных дескриптора, каждый из которых содержит адрес таблицы страниц второго уровня. В свою очередь, каждая из таблиц страниц второго уровня содержит 1024 32-битных дескриптора, адресующих страничные кадры в памяти. Размер страничного кадра — 4 Кбайта.

43б — dr5;
— dr6;
— dr7.

Знание системных регистров не является необходимым для написания программ на Ассемблере, в связи с тем, что они применяются, главным образом, для осуществления самых низкоуровневых операций. Однако в настоящее время тенденции в разработке программного обеспечения (особенно в свете значительно увеличившихся возможностей по оптимизации современных компиляторов высокоуровневых языков, зачастую формирующих код, превосходящий по эффективности код, созданный человеком) сужают область применения Ассемблера до решения самых низкоуровневых задач, где знание вышеозначенных регистров может оказаться весьма полезным.

45a 45. Регистры системных адресов

Эти регистры еще называют регистрами управления памятью.

Они предназначены для защиты программ и данных в мультизадачном режиме работы микропроцессора. При работе в защищенном режиме микропроцессора адресное пространство делится на:

1) глобальное — общее для всех задач;
2) локальное — отдельное для каждой задачи.

Этим разделением и объясняется присутствие в архитектуре микропроцессора следующих системных регистров:

1) регистра таблицы глобальных дескрипторов gdtr (Global Descriptor Table Register), имеющего размер 48 бит и содержащего 32-битовый (биты 16–47) базовый адрес глобальной дескрипторной таблицы GDT и 16-битовое (биты 0—15) значение предела, представляющее собой размер в байтах таблицы GDT;
2) регистра таблицы локальных дескрипторов ldtr (Local Descriptor Table Register), имеющего размер 16 бит и содержащего так называемый селектор дескриптора локальной дескрипторной таблицы LDT. Этот селектор является указателем в таблице GDT, который и описывает сегмент, содержащий локальную дескрипторную таблицу LDT;
3) регистра таблицы дескрипторов прерываний idtr (Interrupt Descriptor Table Register), имеющего размер 48 бит и содержащего 32-битовый (биты 16—47) базовый адрес дескрипторной таблицы прерываний IDT и 16-битовое (биты 0—15) значение предела, представляющее собой размер в байтах таблицы IDT;

46a 46. Регистры отладки

Это очень интересная группа регистров, предназначенных для аппаратной отладки. Средства аппаратной отладки впервые появились в микропроцессоре i486. Аппаратно микропроцессор содержит восемь регистров отладки, но реально из них используются только шесть.

Регистры dr0, dr1, dr2, dr3 имеют разрядность 32 бита и предназначены для задания линейных адресов четырех точек прерывания. Используемый при этом механизм следующий: любой формируемый текущей программой адрес сравнивается с адресами в регистрах dr0 ... dr3 , и при совпадении генерируется исключение отладки с номером 1.

Регистр dr6 называется регистром состояния отладки. Биты этого регистра устанавливаются в соответствии с причинами, которые вызвали возникновение последнего исключения с номером 1.

Перечислим эти биты и их назначение:

1) b0 — если этот бит установлен в 1, то последнее исключение (прерывание) возникло в результате достижения контрольной точки, определенной в регистре dr0;
2) b1 — аналогично b0, но для контрольной точки в регистре dr1;
3) b2 — аналогично b0, но для контрольной точки в регистре dr2;
4) b3 — аналогично b0, но для контрольной точки в регистре dr3;
5) bd (бит 13) — служит для защиты регистров отладки;
6) bs (бит 14) — устанавливается в 1, если исключение 1 было вызвано состоянием флага tf = 1 в регистре eflags;

47a 47. Структура программы на ассемблере

Программа на ассемблере представляет собой совокупность блоков памяти, называемых сегментами памяти. Программа может состоять из одного или нескольких таких блоков-сегментов. Каждый сегмент содержит совокупность предложений языка, каждое из которых занимает отдельную строку кода программы.

Предложения ассемблера бывают четырех типов.

Команды или инструкции, представляющие собой символические аналоги машинных команд.

В процессе трансляции инструкции ассемблера преобразуются в соответствующие команды системы команд микропроцессора. Одна команда Ассемблера, как правило, соответствует одной команде микропроцессора, что, вообще говоря, является характерным для низкоуровневых языков.

Приведем пример инструкции, которая осуществляет увеличение двоичного числа, хранящегося в регистре eax, на единицу:

```
inc eax
```

— макрокоманды — оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями.

Примером макрокоманды может служить следующий макрос конца программы:

```
exit    macro
mov     ax,4c00h
  int   21h
  endm
```

48a 48. Синтаксис ассемблера

Предложения, составляющие программу, могут представлять собой синтаксическую конструкцию, соответствующую команде, макрокоманде, директиве или комментарию. Для того чтобы транслятор ассемблера мог распознать их, они должны формироваться по определенным синтаксическим правилам. Для этого лучше всего использовать формальное описание синтаксиса языка наподобие правил грамматики. Наиболее распространенные способы подобного описания языка программирования — синтаксические диаграммы и расширенные формы Бэкуса-Наура. При работе с синтаксическими диаграммами обращайте внимание на направление обхода, указываемое стрелками. Синтаксические диаграммы отражают логику работы транслятора при разборе входных предложений программы.

Допустимые символы :

1) все латинские буквы: A – Z , a – z;
2) цифры от 0 до 9;
3) знаки ?, @ , $, _ , &;
4) разделители.

Лексемами являются следующие.

1. Идентификаторы — последовательности допустимых символов, использующиеся для обозначения кодов операций, имен переменных и названий меток. Идентификатор не может начинаться цифрой.

2. Цепочки символов — последовательности символов, заключенные в одинарные или двойные кавычки.

3. Целые числа.

Возможные типы операторов ассемблера.

1. Арифметические операторы. К ним относятся:

1) унарные « + » и « – »;
2) бинарные « + » и « – »;

46б 7) bt (бит 15) устанавливается в 1, если исключение 1 было вызвано переключением на задачу с установленным битом ловушки в TSS t = 1.

Все остальные биты в этом регистре заполняются нулями. Об-работчик исключения 1 по содержимому dr6 должен определить причину, по которой произошло исключение, и выполнить необходимые действия.

Регистр dr7 называется регистром управления отладкой. В нем для каждого из четырех регистров контрольных точек отладки имеются поля, позволяющие уточнить следующие условия, при которых следует сгенерировать прерывание:

1) место регистрации контрольной точки — только в текущей задаче или в любой задаче. Эти биты занимают младшие 8 бит регистра dr7 (по 2 бита на каждую контрольную точку (фактически точку прерывания), задаваемую **регистрами dr0**, **dr1**, **dr2**, **dr3** соответственно).

   Первый бит из каждой пары — это так называемое локальное разрешение; его установка говорит о том, что точка прерывания действует, если она находится в пределах адресного пространства текущей задачи.

   Второй бит в каждой паре определяет глобальное разрешение, которое говорит о том, что данная контрольная точка действует в пределах адресных пространств всех задач, находящихся в системе;
2) тип доступа, по которому инициируется прерывание: только при выборке команды, при записи или при записи / чтении данных. Биты, определяющие подобную природу возникновения прерывания, локализуются в старшей части данного регистра.

   Большинство из системных регистров программно доступно.

45б 4) 16-битового регистра задачи tr (Task Register), который подобно регистру ldtr, содержит селектор, т. е. указатель на дескриптор в таблице GDT. Этот дескриптор описывает текущий сегмент состояния задачи (TSS — Task Segment Status). Этот сегмент создается для каждой задачи в системе, имеет жестко регламентированную структуру и содержит контекст (текущее состояние) задачи. Основное назначение сегментов TSS — сохранять текущее состояние задачи в момент переключения на другую задачу.

48б 3)умножения « * »;
4)целочисленного деления « / »;
5) получения остатка от деления « mod ».

2. Операторы сдвига выполняют сдвиг выражения на указанное количество разрядов.

3. Операторы сравнения (возвращают значение «истина» или «ложь») предназначены для формирования логических выражений.

4. Логические операторы выполняют над выражениями побитовые операции.

5. Индексный оператор [ ].

6. Оператор переопределения типа ptr применяется для переопределения или уточнения типа метки или переменной, определяемых выражением.

7. Оператор переопределения сегмента «:» (двоеточие) заставляет вычислять физический адрес относительно конкретно задаваемой сегментной составляющей.

8. Оператор именования типа структуры « . » (точка) также заставляет транслятор производить определенные вычисления, если он встречается в выражении.

9. Оператор получения сегментной составляющей адреса выражения seg возвращает физический адрес сегмента для выражения, в качестве которого могут выступать метка, переменная, имя сегмента, имя группы или некоторое символическое имя.

10. Оператор получения смещения выражения offset позволяет получить значение смещения выражения в байтах относительно начала того сегмента, в котором выражение определено.

47б — директивы, являющиеся указанием транслятору ассемблера на выполнение некоторых действий.

У директив нет аналогов в машинном представлении;

В качестве примера приведем директиву *TITLE*, которая задает заголовок файла листинга:

%TITLE "Листинг 1"

— строки комментариев, содержащие любые символы, в том числе и буквы русского алфавита.

Комментарии игнорируются транслятором.

Пример:

; эта строчка является комментарием

49a 49. Директивы сегментации

Сегментация является частью более общего механизма, связанного с концепцией модульного программирования. Она предполагает унификацию оформления объектных модулей, создаваемых компилятором, в том числе с разных языков программирования. Это позволяет объединять программы, написанные на разных языках. Именно для реализации различных вариантов такого объединения и предназначены операнды в директиве SEGMENT.

Рассмотрим их подробнее.

1. Атрибут выравнивания сегмента (тип выравнивания) сообщает компоновщику о том, что нужно обеспечить размещение начала сегмента на заданной границе:

1) BYTE — выравнивание не выполняется;
2) WORD — сегмент начинается по адресу, кратному двум, т. е. последний (младший) значащий бит физического адреса равен 0 (выравнивание на границу слова);
3) DWORD — сегмент начинается по адресу, кратному четырем;
4) PARA — сегмент начинается по адресу, кратному 16;
5) PAGE — сегмент начинается по адресу, кратному 256;
6) MEMPAGE — сегмент начинается по адресу, кратному 4 Кбайт.

2. Атрибут комбинирования сегментов (комбинаторный тип) сообщает компоновщику, как нужно комбинировать сегменты различных модулей, имеющие одно и то же имя:

1) PRIVATE — сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля;

50a 50. Структура машинной команды

Машинная команда представляет собой закодированное по определенным правилам указание микропроцессору на выполнение некоторой операции или действия. Каждая команда содержит элементы, определяющие:

1) что делать?
2) объекты, над которыми нужно что-то делать (эти элементы называются операндами);
3) как делать?

Максимальная длина машинной команды — 15 байт.

1. Префиксы.

Необязательные элементы машинной команды, каждый из которых состоит из 1 байта или может отсутствовать. В памяти префиксы предшествуют команде. Назначение префиксов — модифицировать операцию, выполняемую командой. Прикладная программа может использовать следующие типы префиксов:

1) префикс замены сегмента;
2) префикс разрядности адреса уточняет разрядность адреса (32- или 16-разрядный);
3) префикс разрядности операнда аналогичен префиксу разрядности адреса, но указывает на разрядность операндов (32- или 16-разрядные), с которыми работает команда;
4) префикс повторения используется с цепочечными командами.

2. Код операции.

Обязательный элемент, описывающий операцию, выполняемую командой.

3. Байт режима адресации modr/m.

Значения этого байта определяет используемую форму адреса операндов. Операнды могут находиться в памяти в одном или двух регистрах. Если операнд находится в памяти, то байт modr/m определяет компоненты (смещение, базовый и индексный регистры),

51a 51. Способы задания операндов команды

Операнд задается неявно на микропрограммном уровне

В этом случае команда явно не содержит операндов. Алгоритм выполнения команды использует некоторые объекты по умолчанию (регистры, флаги в eflags и т. д.).

Операнд задается в самой команде (непосредственный операнд)

Операнд находится в коде команды, т. е. является ее частью. Для хранения такого операнда в команде выделяется поле длиной до 32 бит. Непосредственный операнд может быть только вторым операндом (источником). Операнд-получатель может находиться либо в памяти, либо в регистре.

Операнд находится в одном из регистров

Регистровые операнды указываются именами регистров. В качестве регистров могут использоваться:

1) 32-разрядные регистры EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP;
2) 16-разрядные регистры AX, BX, CX, DX, SI, DI, SP, BP;
3) 8-разрядные регистры AH, AL, BH, BL, CH, CL, DH, DL;
4) сегментные регистры CS, DS, SS, ES, FS, GS.

Например, команда add ax,bx складывает содержимое регистров ax и bx и записывает результат в bx. Команда dec si уменьшает содержимое si на 1.

Операнд располагается в памяти

Это наиболее сложный и в то же время наиболее гибкий способ задания операндов. Он позволяет реализовать следующие два основных вида адресации: прямую и косвенную.

В свою очередь, косвенная адресация имеет следующие разновидности:

52a 52. Способы адресации

Прямая адресация

Это простейший вид адресации операнда в памяти, так как эффективный адрес содержится в самой команде и для его формирования не используется никаких дополнительных источников или регистров. Эффективный адрес берется непосредственно из поля смещения машинной команды, которое может иметь размер 8, 16, 32 бит. Это значение однозначно определяет байт, слово или двойное слово, расположенные в сегменте данных.

Прямая адресация может быть двух типов.

Относительная прямая адресация

Используется для команд условных переходов, для указания относительного адреса перехода. Относительность такого перехода заключается в том, что в поле смещения машинной команды содержится 8-, 16- или 32-битное значение, которое в результате работы команды будет складываться с содержимым регистра указателя команд ip/eip. В результате такого сложения получается адрес, по которому и осуществляется переход.

Абсолютная прямая адресация

В этом случае эффективный адрес является частью машинной команды, но формируется этот адрес только из значения поля смещения в команде. Для формирования физического адреса операнда в памяти микропроцессор складывает это поле со сдвинутым на 4 бита значением сегментного регистра. В команде ассемблера можно использовать несколько форм такой адресации.

Косвенная базовая (регистровая) адресация

При такой адресации эффективный адрес операнда может находиться в любом из регистров общего назначения, кроме sp/esp и bp/ebp (это специфические регистры для работы с сегментом стека). Синтаксически в команде этот режим адресации выражается

50б используемые для вычисления его эффективного адреса. Байт modr/m состоит из трех полей:

1) поле mod определяет количество байт, занимаемых в команде адресом операнда;
2) поле reg/коп определяет либо регистр, находящийся в команде на месте первого операнда, либо возможное расширение кода операции;
3) поле r/m используется совместно с полем mod и определяет либо регистр, находящийся в команде на месте первого операнда (если mod = 11), либо используемые для вычисления эффективного адреса (совместно с полем смещение в команде) базовые и индексные регистры.

4. Байт масштаб — индекс — база (байт sib).

Используется для расширения возможностей адресации операндов. Байт sib состоит из трех полей:

1) поля масштаба ss. В этом поле размещается масштабный множитель для индексного компонента index, занимающего следующие 3 бита байта sib;
2) поля index. Используется для хранения номера индексного регистра, который применяется для вычисления эффективного адреса операнда;
3) поля base. Используется для хранения номера базового регистра, который также применяется для вычисления эффективного адреса операнда.

5. Поле смещения в команде.

8-, 16- или 32-разрядное целое число со знаком, представляющее собой, полностью или частично (с учетом вышеприведенных рассуждений), значение эффективного адреса операнда.

6. Поле непосредственного операнда.

Необязательное поле, представляющее собой 8-, 16- или 32-разрядный непосредственный операнд. Наличие этого поля, конечно, отражается на значении байта modr/m .

49б 2) PUBLIC — заставляет компоновщик соединить все сегменты с одинаковыми именами;
3) COMMON — располагает все сегменты с одним и тем же именем по одному адресу;
4) AT xxxx — располагает сегмент по абсолютному адресу параграфа;
5) STACK — определение сегмента стека.

3. Атрибут класса сегмента (тип класса) — это заключенная в кавычки строка, помогающая компоновщику определить соответствующий порядок следования сегментов при собирании программы из сегментов нескольких модулей.

4. Атрибут размера сегмента:

1) USE16 — это означает, что сегмент допускает 16-разрядную адресацию;
2) USE32 — сегмент будет 32-разрядным.

Необходимо как-то компенсировать невозможность напрямую управлять размещением и комбинированием сегментов. Для этого стали использовать директиву указания модели памяти MODEL. Эта директива связывает сегменты, которые в случае использования упрощенных директив сегментации имеют предопределенные имена, с сегментными регистрами (хотя явно инициализировать ds все равно придется).

Обязательным параметром директивы MODEL является модель памяти. Этот параметр определяет модель сегментации памяти для программного модуля. Предполагается, что программный модуль может иметь только определенные типы сегментов, которые определяются упомянутыми нами ранее упрощенными директивами описания сегментов.

52б заключением имени регистра в квадратные скобки [ ].

Косвенная базовая (регистровая) адресация со смещением

Этот вид адресации является дополнением предыдущего и предназначен для доступа к данным с известным смещением относительно некоторого базового адреса. Этот вид адресации удобно использовать для доступа к элементам структур данных, когда смещение элементов известно заранее, на стадии разработки программы, а базовый (начальный) адрес структуры должен вычисляться динамически, на стадии выполнения программы.

Косвенная индексная адресация со смещением

Этот вид адресации очень похож на косвенную базовую адресацию со смещением. Здесь также для формирования эффективного адреса используется один из регистров общего назначения. Но индексная адресация обладает одной интересной особенностью, которая очень удобна для работы с массивами. Она связана с возможностью так называемого масштабирования содержимого индексного регистра.

Косвенная базовая индексная адресация

При этом виде адресации эффективный адрес формируется как сумма содержимого двух регистров общего назначения: базового и индексного. В качестве этих регистров могут применяться любые регистры общего назначения, при этом часто используется масштабирование содержимого индексного регистра.

Косвенная базовая индексная адресация со смещением

Этот вид адресации является дополнением косвенной индексной адресации. Эффективный адрес формируется как сумма трех составляющих: содержимого базового регистра, содержимого индексного регистра и значения поля смещения в команде.

51б 1) косвенную базовую адресацию; другое ее название — регистровая косвенная адресация;
2) косвенную базовую адресацию со смещением;
3) косвенную индексную адресацию со смещением;
4) косвенную базовую индексную адресацию;
5) косвенную базовую индексную адресацию со смещением.

Операндом является порт ввода/вывода

Помимо адресного пространства оперативной памяти, микропроцессор поддерживает адресное пространство ввода-вывода, которое используется для доступа к устройствам ввода-вывода. Объем адресного пространства ввода-вывода составляет 64 Кбайт. Для любого устройства компьютера в этом пространстве выделяются адреса. Конкретное значение адреса в пределах этого пространства называется портом ввода-вывода. Физически порту ввода-вывода соответствует аппаратный регистр (не путать с регистром микропроцессора), доступ к которому осуществляется с помощью специальных команд ассемблера in и out.

Операнд находится в стеке

Команды могут совсем не иметь операндов, иметь один или два операнда. Большинство команд требуют двух операндов, один из которых является операндом-источником, а второй — операндом назначения. Важно то, что один операнд может располагаться в регистре или памяти, а второй операнд обязательно должен находиться в регистре или непосредственно в команде. Непосредственный операнд может быть только операндом-источником. В двухоперандной машинной команде возможны следующие сочетания операндов:

1) регистр — регистр;
2) регистр — память;
3) память — регистр;
4) непосредственный операнд — регистр;
5) непосредственный операнд — память.

53a 53. Команды пересылки данных

Команды пересылки данных общего назначения

К этой группе относятся следующие команды:

1) **mov** — это основная команда пересылки данных;
2) **xchg** — применяют для двунаправленной пересылки данных.

Команды ввода-вывода в порт

Принципиально управлять устройствами напрямую через порты несложно:

1) **in аккумулятор, номер_порта** — ввод в аккумулятор из порта с номером номер_порта;
2) **out порт, аккумулятор** — вывод содержимого аккумулятора в порт с номером номер_порта.

Команды преобразования данных

К этой группе можно отнести множество команд микропроцессора, но большинство из них имеет те или иные особенности, которые требуют отнести их к другим функциональным группам.

Команды работы со стеком

Эта группа представляет собой набор специализированных команд, ориентированных на организацию гибкой и эффективной работы со стеком.

Стек — это область памяти, специально выделяемая для временного хранения данных программы.

Для работы со стеком предназначены три регистра:

1) **ss** — сегментный регистр стека;
2) **sp/esp** — регистр указателя стека;
3) **bp/ebp** — регистр указателя базы кадра стека.

Для организации работы со стеком существуют специальные команды записи и чтения.

1. **push источник** — запись значения **источник** в вершину стека.

54a 54. Арифметические команды

Такие команды работают с двумя типами:

1) целыми двоичными числами, то есть с числами, закодированными в двоичной системе счисления.

Десятичные числа — специальный вид представления числовой информации, в основу которого положен принцип кодирования каждой десятичной цифры числа группой из четырех бит.

Микропроцессор выполняет сложение операндов по правилам сложения двоичных чисел.

В системе команд микропроцессора имеются три команды двоичного сложения:

1) **inc операнд** — увеличение значения операнда;
2) **add операнд_1, операнд_2** — сложение;
3) **adc операнд_1,операнд_2** — сложение с учетом флага переноса cf.

Вычитание двоичных чисел без знака

Если уменьшаемое больше вычитаемого, то разность положительна. Если уменьшаемое меньше вычитаемого, возникает проблема: результат меньше 0, а это уже число со знаком.

После вычитания чисел без знака нужно анализировать состояние флага CF. Если он установлен в 1, то произошел заем из старшего разряда и результат получился в дополнительном коде.

Вычитание двоичных чисел со знаком

Но для вычитания способом сложения чисел со знаком в дополнительном коде необходимо представлять оба операнда — и уменьшаемое, и вычитаемое. Результат тоже нужно рассматривать как значение в дополнительном коде. Но здесь возникают сложности. Прежде всего они связаны с тем, что старший бит операнда рассматривается как знаковый.

55a 55. Логические команды

Согласно теории, над высказываниями (над битами) могут выполняться следующие логические операции.

1. **Отрицание** (логическое **НЕ**) — логическая операция над одним операндом, результатом которой является величина, обратная значению исходного операнда.

2. **Логическое сложение** (логическое включающее **ИЛИ**) — логическая операция над двумя операндами, результатом которой является «истина» (1), если один или оба операнда имеют значение «истина» (1), и «ложь» (0), если оба операнда имеют значение «ложь» (0).

3. **Логическое умножение** (логическое **И**) — логическая операция над двумя операндами, результатом которой является «истина» (1) только в том случае, если оба операнда имеют значение «истина» (1). Во всех остальных случаях значение операции «ложь» (0).

4. **Логическое исключающее сложение** (логическое исключающее **ИЛИ**) — логическая операция над двумя операндами, результатом которой является «истина» (1), если только один из двух операндов имеет значение «истина» (1), и ложь (0), если оба операнда имеют значение «ложь» (0) или «истина» (1).

Следующий набор команд, поддерживающих работу с логическими данными:

1) **and операнд_1, операнд_2** — операция логического умножения;
2) **or операнд_1, операнд_2** — операция логического сложения;
3) **xor операнд_1, операнд_2** — операция логического исключающего сложения;
4) **test операнд_1, операнд_2** — операция «проверить» (способом логического умножения);

56a 56. Команды передачи управления

То, какая команда программы должна выполняться следующей, микропроцессор узнает по содержимому пары регистров **cs:(e)ip**:

1) cs — сегментный регистр кода, в котором находится физический адрес текущего сегмента кода;
2) eip/ip — регистр указателя команды, в нем находится значение смещения в памяти следующей команды, подлежащей выполнению.

Безусловные переходы

Что должно подвергнуться модификации, зависит:

1) от типа операнда в команде безусловного перехода (ближний или дальний);
2) от указания перед адресом перехода **модификатора**; при этом сам адрес перехода может находиться либо непосредственно в команде (прямой переход), либо в регистре памяти (косвенный переход).

Значения **модификатора**:

1) **near ptr** — прямой переход на метку ;
2) **far ptr** — прямой переход на метку в другом сегменте кода;
3) **word ptr** — косвенный переход на метку ;
4) **dword ptr** — косвенный переход на метку в другом сегменте кода.

Команда безусловного перехода jmp

jmp [модификатор] адрес_перехода

Процедура или **подпрограмма**, — это основная функциональная единица декомпозиции некоторой задачи. Процедура представляет собой группу команд.

Условные переходы

Микропроцессор имеет 18 команд условного перехода. Эти команды позволяют проверить:

1) отношение между операндами со знаком («больше — меньше»);

54б По содержимому флага переполнения of. Его установка в 1 говорит о том, что результат вышел за диапазон представления знаковых чисел (т. е. изменился старший бит) для операнда данного размера, и программист должен предусмотреть действия по корректировке результата.

Принцип вычитания чисел с диапазоном представления, превышающим стандартные разрядные сетки операндов, тот же, что и при сложении, т. е. используется флаг переноса cf. Нужно только представлять себе процесс вычитания в столбик и правильно комбинировать команды микропроцессора с командой sbb.

Для умножения чисел без знака предназначена команда

mul сомножитель_1

Для умножения чисел со знаком предназначена команда

[imul операнд_1,операнд_2,операнд_3]

Для деления чисел без знака предназначена команда **div** делитель

Для деления чисел со знаком предназначена команда **idiv** делитель

53б 2. **pop назначение** — запись значения из вершины стека по месту, указанному операндом **назначение.** Значение при этом «снимается» с вершины стека.

3. **pusha** — команда групповой записи в стек.

4. pushaw — почти синоним команды pusha. Атрибут разрядности может принимать значение use16 или use32. P

5. **pushad** — выполняется аналогично команде pusha, но есть некоторые особенности.

Следующие три команды выполняют действия, обратные вышеописанным командам:

1) **popa;**
2) **popaw;**
3) **popad.**

Группа команд, описанная ниже, позволяет сохранить в стеке регистр флагов и записать слово или двойное слово в стеке.

1. **pushf** — сохраняет регистр флагов в стеке.

2. **pushfw** — сохранение в стеке регистра флагов размером в слово. Всегда работает как pushf с атрибутом use16.

3. **pushfd** — сохранение в стеке регистра флагов flags или eflags в зависимости от атрибута разрядности сегмента (т. е. то же, что и pushf).

Аналогично, следующие три команды выполняют действия, обратные рассмотренным выше операциям:

1) **popf;**
2) **popfw;**
3) **popfd.**

56б 2) отношение между операндами без знака («выше — ниже»);

3) состояния арифметических флагов ZF, SF, CF, OF, PF (но не AF).

Команды условного перехода имеют одинаковый синтаксис:

jcc метка_перехода

Команда сравнения **cmp** имеет интересный принцип работы. Он абсолютно такой же, как и у команды вычитания — **sub** операнд_1, операнд_2.

Команда **cmp** так же, как и команда **sub**, выполняет вычитание операндов и устанавливает флаги. Единственное, чего она не делает — это запись результата вычитания на место первого операнда.

Синтаксис команды cmp — **cmp операнд_1, операнд_2** (compare) — сравнивает два операнда и по результатам сравнения устанавливает флаги.

Организация циклов

Организовать циклическое выполнение некоторого участка программы можно, к примеру, используя команды условной передачи управления или команду безусловного перехода **jmp**:

1) **loop** метка_перехода (Loop) — повторить цикл. Команда позволяет организовать циклы, подобные циклам for в языках высокого уровня с автоматическим уменьшением счетчика цикла;
2) **loope/loopz** метка_перехода

Команды loope и loopz — абсолютные синонимы;

3) **loopne/loopnz** метка_перехода

Команды loopne и loopnz также абсолютные синонимы.

Команды **loope/loopz** и **loopne/loopnz** по принципу своей работы являются взаимообратными.

55б 5) **not операнд** — операция логического отрицания.

а) для **установки** определенных разрядов (бит) в **1** применяется команда **or** операнд_1, операнд_2;
б) для **сброса** определенных разрядов (бит) в 0 применяется команда **and** операнд_1, операнд_2;
в) команда **xor** операнд_1, операнд_2 применяется:
   — для выяснения того, какие биты в операнд_1 и операнд_2 **различаются**;
   — для **инвертирования** состояния заданных бит в операнд_1.

Для проверки состояния заданных бит применяется команда **test** операнд_1, операнд_2 (проверить операнд_1).

Результатом команды является установка значения флага нуля **zf**:

1) если zf = 0, то в результате логического умножения получился нулевой результат, т. е. один единичный бит маски, который **не совпал** с соответствующим единичным битом операнд_1;
2) если zf = 1, то в результате логического умножения получился ненулевой результат, т. е. **хотя бы один** единичный бит маски **совпал** с соответствующим единичным битом операнд_1.

Все команды сдвига перемещают биты в поле операнда влево или вправо в зависимости от кода операции. Все команды сдвига имеют одинаковую структуру — **коп операнд, счетчик_сдвигов**.