А.А. АШБАЕВ, К.Т. МАНСУРОВ

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И НАЧАЛА ПРОГРАММИРОВАНИЯ

Ош – 2005

МИНИСТЕРСТВО ОБРАЗОВАНИЯ КЫРГЫЗСКОЙ РЕСПУБЛИКИ

ОШСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

А. А. АШБАЕВ, К. Т. МАНСУРОВ

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И НАЧАЛА ПРОГРАММИРОВАНИЯ

Ош – 2005

УДК 004
ББК 32.973.-01
А-98

Утвержден в качестве учебного пособия учебно-методическим советом ОшГУ.

Рецензенты:
доктор физико-математических наук, профессор Алымкулов К. А.
доктор физико-математических наук, профессор Сопуев А. С.

А-98 Ашбаев А. А., Мансуров К. Т.
Теоретические основы вычислительной техники и начала программирования: Уч. пособие /ОшГУ. – Ош, 2005. –137 с.

ISBN 9967-03-254-5

Настоящий учебник предназначен для студентов младших курсов специальности «Программное обеспечение вычислительной техники и автоматизированных систем», а также специальности «Вычислительные машины, комплексы, системы и сети», хотя многие главы этой книги могут с успехом использовать студенты и других специальностей при изучении информатики.

Книга является введением в специальность для программистов. Она вводит читателя в круг тех идей, понятий, принципов и методов, на которых зиждется современное программирование и информационные технологии.

Приведены многочисленные задачи и их решения, а также сведения по численным методам, алгоритмах их реализации и пакетах программ.

Книга предназначена для студентов ВУЗов, специализирующихся в области программного обеспечения вычислительной техники.

А 2404010000-05

УДК 004

ББК 32.973-01

ISBN 9967-03-254-5

ПРЕДИСЛОВИЕ

Настоящая книга предназначена в первую очередь для студентов младших курсов специальности "Программное обеспечение вычислительной техники и автоматизированных систем", а также специальности "Вычислительные машины, комплексы, системы и сети", хотя многие главы этой книги могут с успехом использовать студенты и других специальностей при изучении информатики.

Книга является введением в специальность для программистов. Она вводит читателя в круг тех идей, понятий, принципов и методов, на которых зиждется современное программирование и информационные технологии.

Можно и нужно читать эту книгу непосредственно сидя за компьютером!

Небольшое замечание по поводу работы с книгой за компьютером – всюду в тексте, когда говорится щелкните клавишей мыши, подразумевается левая клавиша. Когда речь идет о правой клавише, то это особо оговаривается в тексте.

Авторы

ВВЕДЕНИЕ

Становление информатики как научной дисциплины относится к 60-годам прошлого столетия. В самом общем смысле под информатикой понимают фундаментальную естественную науку, изучающую процессы передачи, накопления и обработки информации.

Термин "Informatique" был введен французами на рубеже 60-70-х годов. Но еще раньше американцами был введен в употребление термин "Computer Science" для обозначения науки о преобразовании информации, базирующейся на вычислительной технике.

В настоящее время термины "Informatique" и "Computer Science" употребляются в эквивалентном смысле.

В нашей стране для обозначения новой научной дисциплины применялись термины "Вычислительные науки" или " Вычислительное дело", использование же термина "Информатика" сдерживалось употреблением его в области, связанной с документалистикой. Новое толкование этого термина относится к 1976г., когда появился русский перевод книги Ф. Бауэра и Г.Гооза " Информатика".

Известно широкое определение информатики, данное Международным конгрессом в Японии в 1978г.: "Понятие информатики охватывает области, связанные с разработкой, созданием, использованием и материально-техническим обслуживанием систем обработки информации, включая машины, оборудование, математическое обеспечение, организационные аспекты, а также комплекс промышленного, коммерческого, административного, социального и политического воздействия". В этом определении можно выделить то главное, что и составляет основу современного содержания информатики, а именно компьютеры и компьютерную обработку информации.

Исходя из такого толкования ядра информатики можно сказать, что его содержание определяют три неразрывно связанные между собой части: алгоритмические, программные и технические средства. Элементы этих составных частей информатики и рассматриваются в настоящем учебнике.

Информация и информатика

Информация об окружающем мире необходима человеку так же, как воздух, вода, тепло. Представьте себе, что Вы оказались в замкнутом пространстве без света и звука, с абсолютно гладкими поверхностями. Вы почувствуете себя очень неуютно. Отсутствие информации о времени, пространстве, окружающей действительности противоестественно для человека. Человек видит, слышит, осязает, чувствует запахи. Органы чувств служат человеку для получения информации.

Информация – это сообщение о состоянии и свойствах объекта, явления, процесса. Информация преобразуется уникальным устройством-человеческим мозгом. Ребенок, впервые увидевший огонь, не боится его. Но обжегшись, запоминает на всю жизнь – с огнем играть нельзя. Значит, мозг человека не только преобразует информацию, но и запоминает – хранит ее. Человечество за тысячелетия своего существования накопило огромное количество информации. Мозг человека не в состоянии хранить такой объем ее и без искажения передавать другим людям. Поэтому для хранения и накопления информации с незапамятных времен использовались природные средства: рисунки на стенах пещер, скалах. Носители информации непрерывно совершенствовались, так постепенно появились: пергамент, папирус, береста, бумага, перфорационные носители информации (перфокарты, перфоленты), фотопленка, магнитные носители (ленты, карты, диски и т.д.).

Первобытный человек, создав первые примитивные орудия труда, положил начало эпохе механизмов, увеличивающих физическую силу человека. В середине XX столетия были созданы первые электронные вычислительные машины, предназначенные для увеличения интеллектуальной мощи человека. Появление ЭВМ не случайно. Лавинообразно растут информационные потоки. Особенно это характерно для промышленности, управления и науки. Общая сумма человеческих знаний в XVIII веке удваивалась каждые 50 лет, к 1950г. – каждые 10 лет, к 1970г. – 5 лет, к 1985г. –каждые 2-3 года. До XX века основным предметом труда были материальные объекты, экономическая мощь государства измерялась его материальными ресурсами, теперь основным предметом труда становится информация. Человек не в состоянии без электронных помощников навести порядок в информационном хозяйстве.

Возросшая роль информации и уровень развития средств обработки информации привели к появлению понятия "информационные ресурсы". Информационные ресурсы – информация, используемая на производстве, в технике, в управлении обществом, специально организованная и обрабатываемая на компьютере. Процесс компьютеризации общества привел к появлению нового термина – пользователь. Пользователем будем называть человека любой профессии, который непосредственно взаимодействует с компьютером.

Формы и виды представления информации

Выделяют две формы представления информации – непрерывную и дискретную. Непрерывная форма – это величина, характеризующая процесс, не имеющий перерывов или промежутков, например: температура человека,

скорость перемещения автомобиля за определенное время на участке пути без остановок.

Дискретная форма представления информации – это последовательность символов, характеризующая прерывистую, изменяющуюся величину. Например, рабочий за первый час смены обработал на станке 40 деталей, за второй –45, за третий 38 и т.д.

Часто возникает необходимость представить информацию в форме отличной от привычной. Так, для передачи информации на расстояние изобретен телеграфный код Морзе, в которой буквы и цифры закодированы с помощью коротких и длинных импульсов (точка, тире). Почтовый индекс – закодированный адрес.

Электронные элементы, применяемые в компьютерах, имеют два состояния – есть импульс или нет импульса, поэтому вся информация для компьютеров кодируется в двоичной системе счисления. Любой символ (цифра, буква, знак) получает кодированное обозначение с помощью цифр 1 и 0, составляющих основу двоичной системы счисления.

Рассмотрим основные единицы измерения информации. Наименьшей единицей информации является бит (bit), принимающий только значение 1 или 0. Более крупная единица информации – байт (byte). Он состоит из восьми бит. Байт – основная единица количества информации, используется для образования еще более крупных единиц информации.

ГЛАВА I. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

§1.1. Системы счисления

Система счисления – совокупность правил и приемов для записи чисел с помощью различных знаков или символов.

Системы счисления делятся на позиционные и непозиционные.

Непозиционная система счисления – система, для которой значение символа не зависит от его положения в числе. Примером непозиционной системы счисления может служить римская система счисления. Используется набор знаков I, V, X, L, C, D, M, …

Например: числа IV и VI Значение символа V одно и то же и равно пяти.

Позиционная система счисления – система, которая использует фиксированный набор знаков (символов) для записи числа и значение этого знака (символа) зависит от его положения в числе.

Например 654 это не одно и то же, что 645

Любая позиционная система счисления характеризуется основанием или базисом.

Базисом (основанием) q позиционной системы счисления является количество знаков или символов используемых для записи числа в данной системе счисления.

Например, в десятичной системе счисления $q=10$, используются знаки или цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Для любой позиционной системы счисления имеет место равенство:

$$A_{(q)} = a_n q^n + a_{n-1} q^{n-1} + \ldots + a_1 q^1 + a_0 q^0 + a_{-1} q^{-1} + \ldots + a_{-m} q^{-m} \qquad (1.1.1)$$

где $A_{(q)}$ - произвольное число в q-ичной системе счисления, q - базис данной системы счисления, a_i - знаки или символы данной системы счисления.

Для десятичной системы имеем:

$q=10$, $a_i=\{0,1,2,3,4,5,6,7,8,9\}$

$$A_{(10)} = a_n 10^n + a_{n-1} 10^{n-1} + \ldots + a_1 10^1 + a_0 10^0 + a_{-1} 10^{-1} + \ldots + a_{-m} 10^{-m} \qquad (1.1.2)$$

или в сокращенной записи:

$$A_{(10)} = \sum_{i=-m}^{n} a_i 10^i \qquad (1.1.3)$$

На практике используют условную, сокращенную запись числа:

$$A_{(q)} = a_n a_{n-1} \dots a_1 a_0, a_{-1} a_{-2} \dots a_{-m} \qquad (1.1.4)$$

Пример:

$$A_{(10)} = 654 = 6 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$$

$$A_{(10)} = 358{,}124 = 3 \cdot 10^2 + 5 \cdot 10^1 + 8 \cdot 10^0 + 1 \cdot 10^{-1} + 2 \cdot 10^{-2} + 4 \cdot 10^{-3}$$

Существуют и другие системы счисления:

1. **Двоичная система счисления**

$$q = 2, \quad a_i = \{0,1\}$$

$$A_{(2)} = \sum_{i=-m}^{n} a_i 10^i \qquad (1.1.5)$$

Правила образования чисел как в десятичной системе: вес k-й позиции числа равен q^k. Для двоичной системы 2^k.

$$0_{(10)} = 0_{(2)}$$

$$1_{(10)} = 1_{(2)}$$

$$2_{(10)} = 10_{(2)}$$

$$3_{(10)} = 11_{(2)}$$

$$4_{(10)} = 100_{(2)}$$

Число $14_{(10)} = 1110_{(2)} = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 4 + 2 + 0 = 14_{(10)}$

2. **Восьмеричная система счисления**

$$q = 8, \quad a_i = \{0,1,2,3,4,5,6,7\}$$

$$A_{(10)} = 118{,}375_{(10)} = 166{,}3_{(8)} = 1 \cdot 8^2 + 6 \cdot 8^1 + 6 \cdot 8^0 + 3 \cdot 8^{-1} = 64 + 48 + 6 + \frac{3}{8} = 118{,}375_{(10)}$$

3. **16 – ричная система счисления**

$$q = 16, \quad a_i = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$$

$$100_{(10)} = 64_{(16)} = 6 \cdot 16^1 + 4 \cdot 16^0 = 100_{(10)}$$

Разрядом называется позиция одной цифры в числе. Вес разряда p_i в позиционной системе счисления это отношение $p_i = \frac{q^i}{q^0} = q^i$, где i- номер разряда справа налево.

Если разряд p_k имеет вес q^k, то следующий старший разряд (слева) имеет вес $p_{k+1} = q^{k+1}$, а следующий младший разряд имеет вес $p_{k-1} = q^{k-1}$.

Такая взаимосвязь между разрядами приводит к необходимости передачи информации между разрядами. Если в текущем разряде накопилось единиц больше или равное q, должна происходить передача единицы в

соседний старший разряд. Этот процесс называется переносом. Передача единицы из старшего разряда в младший называется заемом.

Длиной числа называется количество разрядов необходимым для записи числа. В техническом аспекте длина числа интерпретируется как длина разрядной сетки процессора. В большинстве компьютеров длина разрядной сетки фиксирована. Если длина разрядной сетки задана, то это ограничивает максимальное и минимальное по модулю число, представимое в компьютере.

Пусть длина разрядной сетки равна $n (n > 0$, целое). Тогда

$$A_{(q)\max} = q^n - 1;\ A_{(q)\min} = -(q^n - 1) \qquad (1.1.6)$$

Отсюда диапазон представления чисел в заданной системе счисления и для заданной разрядной сетке это интервал числовой оси, заключенный между $A_{(q)\max}$ и $A_{(q)\min}$, т.е.

$$A_{(q)\min} \le \text{ДП} \le A_{(q)\max} \qquad (1.1.7)$$

Таблица соответствия десятичных чисел в других системах счисления

Десятичные числа	Их эквиваленты в других системах		
	q=2	q=8	q=16
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Перевод чисел из одной системы счисления в другую

Пусть необходимо перевести число в системе счисления с основанием q в число в системе счисления с основанием r.

$$A_{(q)} = a_n q^n + a_{n-1} q^{n-1} + \ldots + a_1 q^1 + a_0 q^0 + a_{-1} q^{-1} + \ldots + a_{-m} q^{-m} = \\ = b_k r^k + \cdots + b_1 r^1 + b_0 r^0 + b_{-1} r^{-1} + \cdots + b_{-s} r^{-s} = A_{(r)} \qquad (1.1.8)$$

Задачу перевода числа из системы счисления q в систему счисления r можно представить в общем виде как задачу определения коэффициентов нового ряда b_j

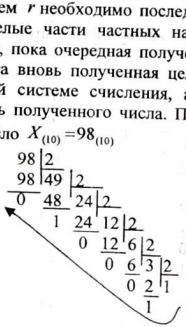
$$0 \le a_i \le q-1, \quad i \in [-m, n] \\ 0 \le b_j \le r-1, \quad j \in [-s, k] \qquad (1.1.9)$$

Например,

$56_{(10)} = 5 \cdot 10^1 + 6 \cdot 10^0 = 111000_{(2)} = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 32 + 16 + 8 = 56_{(10)}$

Правило перевода целых чисел

Для перевода целого числа в системе счисления с основанием q в число в системе с основанием r необходимо последовательно делить это число и получаемые затем целые части частных на основание r новой системы счисления до тех пор, пока очередная полученная целая часть частного не станет меньше r. Эта вновь полученная целая часть определит старшую цифру числа в новой системе счисления, а все предыдущие остатки – младшие цифры вновь полученного числа. Пример: перевести в двоичную систему счисления число $X_{(10)} = 98_{(10)}$



Получим $X_{(10)} = 98_{(10)} = 1100010_{(2)} = X_{(2)}$

Правило перевода правильных дробей.

Для перевода правильной дроби из одной системы счисления в другую надо её и получающиеся затем дробные части произведений умножать на основание новой системы счисления до тех пор, пока в новой дроби не будет нужного количества цифр, которая определяется требуемой точностью представления дроби. Правильная дробь в новой системе счисления записывается из целых частей произведений, получающихся при

последовательном умножении, причем первая целая часть будет старшей цифрой новой дроби.

Пример: $X_{(10)}=0{,}375$ перевести в двоичную систему с 4-мя знаками после запятой.

$$\begin{array}{r} 0{,}375 \\ \times\ \underline{\quad 2} \\ 0{,}750 \\ \times\ \underline{\quad 2} \\ 1{,}500 \\ \times\ \underline{\quad 2} \\ 1{,}000 \\ \times\ \underline{\quad 2} \\ 0{,}000 \end{array}$$

$$X_{(2)}=0{,}0110_{(2)}=0{,}375_{(10)}$$

Правило перевода неправильных дробей.

При переводе неправильных дробей отдельно переводят целую и дробную части числа по правилам, приведенными выше.

$$X_{(10)}=98{,}375_{(10)}=1100010{,}0110_{(2)}$$

Представление чисел в памяти компьютера

Существуют две формы представления чисел: естественная и нормальная.

Естественной называется такая форма записи числа, которая в неявном виде реализует формулу:

$$X=\sum_{i=-m}^{n} a_i q^i \qquad (1.1.10)$$

При этом число записывают только при помощи набора цифр a_i, без явного указания их весов и знаков сложения между ними.

Отсчет весов ведется от запятой, которая фиксируется между целой и дробной частью числа. Запись числа имеет вид:

$$a_n a_{n-1}\cdots a_1 a_0, a_{-1}\cdots a_{-m} \qquad (125=1\cdot 10^2+2\cdot 10^1+5\cdot 10^0)$$

Это формой мы обычно пользуется.

Нормальной называется такая форма записи числа, которая в неявном виде реализует формулу:

$$X=\left(\sum_{i=-m}^{n} a_i q^{i-p}\right) q^p \qquad (1.1.11)$$

когда число представляется в виде произведения правильной дроби и некоторой целой степени основания системы счисления.

При этом правильная дробь называется мантиссой, а показатель степени – порядком. Число 125 в нормальной форме представляется в виде:

$$0{,}125\cdot 10^3$$

Так как в естественной форме положение запятой в числе строго зафиксирована между целой и дробной частью, то часто числа в такой форме называют числами с фиксированной запятой. Следует отметить, что во многих западных странах для отделения целой и дробной частей числа используют точку вместо запятой. Поэтому часто числа в такой форме называют ещё числами с фиксированной точкой.

В дальнейшем мы будем пользоваться в основном термином "с фиксированной точкой" как наиболее распространенной и часто используемой в литературе.

В определении нормальной формы не наложено никаких ограничений на величину мантиссы, лишь бы она была правильной дробью. Поэтому положение запятой может меняться при изменении порядка. Поэтому числа в нормальной форме называют числами с плавающей точкой или с плавающей запятой.

Недостатки чисел с плавающей точкой:

1. Сложная конструкция аппаратуры процессора для реализации различных операций над числами из-за неоднозначности записи чисел и необходимости вследствие этого нормализации числа.
2. Ещё большее усложнение аппаратуры, во – первых из – за сложности структуры числа, состоящей из двух частей: мантиссы и порядка и, во – вторых вследствие того, что приходится производить действия с обоими частями числа.

Представление с фиксированной точкой не имеет указанных недостатков, но, к сожалению, у этой формы есть свои недостатки:

1. Малый диапазон чисел, представимых в компьютере. Пусть имеется некий гипотетический компьютер с 9-ти разрядной сеткой для записи десятичных чисел. Пусть также задано, что компьютер может работать в двух режимах: с фиксированной и с плавающей точкой, причем на мантиссу отведено 5 разрядов и на дробную часть отведено тоже 5 разрядов.

Форма с фиксированной точкой:

1	2	3	4	5	6	7	8	9
знак	целая часть			дробная часть				

Форма с плавающей точкой:

1	2	3	4	5	6	7	8	9
знак мантиссы	мантисса					знак порядка	порядок	

Знак числа имеет двоичную природу, так как может принимать лишь два значения (+) и (-). Знак (+) принято изображать цифрой 0, а знак (-) цифрой 1

Режим с фиксированной точкой.	Режим с плавающей точкой.
$+\lvert X\rvert_{\max} = +999{,}99999$	$+\lvert X\rvert_{\max} = +0{,}99999 \cdot 10^{+99}$
$+\lvert X\rvert_{\min} = +000{,}00000$	$+\lvert X\rvert_{\min} = +0{,}10000 \cdot 10^{-99}$
$-\lvert X\rvert_{\min} = -000{,}00000$	$-\lvert X\rvert_{\min} = -0{,}10000 \cdot 10^{-99}$
$-\lvert X\rvert_{\max} = -999{,}99999$	$-\lvert X\rvert_{\max} = -0{,}99999 \cdot 10^{+99}$

Отсюда мы можем получить оценку диапазонов представления чисел в обоих формах:

$$-10^{+3} < X_{ф.т.} < +10^{+3}$$
$$-10^{+99} < X_{п.т.} < +10^{+99} \qquad (1.1.12)$$

2. Меньшая точность.

Рассмотрим для двоичной системы. Абсолютная погрешность для чисел с фиксированной точкой составляет:

$$\Delta_{ф.т.} = \frac{1}{2}2^{-n} = const \qquad (1.1.13)$$

Абсолютная погрешность для чисел с плавающей точкой составляет:

$$\Delta_{п.т.} = \frac{1}{2}2^{-n}2^{p} \qquad (1.1.14)$$

$\Delta_{п.т.}$ зависит от p и следовательно меняется от

$$\left|\Delta_{п.т.}\right|_{\min} = \frac{1}{2}2^{-n}2^{-|p_{\max}|} \qquad (1.1.15)$$

до

$$\left|\Delta_{п.т.}\right|_{\max} = \frac{1}{2}2^{-n}2^{+|p_{\max}|} \qquad (1.1.16)$$

Относительная погрешность для чисел с фиксированной точкой определяется следующим образом:

$$\delta_{ф.т.\min} = \frac{\Delta_{ф.т.}}{\left|X_{ф.т.\max}\right|} \qquad (1.1.17)$$

$$\delta_{ф.т.\max} = \frac{\Delta_{ф.т.}}{\left|X_{ф.т.\min}\right|} \qquad (1.1.18)$$

где

$$\left|X_{ф.т.\max}\right| = \overbrace{0{,}11\cdots1}^{n} = 1 - 2^{-n}$$
$$\left|X_{ф.т.\min}\right| = 0{,}00\cdots1 = 2^{-n} \qquad (1.1.19)$$

Теперь можно получить оценку относительной погрешности для чисел с фиксированной точкой:

$$\delta_{\phi.m.\min} = \frac{\frac{1}{2}2^{-n}}{1-2^{-n}} \cong 2^{-(n+1)} \tag{1.1.20}$$

если $1 >> 2^{-n}$

$$\delta_{\phi.m.\max} = \frac{\frac{1}{2}2^{-n}}{2^{-n}} = \frac{1}{2} = 50\% \tag{1.1.21}$$

То есть максимальная относительная погрешность может достигать 50%, что конечно во многих случаях неприемлемо!

Рассмотрим теперь относительную погрешность для чисел с плавающей точкой:

$$\delta_{n.m.} = \frac{\Delta_{n.m.}}{|X_{n.m.}|}$$

$$\delta_{n.m.\min} = \frac{|\Delta_{n.m.\max}|}{|X_{n.m.\max}|} = \frac{\frac{1}{2}2^{-n}2^{+|P_{\max}|}}{(1-2^{-n})2^{+|P_{\max}|}} \cong 2^{-(n+1)} \tag{1.1.22}$$

при $1 >> 2^{-n}$

$$\delta_{n.m.\max} = \frac{|\Delta_{n.m.\min}|}{|X_{n.m.\min}|} = \frac{\frac{1}{2}2^{-n}2^{-|P_{\max}|}}{\frac{1}{2}2^{-|P_{\max}|}} = 2^{-n} \tag{1.1.23}$$

Отсюда видно, что относительная погрешность $\delta_{n.m.}$ не зависит от величины P, а зависит от n.

Главный вывод в том, что величина ошибки для чисел с плавающей точкой изменяется в очень малом диапазоне и очень мала ($2^{-n} << 1$), если числа нормализованы.

§1.2. *Кодирование чисел*

Вопрос о кодировании чисел возникает по той причине, что в компьютер либо нельзя, либо нерационально вводить числа в том виде, каком они изображаются человеком – вычислителем на бумаге.

Во – первых, нужно кодировать знак числа. Во – вторых, приходится иногда и само число кодировать. В частности, при выполнении операций с фиксированной точкой все числа приводятся к виду правильных дробей.

Простейший машинный код, называемый прямым, получается в том случае, когда знак двоичной дроби кодируется 0 (+) и 1 (-), а цифровая часть остается без изменений.

Более строго прямой код можно определить следующим образом:

Прямым кодом отрицательной двоичной дроби называется её обычное изображение в естественной форме у которого в знаковом разряде проставляется единица. Прямой код положительной двоичной дроби совпадает с её обычным изображением в естественной форме. Аналитически:

$$[X]_{пр} = \begin{cases} X, & X \geq 0 \\ 1 - X, & X < 0 \end{cases} \qquad (1.2.1)$$

$$X_1 = +0{,}110011_{(2)} \qquad [X_1]_{пр} = 0{,}110011$$

$$X_2 = -0{,}110011_{(2)} \qquad [X_2]_{пр} = 1{,}110011$$

Так как операция сложения значительно проще реализуется, чем операция вычитания, то возникает вопрос: нельзя ли каким либо способом свести алгебраическое сложение к арифметическому, т.е. попросту говоря заменить операцию вычитания сложением.

Идея замены операции вычитания сложением основана на применении некоторых вспомогательных положительных чисел, однозначно связанных с исходными отрицательными числами. Эти вспомогательные числа находятся как дополнения (по абсолютной величине) заданных отрицательных чисел до некоторого $X_{гр}$:

$$X_{гр} = |X_{\max}| + 1_{м.р.} = 10^n \qquad (1.2.2)$$

где $1_{м.р.}$ означает единицу младшего разряда.

Пусть имеются двухразрядные десятичные числа.

$$0 \leq X_{тек} \leq X_{\max} = 99_{(10)}$$

Здесь $X_{гр} = X_{\max} + 1 = 99 + 1 = 100 = 10^2$

Пусть нужно алгебраически сложить

$$X_1 = 84 \quad \text{и} \quad X_2 = -63$$

X_2 согласно вышесказанному, мы заменим его дополнением (по абсолютной величине) до $X_{гр}$ (до ста), т.е.

$$[X_2]_{доп} = X_2 + X_{гр} = X_2 + 100 = -63 + 100 = 37$$

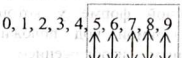
Далее складываем X_1 и $[X_2]_{доп}$

$$X_1 + [X_2]_{доп} = 84 + 37 = 121$$

(1 — отбрасывается)

Прямое вычитание $84 - 63 = 21$ показывает правильность наших действий.

Нахождение $[X_2]_{доп}$ производят без вычитания. Выпишем цифры от 0 до 9 сначала в прямом порядке, затем в обратном:



Отсюда видно, что замена в некотором отрицательном числе всех цифр на взаимно обратные равносильно сложению исходного числа с $|X|_{\max}$.

Правило нахождения дополнения отрицательных чисел без использования операции вычитания:

для получения дополнения отрицательного числа следует все цифры исходного числа заменить на взаимно обратные и к полученному результату добавить единицу младшего разряда.

Действительно:

$$[X_2]_{доп} = X_2 + X_{гр} = X_2 + |X_{\max}| + 1 = -63 + 99 + 1 = 36 + 1 = 37$$

Аналогично и вместе с тем проще для двоичной системы счисления.

0 1
1 0 взаимно обратные цифры

Дополнение здесь ищется до двух.

Дополнительным кодом отрицательной двоичной дроби называется её дополнение по модулю до двух.

Правило:

Для получения дополнительного кода отрицательной двоичной дроби нужно в знаковом разряде записать 1, а во всех других разрядах цифры заменить на взаимно обратные, т.е. 0 $\rightarrow$ 1, 1 $\rightarrow$ 0, после чего к младшему разряду числа добавляется 1.

Дополнительный код положительной двоичной дроби совпадает с её прямым кодом. Аналитически это можно выразить следующим образом:

$$[X]_{доп} = \begin{cases} X, & X \geq 0 \\ X + 10_{(2)}, & X < 0 \end{cases} \qquad (1.2.3)$$

$X_1 = +0{,}110011 \qquad [X_1]_{доп} = 0{,}110011$

$X_2 = -0{,}110011 \qquad [X_2]_{доп} = 1{,}001100$

$$+ \quad 1$$

$$1{,}001101$$

Переход от дополнительного кода отрицательного числа к прямому коду производится следующим образом:

Заменяют все цифры числа, за исключением знакового на взаимно обратные. После чего добавляют единицу к младшему разряду.

$$[X]_{доп} = 1{,}001101$$
$$[X]_{пр} = 1{,}110010$$
$$\underline{+\qquad 1}$$
$$1{,}110011$$

Обратный код чисел

Обратным кодом отрицательной двоичной дроби называется её дополнение по модулю до двух без единицы младшего разряда.

Он получается по правилу: в знаковом разряде числа проставляется 1, а во всех остальных разрядах цифры заменяются на взаимно обратные.

Обратный код положительной двоичной дроби совпадает с прямым.

Аналитически:

$$[X]_{обр} = \begin{cases} X, & X \geq 0 \\ X + 10_{(2)} - 10^{-n}_{(2)}, & X < 0 \end{cases} \qquad (1.2.4)$$

$$X_1 = +0{,}110011 \qquad [X_1]_{обр} = 0{,}110011$$
$$X_2 = -0{,}110011 \qquad [X_2]_{обр} = 1{,}001100$$

Алгебраическое сложение чисел

Алгебраическое сложение всегда производится либо в дополнительном, либо в обратном коде. Сумма, естественно, также получается в одном из этих кодов соответственно. Правильный знак суммы получается автоматически.

В случае возникновения единицы переноса из знакового разряда суммы, её нужно отбросить при производстве сложения в дополнительном коде и прибавить к младшему разряду суммы при сложении в обратном коде.

Необходимо учитывать, что в поразрядном сложении участвуют не две, а три цифры. Третьим слагаемым будет единица переноса.

Таблица сложения

Текущий разряд первого слагаемого	Текущий разряд второго слагаемого	Цифра переноса из предыдущего разряда	Результат (текущий разряд и цифра переноса)
0	0	0	00
0	0	1	01
0	1	0	01
1	0	0	01
1	0	1	10
1	1	0	10
0	1	1	10
1	1	1	11

Сложение с фиксированной точкой

При алгебраическом сложении возможны 4 различных случая:

1. $X_1 > 0;\ X_2 > 0;\ X_1 + X_2 = X_3 > 0$

$X_1 = +0{,}1100$

$X_2 = +0{,}0010$

$[X_1]_{пр} = 0{,}1100; \quad [X_1]_{доп} = [X_1]_{пр}$

$[X_2]_{пр} = 0{,}0010; \quad [X_2]_{доп} = [X_2]_{пр}$

$$\begin{array}{r} 0{,}1100 \\ 0{,}0010 \\ \hline 0{,}1110 \end{array}$$

$X_3 = +0{,}1110$

2. $X_1 > 0;\ X_2 < 0;\ X_1 + X_2 = X_3 > 0$

$X_1 = +0{,}1100$

$X_2 = -0{,}0010$

$[X_1]_{пр} = 0{,}1100; \quad [X_1]_{доп} = [X_1]_{пр}$

$[X_2]_{пр} = 1{,}0010; \quad [X_2]_{доп} = 1{,}1110$

$[X_2]_{обр} = 1{,}1101$

	в доп. коде	*в обр. коде*
	0,1100	0,1100
	1,1110	1,1101
	1|0,1010	1|0,1001
		1
		0,1010

$X_3 = +0{,}1010$

3. $X_1 < 0;\ X_2 > 0;\ X_1 + X_2 = X_3 < 0$

$X_1 = -0{,}1100$

$X_2 = +0{,}0010$

$[X_1]_{пр} = 1{,}1100; \quad [X_1]_{обр} = 1{,}0011$

$[X_1]_{доп} = 1{,}0100$

	в доп. коде	*в обр. коде*
	1,0011	1,0100
	0,0010	0,0010
	1,0101	1,0110

$[X_2]_{пр} = 0{,}0010 = [X_2]_{обр} = [X_2]_{доп}$

$[X_3]_{обр} = 1{,}0101 \qquad [X_3]_{доп} = 1{,}0110$

$[X_3]_{пр} = 1{,}1010 \qquad [X_3]_{пр} = 1{,}1001$

$$\begin{array}{r} 1 \\ \hline 1{,}1010 \end{array}$$

$X_3 = -0{,}1010$

4. $X_1 < 0;\ X_2 < 0;\ X_1 + X_2 = X_3 < 0$

$X_1 = -0{,}1100$

$X_2 = -0{,}0010$

		в доп. коде	в обр. коде
$[X_1]_{пр} = 1{,}1100;$	$[X_1]_{обр} = 1{,}0011$	1,0100	1,0011
		0,1110	1,1101
	$[X_1]_{доп} = 1{,}0100$	1|1,0010	1|1,0000
			1
			1,0001

$[X_2]_{пр} = 1{,}0010 \qquad [X_2]_{обр} = 1{,}1101$

$[X_2]_{доп} = 1{,}1110$

$[X_3]_{обр} = 1{,}0001 \qquad [X_3]_{доп} = 1{,}0010$

$[X_3]_{пр} = 1{,}1110 \qquad [X_3]_{пр} = 1{,}1101$

1

1,1110

$X_3 = -0{,}1110$

При сложении любых положительных чисел, сумма которых > 1, результат их машинного суммирования будет иметь вид отрицательного числа, хотя никаких ошибок не было. В этом случае происходит переполнение разрядной сетки.

Необходимо, чтобы компьютер научился фиксировать такие ситуации. Для этого используются так называемые модифицированные коды. Модифицированные коды отличаются от обыкновенных тем, что в них знак числа изображаются двумя разрядами.

Знак + кодируется 00, а знак – кодируется 11. Комбинации 01 и 10 являются запрещенными.

Пусть имеются $[X_1]_{пр} = 0{,}10101$ и $[X_2]_{пр} = 1{,}10101$. Перевести их в модифицированный обратный код.

$$[X_1]^{мод}_{обр} = 00{,}10101$$

$$[X_2]^{мод}_{обр} = 11{,}01010$$

Пусть требуется сложить числа:

$X_1 = +0{,}1100$

$X_2 = +0{,}1010$

Переведем их в модифицированный дополнительный код и выполним требуемую операцию:

$[X_1]^{мод}_{доп} = 00{,}1100$ 00,1100

$[X_2]^{мод}_{доп} = 00{,}1010$ 00,1010

01,0110

Произошло переполнение разрядной сетки (здесь мы предполагаем, что длина разрядной сетки равно 4).

Пусть теперь необходимо сложить числа:

$X_1 = +0{,}1100$

$X_2 = -0{,}1010$

Имеем:

$$[X_1]^{мод}_{доп} = 00{,}1100 \qquad 00{,}1100$$

$$[X_2]^{мод}_{доп} = 11{,}1110 \qquad \underline{11{,}1110}$$

$$1\,|\,00{,}1010$$

$$X_3 = +0{,}1010$$

т.е. знак результата получился автоматически.

Сложение чисел с плавающей точкой.

Правило:

1) Уравниваются порядки слагаемых. Меньший порядок увеличивается до большего, а мантисса этого числа сдвигается вправо на соответствующее количество разрядов.
2) Производится преобразование мантисс в один из модифицированных кодов, дополнительный или обратный.
3) Производится сложение мантисс по правилам сложения чисел с фиксированной точкой.
4) Полученная сумма мантисс переводится в прямой код. К ней приписывается общий порядок слагаемых, производится нормализация суммы (в случае необходимости) и округление мантиссы.

Округление производится следующим образом: пусть используется n разрядов. В результате сложения мантисс получилось предположим $n+k$ разрядов. Прежде чем отбросить последние k разрядов, к $n+1$ разряду добавляется 1 (напомним, что речь идет о двоичном сложении), затем разряды с $n+1$ до $n+k$ отбрасываются.

Пример:

$$\begin{array}{l} 0{,}01011 \\ \underline{\quad\quad 1} \\ 0{,}0110\,|\,0 \end{array} \qquad \begin{array}{l} 0{,}01010 \\ \underline{\quad\quad 1} \\ 0{,}0101\,|\,0 \end{array}$$

Для лучшего уяснения рассмотрим операцию сложения с плавающей точкой сначала для десятичных чисел:

$$X_1 = 0{,}125 \cdot 10^3$$

$$X_2 = 0{,}625 \cdot 10^2 = 0{,}0625 \cdot 10^3$$

$$\begin{array}{l} 0{,}125 \\ + \\ \underline{0{,}0625} \\ 0{,}1875 \end{array}$$

$$0{,}1875 \cdot 10^3 \cong 0{,}188 \cdot 10^3$$

Пример нормализации:

$$0{,}0256 \cdot 10^3 = 0{,}256 \cdot 10^2$$

$$0{,}625 \cdot 10^3$$

$$+$$

$$\frac{0{,}600 \cdot 10^3}{1{,}225 \cdot 10^3}$$

$$1{,}225 \cdot 10^3 = 0{,}1225 \cdot 10^4$$

Теперь рассмотрим в двоичной системе счисления:

$$[X_1]_{пр} = 0\ 1100\ 0\ 0100 = 0\ 001100\ 0\ 110$$

$$[X_2]_{пр} = 0\ 0010\ 0\ 110$$

$$[m_1]_{доп}^{мод} = 00\ 001100 \quad (m - \textit{мантисса})$$

$$[m_2]_{доп}^{мод} = 00\ 001000$$

$$[m_1]_{доп}^{мод} + [m_2]_{доп}^{мод} = \begin{array}{r} 00\ 001100 \\ \underline{00\ 001000} \\ 00\ 010100 \end{array}$$

$$[m_3]_{доп}^{мод} = 00\ 010100 = [m_3]_{пр}^{мод}$$

нормализация:

$$[X_3]_{пр} = 0\ 010100\ 0\ 0110 = 0\ 10100\ 0\ 0101$$

округление:

$$\begin{array}{r} 0{,}10100 \\ \underline{\qquad\quad 1} \\ 0{,}10101 \end{array}$$

$$[X_3]_{пр} = 0\ 1010\ 0\ 0101$$

Умножение чисел с фиксированной точкой

Умножение чисел с фиксированной точкой производится в два этапа:

1) определяется знак произведения при помощи сложения по модулю 2 (mod 2). Суммирование по mod 2 производится согласно правилам двоичной арифметики, но без учета единицы переноса. Правомерность такого определения знака доказывается с помощью таблицы:

$(+)\cdot(+) = (+)$	$0 \oplus 0 = 0$
$(+)\cdot(-) = (-)$	$0 \oplus 1 = 1$
$(-)\cdot(+) = (-)$	$1 \oplus 0 = 1$
$(-)\cdot(-) = (+)$	$1 \oplus 1 = 0$

2) производится перемножение модулей сомножителей, затем в случае необходимости округление и к полученному результату приписывается знак.

Умножение производится по обычным правилам арифметики согласно двоичной таблице умножения. Пример:

$[X_1]_{пр} = 0{,}1010$

$[X_2]_{пр} = 1{,}1101$

1. определяем знак произведения

$0 \oplus 1 = 1$

2. выполняем умножение

$$\begin{array}{r} 0{,}1010 \\ 0{,}1101 \\ \hline 1010 \\ 0000 \\ 1010 \\ 1010 \\ \hline 0{,}10000010 \end{array}$$

$[X_3]_{пр} = 1{,}1000$

Умножение чисел с плавающей точкой

$X_1 = m_1 \cdot 10^{P_1}$

$X_2 = m_2 \cdot 10^{P_2}$

$X_3 = X_1 \cdot X_2 = (m_1 \cdot m_2) \cdot 10^{P_1 + P_2}$

Отсюда 4 этапа для умножения чисел с плавающей точкой:

1. определение знака произведения путем сложения знаковых разрядов по mod 2.
2. перемножение модулей мантисс сомножителей по правилам умножения чисел с фиксированной точкой.
3. сложение порядков по правилам алгебраического сложения чисел с фиксированной точкой.
4. округление полученного результата и приписывание соответствующего знака результату.

Пример:

$[X_1]_{пр} = 1\ 1010\ 0\ 101$

$[X_2]_{пр} = 0\ 1011\ 1\ 011$

1) $1 \oplus 0 = 1$

2)

$$\begin{array}{r} 0{,}1010 \\ \underline{0{,}1011} \\ 1010 \\ 1010 \\ 0000 \\ \underline{1010} \\ 0{,}01101110 \end{array}$$

3) $[P_1]^{мод}_{пр} = 00{,}101$

$[P_2]^{мод}_{пр} = 11{,}011$

$[P_2]^{мод}_{доп} = 11{,}101$

$$\begin{array}{r} 00\ 101 \\ \underline{11\ 101} \\ 1|\ 00\ 010 \end{array}$$

$[P_3]_{пр} = 0\ 010$

4) $[X_3]_{пр} = 1{,}01101110\ 0\ 010$

округление:

$[X_3]_{пр} = 1{,}0111\ 0\ 010$

Деление чисел с фиксированной точкой

Деление чисел с фиксированной точкой осуществляется в два этапа:

1. определение знака
2. деление

$X_1 = +0{,}101$

$X_2 = -0{,}110$

$[X_1]_{пр} = 0{,}101$

$[X_2]_{пр} = 1{,}110$

1) $0 \oplus 1 = 1$

2) $0,1010 \mid 0,110$

$110 \mid 0,1101 \equiv 0,111$

1000

110

1000

110

10

$[X_3]_{пр} = 1,111$

Деление чисел с плавающей точкой

$X_1 = m_1 \cdot 10^{P_1}$

$X_2 = m_2 \cdot 10^{P_2}$

$X_3 = X_1 : X_2 = (m_1 : m_2) \cdot 10^{P_1 - P_2}$

Правила деления:

1) определение знака частного посредством сложения знаковых цифр мантисс операндов по mod 2.
2) деление модуля мантиссы делимого на модуль мантиссы делителя по правилам деления чисел с фиксированной точкой.
3) нахождение порядка частного путем вычитания из порядка делимого порядка делителя с учетом их знаков.
4) нормализация результата, если это необходимо.

$X_1 = 0,101\ 0\ 110$

$X_2 = 1,110\ 0\ 011$

1) $0 \oplus 1 = 1$

2) $0,1010 \mid 0,110$

$110 \mid 0,1101 \equiv 0,111$

1000

110

1000

110

10

$|m_3| = 0,111$

3) $[P_1]_{доп}^{мод} = 00,110$

$[P_2]_{доп}^{мод} = 11,101$

$00,110$

$11,101$

$1 \mid 00,011$

$[P_3]_{пр} = 0,011$

$[X_3]_{пр} = 1\ 111\ 0\ 011$

§1.3. *Логические основы вычислительной техники*

Как известно, логика – это наука о законах и формах мышления. Математическая логика занимается изучением возможностей применения формальных математических методов для решения логических задач. В вычислительной технике используется главным образом начальный раздел математической логики – исчисление высказываний или алгебра логики. Иногда её ещё называют булевой алгеброй по имени английского математика Джорджа Буля, который ещё в XIX веке разработал основные положения исчисления высказываний.

Однако сам Буль не видел путей практической реализации своих исследований. Но уже в 40-х годах XX века алгебра логики нашла широкое применение при проектировании и синтезе сложных переключательных схем в автоматических телефонных станциях, а затем и в вычислительной технике.

Начальным понятием алгебры логики является высказывание. Под высказыванием понимается любое утверждение, которое оценивается только с точки зрения его истинности или ложности. То есть высказывание может быть либо истинным, либо ложным.

В соответствии с такой двоичной природой высказываний условились называть их логическими двоичными переменными. Любое высказывание можно обозначить символом x и считать, что x=1, если высказывание истинно, x=0, если высказывание ложно.

Таким образом, логическая (булевая) переменная – это такая величина x, которая может принимать только два значения (0 или 1):

x={0, 1}

Высказывание абсолютно истинно, если соответствующая ей логическая величина принимает значение x=1 при любых условиях.

Высказывание абсолютно ложно, если соответствующая ей логическая переменная принимает значение 0 при любых условиях.

Сложным называется высказывание, значение истинности которого зависит от значение истинности других высказываний. Следовательно, любое сложное высказывание можно считать функцией некоторых двоичных аргументов – простых высказываний, входящих в его состав.

Логическая функция (функция алгебры логики) – функция $f(x_1, x_2, \ldots, x_n)$ принимающая значение 0 или 1 на наборе логических переменных $x_1, x_2, \ldots, x_n$. Причем $x_1, x_2, \ldots, x_n$ могут быть в свою очередь логическими функциями. Отсюда следует, что можно построить функцию любой наперед заданной сложности, пользуясь принципом суперпозиции.

Ограниченный набор функций, обеспечивающий на основе суперпозиции построение функции любой наперед заданной сложности, называется функционально полной системой функций.

Из множества функционально полных систем рассмотрим один, получивший наибольшее распространение и вследствие этого названного

основным.

В его состав входят всего 3 функции:

I. функция отрицания (функция НЕ, инверсия).

Отрицанием называется такая функция y, которая истинна тогда и только тогда, когда ложен аргумент x и наоборот, y ложна тогда и только тогда, когда аргумент x истинен.

В алгебре логики любые функции удобно изображать в виде таблицы соответствия всех возможных комбинаций значений истинности входных переменных (аргументов) и выходной логической функции.

аргумент x	0	1
значение функции y	1	0

При помощи логико-математической символики функция отрицания записывается в виде:

$y = \overline{x}$, $\neg x$, NOT

=

Эта функция обладает свойством $x = x$

II. Конъюнкция (логическое умножение, функция "И")

Конъюнкцией n высказываний называется такое сложное высказывание y, которое истинно тогда и только тогда, когда истинны все исходные высказывания; во всех остальных случаях y ложно. Таблица истинности для функции двух переменных:

x_1	0	0	1	1
x_2	0	1	0	1
y	0	0	0	1

Обозначения:

$y = x_1 \cdot x_2$

$y = x_1 \& x_2$

AND

Конъюнкция обладает 4-мя очевидными свойствами:

1. $x \cdot x \cdot \cdots \cdot x = x$

результат многократного логического умножения аргумента x на себя равен исходному.

2. $x \cdot 1 = x$

логическое произведение двух аргументов, один из которых абсолютно истинен, равно второму аргументу.

3. $x \cdot 0 = 0$

логическое произведение двух аргументов, один из которых абсолютно ложен, равно 0.

4. $x \cdot \bar{x} = 0$

логическое произведение некоторого аргумента и его отрицания равно 0.

III. дизъюнкция (логическое сложение, функция "ИЛИ")

Дизъюнкцией n высказываний называется такое сложное высказывание y, которое ложно тогда и только тогда, когда ложны все исходные высказывания. Если же истинно хотя бы одно исходное высказывание, функция y тоже истинно.

Таблица истинности для функции двух переменных:

x_1	0	0	1	1
x_2	0	1	0	1
y	0	1	1	1

Обозначения:

$y = x_1 + x_2$

$y = x_1 \vee x_2$

OR

Дизъюнкция обладает 4-мя очевидными свойствами:

1. $x + x + \cdots + x = x$

результат многократного логического сложения аргумента x на себя равен исходному.

2. $x + 1 = 1$

логическое сложение двух аргументов, один из которых абсолютно истинен, равно 1.

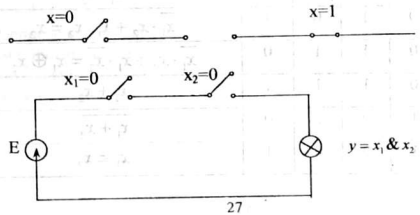
3. $x + 0 = x$

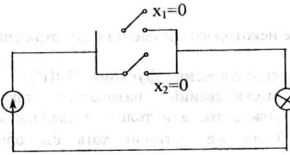
логическое сложение двух аргументов, один из которых абсолютно ложен, равно второму аргументу.

4. $x + \bar{x} = 1$

логическое сложение некоторого аргумента и его отрицания равно 1.

Рассмотренные выше функции можно элементарным образом реализовать с помощью электрических схем.





Абсолютно истинную функцию называют константа 1, абсолютно ложную – константой 0.

Функция $f(x) = x$, т.е. функция повторяющая значение аргумента, называется тождественной.

Существуют и другие функции от двух логических переменных. Их всего 16. См. таблицу 1.2.

Функция $f_9 = \overline{x_1 + x_2}$ называется функцией Пирса. Это функция, которая истинна тогда и только тогда, когда оба x_1 и x_2 ложны.

Функция $f_{15} = \overline{x_1 \cdot x_2}$ называется функцией Шеффера. Это функция, которая ложна тогда и только тогда, когда оба x_1 и x_2 истинны.

$f_9 = x_1 \downarrow x_2$ - стрелка Пирса

$f_{15} = x_1 | x_2$ - штрих Шеффера

Табл. 2

Функция	Значения x_1 и x_2				Аналитическое представление функции
	00	01	10	11	
f_1	0	0	0	0	0
f_2	0	0	0	1	$x_1 \cdot x_2$
f_3	0	0	1	0	$x_1 \cdot \overline{x_2}$
f_4	0	0	1	1	x_1
f_5	0	1	0	0	$\overline{x_1} \cdot x_2$
f_6	0	1	0	1	$\overline{x_1} \cdot x_2 + x_1 \cdot x_2 = x_2$
f_7	0	1	1	0	$\overline{x_1} \cdot x_2 + x_1 \cdot \overline{x_2} = x_1 \oplus x_2$
f_8	0	1	1	1	$x_1 + x_2$
f_9	1	0	0	0	$\overline{x_1 + x_2}$
f_{10}	1	0	0	1	$x_1 \equiv x_2$

f_{11}	1	0	1	0	$\overline{x_1} \cdot \overline{x_2} + x_1 \cdot \overline{x_2}$
f_{12}	1	0	1	1	$\overline{x_1} \cdot \overline{x_2} + x_1 \cdot \overline{x_2} + x_1 \cdot x_2$
f_{13}	1	1	0	0	$\overline{x_1} \cdot \overline{x_2} + \overline{x_1} \cdot x_2 = \overline{x_1}$
f_{14}	1	1	0	1	$x_1 \to x_2$
f_{15}	1	1	1	0	$\overline{x_1 \cdot x_2}$
f_{16}	1	1	1	1	1

Законы алгебры логики

В алгебре логики имеются 4 основных закона, регламентирующие порядок выполнения операций НЕ, И, ИЛИ в любом логическом выражении:

1. переместительный (коммутативный);
2. сочетательный (ассоциативный);
3. распределительный (дистрибутивный);
4. инверсии (правило Де Моргана).

Рассмотрим эти законы:

Переместительный закон

Справедлив как для операции И, так и для операции ИЛИ.

1. $x_1 + x_2 = x_2 + x_1$
2. $x_1 \cdot x_2 = x_2 \cdot x_1$

Сочетательный закон

Также справедлив как для операции И, так и для операции ИЛИ.

1. $x_1 + x_2 + x_3 = x_1 + (x_2 + x_3) = x_2 + (x_1 + x_3) = x_3 + (x_1 + x_2)$
2. $x_1 \cdot x_2 \cdot x_3 = x_1 \cdot (x_2 \cdot x_3) = x_2 \cdot (x_1 \cdot x_3) = x_3 \cdot (x_1 \cdot x_2)$

Распределительный закон

1. Распределительный закон первого рода

Произведение суммы нескольких аргументов на какую – либо переменную равно сумме произведений.

$(x_1 + x_2) \cdot x_3 = x_1 \cdot x_3 + x_2 \cdot x_3$

2. Распределительный закон второго рода

Сумма произведения нескольких аргументов и какой – либо переменной равна произведению суммы каждого сомножителя и этой переменной.

$x_2 \cdot x_3 + x_1 = (x_1 + x_2) \cdot (x_1 + x_3)$

Этот закон выглядит несколько парадоксальным. Докажем его:

$$(x_1 + x_2)\cdot(x_1 + x_3) = x_1 \cdot x_1 + x_2 \cdot x_1 + x_1 \cdot x_3 + x_2 \cdot x_3 =$$
$$= x_1 + x_1 \cdot x_2 + x_1 \cdot x_3 + x_2 \cdot x_3 = x_1 \cdot (1 + x_2 + x_3) + x_2 \cdot x_3 =$$
$$= x_1 \cdot 1 + x_2 \cdot x_3 = x_1 + x_2 \cdot x_3 = x_2 \cdot x_3 + x_1$$

Законы инверсии

1. Отрицание логической суммы нескольких аргументов равно логическому произведению отрицаний этих же аргументов.

$$\overline{x_1 + x_2 + \ldots + x_n} = \overline{x_1} \cdot \overline{x_1} \cdot \ldots \cdot \overline{x_n}$$

2. Отрицание логического произведения нескольких аргументов равно логической сумме отрицаний этих же аргументов.

$$\overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n} = \overline{x_1} + \overline{x_1} + \ldots + \overline{x_n}$$

Следствия из законов алгебры логики

Логическим выражением называют всякую совокупность логических переменных и знаков операций, указывающих какие действия над этими переменными нужно выполнить. Будем считать отрицание логическим действием первой ступени (старшей логической операцией), конъюнкцию – логическим действием второй ступени, дизъюнкцию – логическим действием третьей ступени.

Тогда можно определить правило выполнения логических операций:

Если в логическом выражении встречаются только операции одной и той же ступени, то их нужно выполнять в том порядке, в каком они записаны (т.е. слева направо).

Если же в логическом выражении встречаются действия различных ступеней, то сначала выполняются действия первой ступени, затем второй и только после этого третьей.

Всякое отклонение от этого порядка должно регламентироваться скобками.

Пример. Вычислить значение логического выражения

$$y = x_1 \cdot \overline{x_2} + x_3 \cdot \overline{\overline{x_4} + x_5}$$

при $x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 0,\ x_5 = 1$

$$y = 1 \cdot 1 + 1 \cdot \overline{1 + 1} = 1 + 1 \cdot 0 = 1 + 0 = 1$$

Контрольные вопросы:

1. Формы представления информации.
2. Что такое система счисления?
3. В чем различие между позиционной и непозиционной системами счисления?
4. Перечислите наиболее распространенные системы счисления.
5. Правила перевода чисел из одной системы счисления в другую.
6. Формы представления чисел в памяти компьютера.
7. Дополнительный и обратный коды двоичных чисел.
8. Алгебраическое сложение двоичных чисел.
9. Умножение двоичных чисел.
10. Деление двоичных чисел.
11. Логические переменные. Логические функции.
12. Основная система логических функций ("И","ИЛИ", "НЕ").
13. Законы алгебры логики.
14. Правило выполнения логических операций.

ГЛАВА II. ТЕХНИЧЕСКИЕ СРЕДСТВА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

§2.1. Краткая история развития средств вычислительной техники

Многие математики и механики прошлого пытались сконструировать различные вычислительные устройства, приборы и машины для облегчения вычислений. Во многих странах с древнейших времен были известны счеты.

В 1617 году Джон Непер описал специальные палочки для быстрого умножения и деления на счетах.

В 1642 году Блез Паскаль изобрел суммирующую машину.

В 1694 году Лейбниц создал машину, которая могла не только складывать, но и умножать.

Идею построения вычислительной машины похожей на современную, впервые высказал ещё в XIX веке английский математик Чарльз Бэббидж. В 1833 году им был предложен проект аналитической машины, которая должна была выполнять элементарные математические операции, запоминать исходные, промежуточные и окончательные результаты. Его машина состояла из "склада" для хранения чисел и "мельницы" для выполнения действий над числами. Из-за невысокого уровня развития тогдашней техники, Ч. Бэббидж не смог собрать эту машину.

В 1937 г. американский ученый Айткен заново разработал принципы построения цифровых вычислительных машин с автоматическим ходом вычислений, не зная о том, что это давно было сделано Ч. Бэббиджем.

В 1944 г. вступила в строй первая релейно-механическая цифровая вычислительная машина "МАРК – I". Постройка следующей машины "МАРК – II" на реле улучшенных конструкций показала, что надежность релейных машин крайне низка.

В начале 40-х годов Клод Шеннон предложил для синтеза релейных схем использовать аппарат математической логики (булевой алгебры). Это дало возможность формальным способом строить экономичные схемы. Вследствие недостатков электромеханических реле конструкторы решают использовать электронные реле – триггеры. Это во много раз повысило надежность вычислительных машин.

Первая электронно-цифровая вычислительная машина была разработана в США в 1946 году и называлась "ЭНИАК". Она содержала 19000 электронных ламп. Скорость 5000 сложений в секунду.

Этот момент явился точкой отсчета пути по которому пошло развитие вычислительной техники.

Развитие вычислительной техники определялось развитием электроники, появлением новых элементов и принципов их действия, т.е. развитием элементной базы. В настоящее время можно выделить 4 поколения компьютеров. Причем компьютеры четвертого поколения

отличаются от предшествующих не только элементной базой, но и структурой построения, а также способом общения с человеком и эта тенденция сохранится и в будущем.

Под поколением компьютеров понимаются все типы и модели компьютеров, разработанные хотя и различными конструкторскими коллективами, но построенные на одних и тех же научных принципах и элементной базе. Появление каждого нового поколения определялось тем, что появлялись новые электронные элементы, технология изготовления которых принципиально отличалась от предыдущего поколения.

Первое поколение (1946 – середина 50-х годов).

Элементная база – электронные лампы. В ЭВМ "ЭНИАК" было 19000 ламп, из которых ежемесячно заменяли 5000.

В Советском Союзе первая вычислительная машина была создана в 1951 году и называлась МЭСМ.

Второе поколение (середина 50-х – середина 60-х годов)

Элементная база – полупроводниковые приборы (транзисторы, диоды). Значительно улучшились характеристики компьютеров (быстродействие, надежность, габариты, условия эксплуатации).

Третье поколение (середина 60-х – середина 70-х годов)

В качестве элементной базы используется интегральные схемы (ИС). ИС выполняет те же функции, что и аналогичная схема на полупроводниках, но при этом существенно меньше по размерам, увеличилось надежность и быстродействие. Более точно, ИС – это электронная схема, реализующая определенную функцию, технически выполненная на одном кристалле кремния площадью всего несколько квадратных миллиметров.

Первый компьютер на ИС – широко известная IBM/360.

Четвертое поколение (середина 70-х – по настоящее время)

Компьютеры создаются на основе БИС (больших интегральных схем) со степенью интеграции от сотен тысяч до нескольких миллионов элементов на одном кристалле.

В виде БИС удалось создать процессор компьютера, который получил название микропроцессор (МП). МП повлияли на структуру и конструкцию компьютеров. Размеры компьютеров значительно уменьшились, появились персональные компьютеры (ПК).

§2.2. *Логические элементы вычислительной техники, реализующие основные логические функции.*

Логическими элементами называются такие электронные устройства, в которых между входными и выходными сигналами реализуются те или иные логические операции. Логическим элементом называется устройство, реализующее только одну булеву функцию.

I. Схема отрицания (элемент НЕ, инвертор)

Как следует из определения функции НЕ, единичный сигнал на выходе схемы отрицания должен появиться при наличии на её входе нулевого сигнала и наоборот, на выходе схемы должен появиться нулевой сигнал, если на входе действует сигнал, соответствующий логической единице.

Функциональная схема имеет вид:

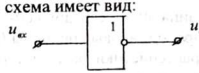


Рис. 2.1. Функциональная схема элемента НЕ

Слева вход, справа выход. На выходной стороне прямоугольника проставляется о означающий инверсию. На практике чаще всего элемент НЕ реализуется при помощи однокаскадного усилителя постоянного тока, собранного на транзисторе. Схема электрическая принципиальная имеет вид:

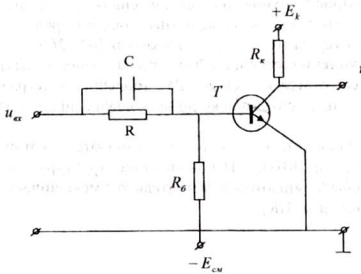


Рис. 2.2. Схема электрическая принципиальная

При этом используется свойство любого линейного усилительного каскада изменять у входного переменного сигнала фазу на 180^0. В логических схемах все электронные приборы (транзистор в нашем случае) работают в ключевом режиме. При поступлении на вход схемы низкого уровня напряжения, транзистор (n-p-n типа) остается закрытым и на выходе

будет $u_{вых} = +E_k$ означающий "1". Если на вход поступает высокий уровень сигнала ("1"), то транзистор откроется и на выходе будет низкий уровень напряжения $u_{вых} = E_k - I_k R_k$, означающий логический "0".

О том, как работает транзистор и другие элементы этой схемы вы сможете узнать из соответствующих курсов физики, ТОЭ и электроники. Здесь за неимением места мы эти подробности опускаем. Для целей нашего курса достаточно уяснить, что между входным и выходным напряжениями схемы существует такая взаимосвязь, которая и реализует логическую функцию НЕ.

II. Схема совпадения (схема "И")

Единичный сигнал на выходе схемы должен появляться только тогда, когда на всех входах "1".

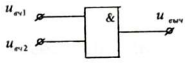


Рис. 2.3. Функциональная схема

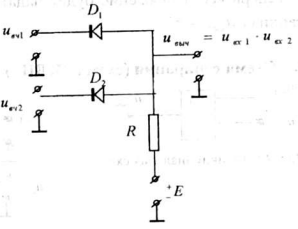


Рис. 2.4. Схема электрическая принципиальная

Электрическая схема реализована на полупроводниковых диодах, работающих в ключевом режиме (т.е. сопротивление открытого диода $r_{д.о.} \to 0$, а сопротивление закрытого диода $r_{д.з.} \to \infty$). Резистор R выбирается из условия:

$$r_{д.о.} << R << r_{д.з.}$$

Выходное напряжение снимается с двух параллельно включенных диодов, сопротивления которых совместно с резистором R образует обыкновенный делитель напряжения. Возможны 3 случая:

1) Оба входных напряжения низкие

$$u_{вх1} = u_{вх2} = 0$$

и следовательно оба диода открыты. Тогда общее сопротивление диодов:

$$r = r_{д.o1} \| r_{д.o1} = 0$$

$$R >> r$$

Практически все напряжение E падает на R, а на выходе низкий

потенциал $u_{вых} = 0$.

2) Один $u_{вх}$ низкий, а другой высокий. Пусть $u_{вх1}$ низкий. Следовательно:

$$r_{д.о1} = 0, \quad r_{д.о2} \to \infty$$

$$r = r_{д.о1} \| r_{д.о2} = 0$$

$$R >> r$$

Все напряжение E падает на R. На выходе низкий потенциал.

3) Оба входных сигнала высокие. Тогда:

$$r_{д.з1} \to \infty, \quad r_{д.з2} \to \infty$$

$$r \to \infty$$

$$R << r$$

Теперь все напряжение будет падать на r и на выходе будет высокий потенциал $u_{вых} = E$.

III. Схема собирания (схема "ИЛИ")

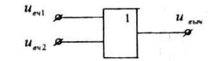


Рис. 2.5. Функциональная схема

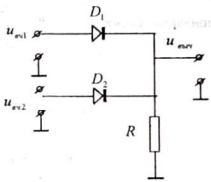


Рис. 2.6. Схема электрическая принципиальная

$$r_{д.о} << R << r_{д.з}.$$

Анализ работы схемы производится аналогично предыдущей схеме.

IV. Схема И – НЕ

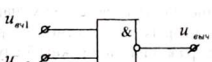


Рис. 2.7. Функциональная схема

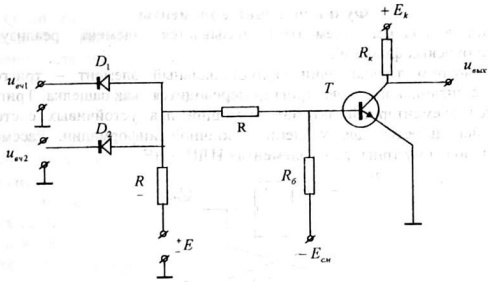


Рис. 2.8. Схема электрическая принципиальная

V. Схема ИЛИ – НЕ

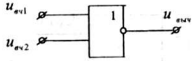


Рис. 2.9. Функциональная схема

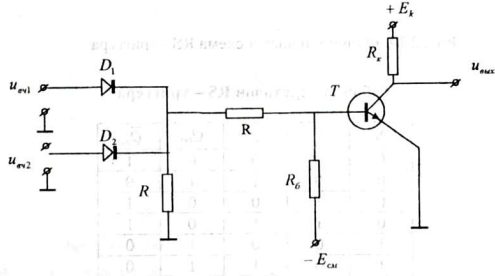


Рис. 2.10. Схема электрическая принципиальная

Функциональные элементы

Функциональным элементом называется элемент реализующий несколько булевых функций.

Рассмотрим только один функциональный элемент – триггер. В переводе с английского слово триггер переводится как защелка. Триггером называется элементарный автомат имеющий два устойчивых состояния. Триггер используется для хранения двоичной информации. Рассмотрим структурную схему триггера на элементах ИЛИ – НЕ:

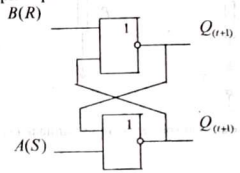


Рис. 2.11. Структурная схема триггера на элементах ИЛИ – НЕ

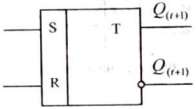


Рис. 2.12. Функциональная схема RS - триггера

Таблица переходов RS – триггера

S_t	R_t	Q_t	Q_{t+1}	$\overline{Q}_{t+1}$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	-	-
1	1	1	-	-

Принято считать, что триггер перешел в единичное состояние, если на выходе Q появился потенциальный сигнал низкого уровня. Соответственно за переход в нулевое состояние триггера принято считать появление на

выходе Q сигнала высокого уровня. Для передачи единичного сигнала из триггера необходимо изменить его состояние с 1 на 0 и получить на выходе Q изменение потенциального сигнала с низкого на высокий.

Типовые функциональные схемы

Типовые функциональные схемы строятся на основе логических и функциональных элементов. К ним относятся:

Регистры.

Регистром называется устройство, предназначенное для временного хранения двоичной информации. Каждый разряд регистра используется для ввода, вывода и хранения одного разряда двоичного числа. Рассмотрим, например, схему 3-х разрядного регистра:

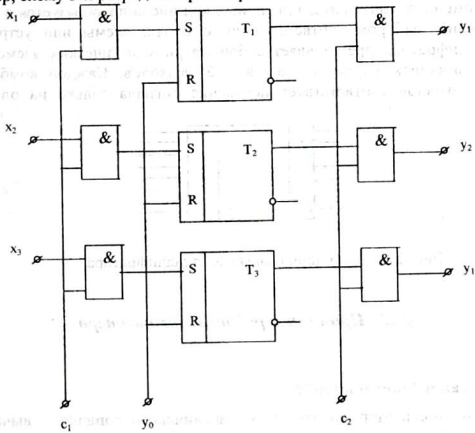


Рис. 2.13. Схема 3-х разрядного регистра

В начале работы все триггеры устанавливаются в 0 сигналом y_0. Конъюнкторы на входе пропустят двоичную информацию на входы S триггеров только при наличии синхроимпульса c_1. Для передачи двоичного числа из регистра на входы конъюнкторов необходимо на R входы триггеров подать импульс y_0 который перебросит триггеры из 1→0. При переходе из 1→0 на входы конъюнкторов поступят сигналы передачи 1. При этом они пройдут в другие устройства только при наличии синхроимпульса c_2.

Счетчики.

Назначением счетчиков является:

подсчет числа импульсов, поступающих на вход счетчика

временное хранение каждого состояния

преобразование непрерывного сигнала, заданного последовательностью импульсов в двоичный код

Счетчики бывают прямого счета (суммирующие), вычитающие, реверсивные.

Дешифраторы

Дешифратором называется устройство, позволяющее преобразовать сочетание входных сигналов (входной код) в сигнал только на одном из выходов.

С помощью дешифратора двоичный код числа преобразуется в сигнал, управляющий выбором соответствующего блока, схемы или устройства. Схема дешифратора представляет собой матрицу логических элементов и содержит в общем случае n входов и 2^n выходов. Каждой комбинации входных сигналов соответствует появление сигнала только на одном из выходов.

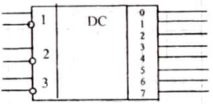


Рис. 2.14. Функциональная схема дешифратора

§2.3. Принципы работы компьютера

Что такое "компьютер"?

Слово компьютер произошло от английского compute – вычислять. Ранее в нашей стране применялся термин "электронно-вычислительная машина" (ЭВМ), что, на мой взгляд, более точно отражает суть дела. Дело в том, что в основе любого современного компьютера лежат электронные устройства, о которых мы говорили выше.

Автоматизация работ с информацией (данными) имеет свои особенности и отличия от автоматизации других видов работ. Для этого класса задач используют особые устройства, большинство из которых являются электронными приборами и устройствами. Совокупность устройств, предназначенных для автоматической или автоматизированной обработки данных, называют *вычислительной техникой*.

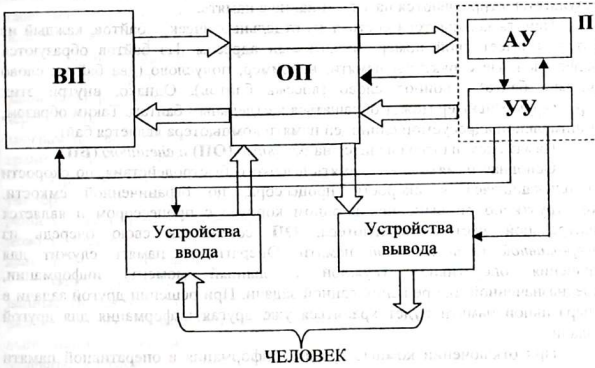
Конкретный набор взаимодействующих между собой устройств и программ называют *вычислительной системой*. Центральным устройством любой вычислительной системы является *компьютер*.

Итак, компьютер – это электронный прибор, предназначенный для автоматизации создания, хранения, обработки и передачи данных.

Простейшая структура компьютера

Принципы работы всех компьютеров одинаковы и носят название *принципов фон – Неймана*.

Общность принципов работы компьютеров и определяет общность структур компьютеров. Рассмотрим простейшую структуру компьютера лишь с целью уяснения основных принципов работы компьютера.



Здесь двойные стрелки означают передачу информации (данных), одинарные стрелки означают управляющие сигналы процессора. На рисунке процессор обозначен буквой **П**. Процессор – это устройство, которое управляет всеми остальными устройствами компьютера и выполняет арифметические и логические операции над данными, находящимися в памяти.

Ту часть процессора, которая предназначена для непосредственного выполнения операций называют *арифметическим устройством* (**АУ**). Остальные функции процессора, как уже было сказано, – это управление работой различных устройств. Поэтому часть процессора, выполняющий эти

функции, называют *устройством управления* (**УУ**).

Память компьютера предназначена для хранения информации и команд программы. Информация, хранящаяся в памяти компьютера, представляет собой закодированные с помощью всего двух цифр 0 и 1 числа, буквы, символы, слова.

Команды, хранящиеся в памяти – это закодированные с помощью 0 и 1 инструкции компьютеру по выполнению определенных действий. Например, командой может служить предписание компьютеру сложить два числа, выдать сообщение на экран и т.п. Совокупность команд образует программу, которая и определяет последовательность действий компьютера.

Стрелка от **ОП** к **П** отражает два факта:

1) Процессор в своей работе руководствуется командами программы, выбираемых из памяти .

2) Аргументы выполняемых операций тоже выбираются из памяти.

Стрелка **П** →**ОП** отражает тот факт, что получаемые в процессоре результаты отправляются на запоминание в память.

Память компьютера состоит из отдельных ячеек – байтов, каждый из которых имеет свой номер, называемый адресом. Из байтов образуются более сложные структуры памяти, например, полуслово (два байта), слово (четыре байта), двойное слово (восемь байтов). Однако, внутри этих структур, компьютер может обращаться к отдельным байтам. Таким образом, минимальной адресуемой единицей памяти компьютера является байт.

Память компьютера делится на *основную* (**ОП**) и *внешнюю* (**ВП**).

Основная память – это память высокого быстродействия, по скорости приближающаяся к скорости процессора, но ограниченной емкости. Конструктивно он выполнен в одном корпусе с процессором и является центральной частью компьютера. **ОП** состоит в свою очередь из *оперативной* и *постоянной* памяти. Оперативная память служит для хранения оперативной (нужной в данный момент) информации, предназначенной для решения данной задачи. При решении другой задачи в оперативной памяти будет храниться уже другая информация для другой задачи.

При отключении компьютера вся информация в оперативной памяти теряется.

Постоянная память (**ПЗУ** – постоянное запоминающее устройство) предназначена для хранения постоянной (но часто нужной) информации, которая не зависит от того, какая задача решается. Чаще всего в ней хранится служебная информация, необходимая процессору. При отключении компьютера информация в постоянной памяти сохраняется. Кроме того, информацию в постоянной памяти нельзя стирать, изменять, дополнять.

Внешняя память (**ВП**) предназначена для долговременного хранения информации независимо от того, работает компьютер или нет и характеризуется очень большим объемом. Информацию во внешней памяти можно изменять, дополнять, уничтожать и т.д.

Для ввода данных с внешних носителей в **ОП** служат устройства *ввода*. Результаты решения задачи запоминаются в памяти компьютера.

Для выдачи результатов, хранящихся в памяти служат устройства *вывода*.

Принципы фон - Неймана

1. Принцип произвольного доступа к ячейкам памяти.

Компьютер на каждом такте своей работы может с одинаковым успехом обращаться к любым ячейкам памяти – как для чтения, так и для записи. Для обеспечения такого произвольного доступа к памяти каждой ячейке дается индивидуальное имя – эти имена и используются для указания нужных ячеек. Как правило, все ячейки пронумерованы от 0 до N-1, где N – общее число ячеек памяти и в качестве имени ячейки принимается ее порядковый номер, который называют адресом ячейки, а также адресом ее содержимого (по аналогии с обычным почтовым адресом). Этот принцип обеспечивает большую свободу и простоту использования информации в памяти компьютера.

2. Принцип хранимой программы.

Компьютер выполняет программу, которая представлена в цифровой (двоичной) форме и хранится в той же памяти, что и данные, подлежащие переработке по этой программе. Перед началом решения задачи программа вводится (записывается) в отведенные для нее ячейки памяти и хранится там во время выполнения этой программы компьютером.

Принцип хранимой программы прежде всего делает компьютер универсальным в том смысле, что на одном и том же компьютере можно решать самые разные задачи, поскольку для решения каждой из них можно составить программу, ввести ее в память и заставить компьютер выполнить эту программу.

Факт представления программы в такой же форме, что и другие данные (например, числа) и хранения программы в памяти открывает целый ряд и других возможностей. Например, программа в процессе своего выполнения также может подвергаться переработке, что позволяет задавать в самой программе правила получения некоторых ее частей и тем самым получать весьма компактные программы. Более того, одна программа может является результатом работы другой программы, что открывает принципиальную возможность автоматизировать процесс составления программы, поручив эту работу самой машине путем выполнения специальных программ.

Машинные операции.

Каждый тип компьютера имеет свой набор машинных операций, т.е. элементарных действий (с точки зрения программиста) по преобразованию информации. Например имеется операция сложения двух чисел. Причем

сложение с фиксированной точкой и сложение с плавающей представляют собой две разные операции. Решение любой задачи должно быть сведено к выполнению последовательности операций из этого набора.

С точки зрения удобства использования желательно, чтобы набор операций был возможно шире и чтобы он включал достаточно содержательные операции. С точки же зрения простоты реализации желательно, чтобы этот набор был невелик и чтобы в него входили лишь простейшие операции, т.к. это позволяет сделать компьютер более простым, дешевым и надежным. Эти два требования противоречивы и на практике выбирается некоторый компромиссный вариант.

В каждой ячейке памяти хранится упорядоченная последовательность символов, которая называется *машинным словом* или просто словом.

Машинное слово может, например, представляет собой отдельное число, изображенное с достаточно высокой степенью точности, так что за одно обращение к памяти можно выбрать из нее сразу все число, являющийся одним из аргументов арифметической операции или запомнить результат выполнения этой операции.

Машинное слово может представлять собой и отдельную команду программы. Команда – это предписание компьютеру выполнить одну операцию из ее набора операций. Компьютер предназначен для выполнения не какого-нибудь одного, а многих разных алгоритмов. Это обстоятельство требует наличия специального языка, который был бы понятен машине. Такой язык называется *машинным языком*, запись алгоритма на машинном языке называется *программой*, а самостоятельная фраза языка, являющихся некоторой операцией, называется *командой*. Таким образом, программа представляет собой некоторую последовательность команд, а выполнение компьютерной программы сводится к последовательному выполнению отдельных ее команд.

В общем случае команда должна содержать информацию о том, какую именно операцию нужно выполнить, информацию об ее операндах (аргументах и результате) и информацию о том, какую команду надо выполнить вслед за данной командой.

Информация об операндах и следующей команде, подлежащей выполнению, задается путем указания их адресов. На каждом типе компьютеров фиксируется определенный формат команд, позиции (разряды) машинного слова, представляющего собой команду, объединяются в определенные поля, и каждое поле служит для задания информации определенного назначения. Одно из полей, называемое полем кода операции, служит для указания вида операции.

Для этого все операции, входящие в набор машинных операций компьютера, нумеруются (кодируются) и в поле операции команды указывается номер (код) требуемой операции.

Например, команда может иметь следующий формат:

θ	A1	A2	A3	A4
Поле кода операции	Поле первого адреса	Поле 2-го адреса	Поле 3-го адреса	Поле адреса следующей команды

Например:

0010	0100	0110	1000	1100

Такая команда называется *четырехадресной*. Она обеспечивает принудительный порядок выполнения команд в компьютере, т.к. из памяти выбирается и выполняется та команда, адрес которой записан в четвертом поле адресной части команды. В настоящие время подавляющее распространение получили компьютеры с естественным порядком выполнения команд, при котором команды нумеруются при программировании и в этом порядке записываются в память, затем в этом же порядке выбираются из памяти и выполняются. При естественном порядке надобность в четвертом адресе отпадает и команда становится 3-х адресной.

θ	A1	A2	A3

Это структура наиболее точно отвечает формуле

Z=X θ Y,

что представляет известные удобства при программировании. Но эта структура с точки зрения технической реализации не является оптимальной. Структуру команды можно значительно упростить, если результат операции оставлять в сумматоре АУ или заносить в специальную ячейку, например в A1.

Тогда надобность в 3-м адресе отпадает.

θ	A1	A2

И наконец, структура команды становится наиболее простой, если одним из двух операндов считать результат предыдущей операций, зафиксированным в сумматоре.

θ	A

§2.4. Классификация компьютеров

Существуют различные системы классификации компьютеров. Рассмотрим некоторые из них.

Классификация по назначению

Классификация по назначению – одна из наиболее ранних систем классификации. Он связан с тем, как компьютер применяется. По этому принципу различают большие ЭВМ, мини – ЭВМ, микро – ЭВМ и персональные компьютеры, которые в свою очередь, подразделяются на массовые, деловые, портативные, развлекательные и рабочие станции.

Большие ЭВМ

Это самые мощные компьютеры. Их применяют для очень сложных расчетов, например в космических исследованиях, ядерной физике, в военных целях и др. За рубежом компьютеры этого класса называют *мэйнфреймами (mainframe)*. Для обслуживания таких компьютеров необходимо несколько десятков человек. На базе таких суперкомпьютеров создаются *вычислительные центры*, включающие в себя несколько отделов или групп.

Мини – ЭВМ

От больших ЭВМ компьютеры этой группы отличаются уменьшенными размерами и, соответственно, меньшей производительностью и стоимостью. Такие компьютеры используются крупными предприятиями, научными учреждениями и некоторыми высшими учебными заведениями, сочетающими учебную деятельность с научной.

Мини – ЭВМ часто применяются для управления производственными процессами. Для обслуживания мини – ЭВМ также требуется создание вычислительного центра, хотя и не такой многочисленный, как для больших ЭВМ.

Микро – ЭВМ

Компьютера данного класса доступны многим предприятиям и организациям. Для обслуживания этих компьютеров обычно не создаются вычислительные центры, т.к. для этого достаточно всего несколько человек. Микро – ЭВМ чаще всего используются как вспомогательные компьютеры для больших ЭВМ. Например, они используются для предварительной подготовки данных, перед тем как их будет обрабатывать большая ЭВМ.

Персональные компьютеры (ПК)

Эта категория компьютеров получила особо бурное развитие в течение последних двадцати лет. Из названия видно, что такой компьютер предназначен для обслуживания одного рабочего места. Как правило, с персональным компьютером работает один человек. Несмотря на свои небольшие размеры и относительно невысокую стоимость, современные персональные компьютеры обладают немалой производительностью.

Последние модели персональных компьютеров превосходят по производительности и особенно по своим возможностям большие ЭВМ 70-х годов, мини – ЭВМ 80-х годов и микро – ЭВМ первой половины 90-х годов 20 века.

Особенно широкую популярность персональные компьютеры приобрели после 1995 года в связи с бурным развитием Интернета. Возможностей персонального компьютера вполне достаточно для использования всемирной сети в качестве источника научной, справочной, учебной, культурной и развлекательной информации. Персональные компьютеры являются удобным средством автоматизации учебного процесса по любым дисциплинам, средством организации дистанционного (заочного) обучения и средством организации досуга.

До последнего времени персональные компьютеры условно рассматривали в двух категориях: бытовые ПК и профессиональные ПК. Бытовые модели имели, как правило, меньшую производительность, но в них были приняты особые меры для работы с цветной графикой и звуком, чего не требовалось для профессиональных моделей. В связи с достигнутым в последние годы резким удешевлением средств вычислительной техники, границы между профессиональными и бытовыми моделями практически стерлись.

Начиная с 1999 года в области персональных компьютеров действует международный классификационный стандарт – *спецификация PC99*. Он регламентирует принципы классификации персональных компьютеров и оговаривает минимальные и рекомендуемые требования к каждой категории персональных компьютеров. Новый стандарт устанавливает следующие категории персональных компьютеров:

- Consumer Personal Computer (PC) (массовый ПК);
- Office PC (деловой ПК);
- Mobile PC (портативный ПК);
- Workstation PC (рабочая станция);
- Entertainment PC (развлекательный ПК).

Согласно спецификации PC99 большинство персональных компьютеров попадают в категорию массовых ПК. Для деловых ПК минимизированы требования к средствам воспроизведения графики, а к средствам работы со звуковыми данными требования вообще не предъявляются. Для портативных ПК обязательным является наличие средств для создания соединений удаленного доступа, то есть средств компьютерной связи. В категории рабочих станций повышены требования к устройствам хранения данных, а категории развлекательных ПК – требования к средствам воспроизведения графики и звука.

Однако, жизнь не стоит на месте. Классификация PC99 уже устарела и мы стоим на пороге новой классификации. Грани между указанными категориями персональных компьютеров также практически стерлись.

Сейчас уже невозможно представить себе офисный компьютер без соответствующих мультимедийных средств, а на развлекательном ПК можно производить достаточно сложные вычисления и расчеты.

Копилка Под термином *мультимедиа* подразумевается сочетание нескольких видов данных в одном документе (текстовые, графические, музыкальные и видеоданные) или совокупность устройств для воспроизведения этих данных.

Персональные компьютеры получили столь широкое и повсеместное распространение, их применение настолько разнообразное, что сейчас уже не говорят персональные компьютеры, а говорят просто компьютеры. В дальнейшем мы тоже не будем подчеркивать "персональность" компьютеров и будем говорить просто о компьютерах.

Другие системы классификации компьютеров

Классификация по уровню специализации. По уровню специализации компьютеры делят на *универсальные и специализированные*. На базе универсальных компьютеров можно собирать вычислительные системы произвольной конфигурации. Так, например, один и тот же компьютер можно использовать для работы с текстами, музыкой, графикой, фото и видеоматериалами.

Специализированные компьютеры предназначены для решения конкретного круга задач. К таким компьютерам относятся, например, бортовые компьютеры автомобилей, судов, самолетов, космических аппаратов. Бортовые компьютеры решают только специальные задачи, на которые они запрограммированы на заводе изготовителе (например, оптимизацию расхода топлива двигателя в зависимости от конкретных условий движения или управление средствами ориентации и навигации и т.д.).

Специализированные компьютеры, ориентированные на работу с графикой, называют *графическими станциями*. Специализированные компьютеры, объединяющие компьютеры какого – нибудь предприятия в одну сеть, называют *файловыми серверами*. Компьютеры, обеспечивающие передачу информации между различными участками Интернета, называют *сетевыми серверами*.

Во многих случаях с задачами специализированных компьютеров могут успешно справляться и обычные универсальные компьютеры и так часто на практике и происходит, хотя считается, что использование специализированных компьютеров все – таки эффективнее. Здесь опять таки идет тенденция к сближению обоих категорий компьютеров.

Классификация по типоразмерам. Различают *настольные (desktop), портативные (notebook), карманные (palmtop)*.

Настольные модели распространены наиболее широко. Они являются принадлежностью рабочего места. Эти модели отличаются простотой изменения конфигурации за счет несложного подключения дополнительных внешних устройств или установки дополнительных внутренних компонентов. Достаточные размеры корпуса в настольном исполнении позволяют выполнять большинство подобных работ без привлечения специалистов, а это позволяет настраивать компьютерную систему оптимально для решения именно тех задач, для которых она была приобретена.

Портативные модели удобны для транспортировки. Их используют бизнесмены, коммерсанты, руководители предприятий и организаций, проводящие много времени в командировках и переездах. Особая привлекательность портативных компьютеров связана с тем, что их можно использовать в качестве средства связи.

Карманные модели выполняют функции "интеллектуальных записных книжек". Они позволяют хранить оперативные данные и получать к ним быстрый доступ. Некоторые карманные модели имеют жестко встроенное программное обеспечение, что облегчает непосредственную работу, но снижает гибкость в выборе прикладных программ.

Классификация по совместимости. В мире существуют множество различных видов и типов компьютеров. Они выпускаются разными производителями, работают с разными программами. При этом очень важным вопросом становится *совместимость* различных компьютеров между собой. От совместимости зависит возможность переноса программ с одного компьютера на другой и возможность совместной работы разных типов компьютеров с одними и теми же данными.

Аппаратная совместимость. По аппаратной совместимости различают так называемые *аппаратные платформы.* В области персональных компьютеров на сегодняшний день наиболее широко распространены две аппаратные платформы – *IBM PC* и *Apple Macintosh.* Принадлежность компьютеров к одной платформе повышает совместимость между собой, принадлежность к разным платформам соответственно понижает совместимость, но тем не менее, говорить о полной несовместимости этих двух платформ нельзя.

Кроме аппаратной совместимости существуют и другие виды совместимости: *совместимость на уровне операционной системы, программная совместимость, совместимость на уровне данных.*

§2.5. Аппаратное обеспечение

В §2.3. мы на элементарном уровне рассмотрели принципы работы компьютера и основные его устройства. Рассмотрим теперь более подробно и на современном уровне устройства из которых состоит компьютер.

Состав вычислительной системы называется *конфигурацией*. Аппаратные и программные средства вычислительной техники принято рассматривать отдельно. Соответственно, отдельно рассматривают *аппаратную конфигурацию* вычислительных систем и их *программную конфигурацию*.

К *аппаратному обеспечению* вычислительных систем относятся устройства и приборы, образующие аппаратную конфигурацию. Современные компьютеры и вычислительные комплексы имеют блочно – модульную конструкцию. Аппаратную конфигурацию, необходимую для выполнения конкретных видов работ, можно собирать из готовых узлов и блоков.

По способу расположения устройств относительно *центрального процессорного устройства* (ЦПУ – *Central Processing Unit, CPU*) различают внутренние и внешние устройства. Внешними, как правило, являются большинство устройств ввода/вывода данных (их также часто называют *периферийными устройствами*).

Согласование между отдельными узлами и блоками выполняют с помощью переходных аппаратно – логических устройств, называемых *аппаратными интерфейсами*. Стандарты на аппаратные интерфейсы в вычислительной технике называют *протоколами*.

КОНЕЦ ЦИТАТЫ Таким образом, *протокол – это совокупность технических условий, которые должны быть обеспечены разработчиками устройств для успешного согласования их работы с другими устройствами.*

Многочисленные интерфейсы, присутствующие в архитектуре любой вычислительной системы, можно условно разбить на две большие группы: *последовательные* и *параллельные*. Через последовательный интерфейс данные передаются последовательно, бит за битом, а через параллельный – одновременно группами битов. Количество битов, участвующих в одной посылке, определяется разрядностью интерфейса. Например, восьмиразрядные параллельные интерфейсы передают 8 бит (один байт) за один цикл.

Параллельные интерфейсы обычно имеют более сложное устройство, чем последовательные, но обеспечивают более высокую производительность. Их применяют там, где важна скорость передачи данных. Производительность параллельных интерфейсов измеряют *байтами в секунду*. (байт/с; Кбайт/с; Мбайт/с).

В последовательных интерфейсах, как правило, не надо синхронизировать работу передающего и принимающего устройства

(поэтому их часто называют *асинхронными интерфейсами*), но пропускная способность их меньше и коэффициент полезного действия ниже, так как из-за отсутствия синхронизации полезные данные предваряются и завершаются посылками служебных данных. Т.е. на один байт полезных данных могут приходиться 1 – 3 служебных бита (состав и структуру посылки определяет конкретный протокол).

Поскольку обмен данными через последовательные устройства производится не байтами, а битами, их производительность измеряют битами в секунду (бит/с; Кбит/с; Мбит/с). Несмотря на кажущую простоту перевода единиц измерения скорости последовательной передачи в единицы измерения скорости параллельной передачи путем механического деления на 8, такой пересчет не выполняют, поскольку он некорректен из-за наличия служебных данных.

Последовательные интерфейсы применяют для подключения "медленных" устройств (устройств печати, устройств ввода/вывода знаковой и сигнальной информации, например клавиатуры, контрольных датчиков, малопроизводительных устройств связи и т.п.).

Базовая аппаратная конфигурация

Компьютер – универсальная техническая система. Его *конфигурацию* (состав оборудования) можно гибко изменять по мере необходимости. Тем не менее, существует понятие *базовой конфигурации*, которую считают типовой. В таком комплекте компьютер обычно поставляется. В настоящее время в базовой конфигурации рассматривают четыре устройства:

- системный блок;
- монитор;
- клавиатура;
- мышь.

Системный блок

Системный блок представляет собой основной узел, внутри которого установлены наиболее важные компоненты. Устройства, находящиеся в системном блоке, называются *внутренними*, а устройства, подключаемые к нему снаружи, называются *внешними*.

По внешнему виду системные блоки отличаются формой корпуса. Корпуса выпускаются в горизонтальном (*desktop*) и вертикальном (*tower*) исполнении. Корпуса, имеющие вертикальное исполнение, различают по габаритам: *полноразмерный (big tower), среднеразмерный (midi tower) и малоразмерный (mini tower)*. Среди корпусов, имеющих горизонтальное исполнение, выделяют *плоские* и *особо плоские (slim)*.

Кроме формы, для корпуса важен параметр, называемый *форм – фактором*. В настоящее время в основном используются корпуса двух форм

– факторов: AT и ATX. Форм – фактор корпуса должен быть обязательно согласован с форм – фактором главной (системной) платы компьютера, так называемой *материнской платы*.

Корпуса компьютеров поставляются вместе с блоком питания и, таким образом, мощность блока питания также является одним из параметров корпуса. Для массовых моделей достаточной является мощность блока питания 200 – 250 Вт.

Одним из основных устройств, находящихся в системном блоке является материнская плата. На ней размещаются:

- *процессор* – основная микросхема, выполняющая математические и логические операции;
- *микропроцессорный комплект (чипсет)* – набор микросхем, управляющих работой внутренних устройств компьютера и определяющая основные функциональные возможности материнской платы;
- *шины* – наборы проводников, по которым происходит обмен сигналами между устройствами компьютера;
- *оперативная память* – набор микросхем, предназначенных для временного хранения данных, когда компьютер включен;
- *ПЗУ (постоянное запоминающее устройство)* – микросхема, предназначенная для длительного хранения данных, в том числе и когда компьютер выключен;
- разъемы для подключения дополнительных устройств (*слоты*).

Рассмотрим эти устройства более подробно.

Оперативная память

Оперативная память (RAM – Random Access Memory) – это массив кристаллических ячеек, способных хранить данные. Существуют много различных типов оперативной памяти, но с точки зрения физического принципа действия различают *динамическую память (DRAM)* и *статическую память (SRAM)*.

Ячейки динамической памяти (DRAM) можно представить в виде микроконденсаторов, способных накапливать заряд на своих обкладках. Это наиболее распространенный и экономически доступный тип памяти. Недостатки этого типа связаны, во-первых, с тем, что как при заряде, так и при разряде конденсаторов неизбежны переходные процессы, то есть запись данных происходит сравнительно медленно. Второй важный недостаток связан с тем, что заряды ячеек имеют свойство рассеиваться в пространстве, причем весьма быстро. Если оперативную память постоянно не "подзаряжать", утрата данных происходит через несколько сотых долей секунды. Для борьбы с этим явлением в компьютере происходит постоянная *регенерация (освежение, подзарядка)* ячеек оперативной памяти. Регенерация происходит несколько десятков раз в секунду и вызывает непроизводительный расход ресурсов компьютера.

Ячейки статической памяти (SRAM) можно представить как электронные микроэлементы – *триггеры*, состоящие из нескольких транзисторов. В триггере хранится не заряд, а состояние (включен/выключен), поэтому этот тип памяти обеспечивает более высокое быстродействие, хотя технологически он сложнее и, соответственно, дороже.

Микросхемы динамической памяти используют в качестве основной оперативной памяти компьютера. Микросхемы статической памяти используют в качестве вспомогательной памяти (так называемой *кэш-памяти*), предназначенной для оптимизации работы процессора.

Каждая ячейка памяти имеет свой адрес, который выражается числом. В настоящее время в процессорах Intel Pentium и некоторых других принята 32-разрядная адресация, а это означает, что всего независимых адресов может быть 2^{32}. Таким образом, в современных компьютерах возможна непосредственная адресация к полю памяти размером 2^{32}=4 294 967 296 байт (4,3 Гбайт). Однако это отнюдь не означает, что именно столько оперативной памяти непременно должно быть в компьютере. Предельный размер поля оперативной памяти, установленной в компьютере, определяется микропроцессорным комплектом (чипсетом) материнской платы и обычно составляет несколько сот Мбайт.

Одна адресуемая ячейка содержит восемь двоичных ячеек, в которых можно сохранить 8 бит, то есть один байт данных. Таким образом, адрес любой ячейки памяти можно выразить четырьмя байтами.

Представление о том, сколько оперативной памяти должно быть в типовом компьютере, непрерывно меняется. В середине 80-х годов поле памяти размером в 1 Мбайт казалось огромным, в начале 90-х достаточным считался объем 4 Мбайт, к середине 90-х годов он увеличился до 8 Мбайт, а затем и до 16 Мбайт. Сегодня типичным считается размер "оперативки" 64 Мбайт, но очень скоро эта величина будет превышена в несколько раз даже для моделей массового потребления.

Оперативная память в компьютере размещается на стандартных панельках, называемых *модулями*. Модули оперативной памяти вставляют в соответствующие разъемы на материнской плате. Конструктивно модули памяти имеют два исполнения – однорядные (*SIMM-модули)* и двухрядные (*DIMM-модули)*. На компьютерах с процессором Pentium однорядные модули можно применять только парами (количество разъемов для их установки на материнской плате всегда четное), а DIMM-модули можно устанавливать по одному. Многие модели материнских плат имеют разъемы как того, так и другого типа, но комбинировать на одной плате модули разных типов нельзя.

Основными характеристиками модулей оперативной памяти являются объем памяти и время доступа. SIMM-модули поставляются объемами 4, 8, 16, 32 Мбайт, а DIMM-модули 16, 32, 64, 128 Мбайт и более. Время доступа показывает, сколько времени необходимо для обращения к ячейкам памяти – чем оно меньше, тем лучше. Время доступа измеряется в миллиардных долях секунды (*наносекундах, нс*). Типичное время доступа к оперативной памяти

для SIMM-модулей – 50-70 нс. Для современных DIMM-модулей оно составляет 7 – 10 нс.

Процессор

Процессор – основная микросхема компьютера, в которой и производятся все вычисления. Конструктивно процессор состоит из ячеек, похожих на ячейки оперативной памяти, но в этих ячейках данные могут не только храниться, но и изменяться. Внутренние ячейки процессора называются *регистрами*. Важно также отметить, что данные, попавшие в некоторые регистры, рассматриваются не как данные, а как команды, управляющие обработкой данных в других регистрах. Среди регистров процессора есть и такие, которые в зависимости от своего содержания способны модифицировать исполнение команд. Таким образом, управляя засылкой данных в разные регистры процессора, можно управлять обработкой данных. На этом и основано исполнение программ.

С остальными устройствами компьютера, и в первую очередь с оперативной памятью процессор связан несколькими группами проводников, называемых *шинами*. Основных шин три: *шина данных, адресная шина* и *командная шина*.

Адресная шина. У процессоров Intel Pentium адресная шина 32-разрядная, то есть состоит из 32 параллельных линий. В зависимости от того, есть напряжение на какой-то линии или нет, говорят, что на этой линии выставлена единица или ноль. Комбинация из 32 нулей и единиц образует 32-разрядный адрес, указывающих на одну из ячеек оперативной памяти. К ней и обращается процессор для копирования содержимого этой ячейки в один из своих регистров либо для записи содержимого регистра в эту ячейку.

Шина данных. По этой шине непосредственно и происходит процесс копирования данных из оперативной памяти в регистры процессора и обратно. В компьютерах с процессором Intel Pentium шина данных 64-разрядная, т.е. состоит из 64 линий, по которым за один раз (такт или цикл) на обработку поступает сразу 8 байтов.

Шина команд. Для того, чтобы процессор мог обрабатывать данные, ему нужны команды. Он должен знать, что следует сделать с теми байтами, которые хранятся в его регистрах. Эти команды в соответствии со вторым принципом фон Неймана поступают в процессор тоже из оперативной памяти, но не из тех областей, где хранятся данные, а оттуда, где хранятся программы. Команды тоже представлены в виде байтов. Самые простые команды укладываются в один байт, однако есть и такие, для которых нужно два, три и более байтов. В большинстве современных процессоров шина команд 32-разрядная (например, в процессоре Intel Pentium), хотя существуют 64-разрядные и даже 128-разрядные.

Система команд процессора. Совокупность всех возможных команд, которые может выполнить процессор над данными, образует так называемую

систему команд процессора. Процессоры, относящиеся к одному семейству, имеют одинаковые или близкие системы команд. Процессоры, относящиеся к разным семействам, различаются по системе команд и невзаимозаменяемы.

Процессоры с расширенной и сокращенной системой команд. Чем шире набор системных команд процессора, тем сложнее его архитектура, тем длиннее формальная запись команды (в байтах), тем выше средняя продолжительность исполнения одной команды, измеренная в тактах работы процессора. Так, например, система команд процессоров Intel Pentium в настоящее время насчитывает более тысячи команд. Такие процессоры называют *процессорами с расширенной системой команд – CISC-процессорами (CISC – Complex Instruction Set Computing).*

В противоположность CISC-процессорам в середине 80-х годов появились процессоры архитектуры RISC с сокращенной системой команд (*RISC – Reduced Instruction Set Computing).* При такой архитектуре количество команд в системе намного меньше, и каждая из них выполняется намного быстрее. Таким образом, программы, состоящие из простейших команд, выполняются этими процессорами много быстрее. Оборотная сторона сокращенного набора команд состоит в том, что сложные операции приходится эмулировать далеко не эффективной последовательностью простейших команд сокращенного набора.

В результате конкуренции между двумя подходами к архитектуре процессоров сложилось следующее распределение их сфер применения:

- CISC-процессоры используют в универсальных вычислительных системах;
- RISC-процессоры используют в специализированных вычислительных системах или устройствах, ориентированных на выполнение единообразных операций.

Для компьютеров платформы IBM PC долгое время выпускались только CISC-процессоры, к которым относятся и все процессоры семейства Intel Pentium. Однако в последнее время компания AMD приступила к выпуску процессоров семейства AMD-K6, в основе которых лежит внутреннее ядро, выполненное по RISC-архитектуре, и внешняя структура, выполненная по архитектуре CISC. Таким образом, сегодня появились процессоры, имеющие гибридную структуру.

Совместимость процессоров. Если два процессора имеют одинаковую систему команд, то они полностью совместимы на программном уровне. Это означает, что программа, написанная для одного процессора, может исполняться и другим процессором. Процессоры, имеющие разные системы команд, как правило, несовместимы или ограниченно совместимы на программном уровне.

Группы процессоров, имеющих ограниченную совместимость, рассматривают как *семейство процессоров.* Так, например, все процессоры Intel Pentium относятся к так называемому семейству x86. Родоначальником этого семейства был 16-разрядный процессор Intel 8086, на базе которого

собиралась первая модель компьютера IBM PC. Впоследствии выпускались процессоры Intel 80286, Intel 80386, Intel 80486, Intel Pentium 60, 66, 75, 90, 100, 133; несколько моделей процессоров Intel Pentium MMX, модели Intel Pentium Pro, Intel Pentium II, Intel Celeron, Intel Xeon, Intel Pentium III и другие. Все эти модели, и не только они, а также многие модели процессоров компаний AMD и Cyrix относятся к семейству x86 и обладают совместимостью по принципу "сверху вниз".

Принцип совместимости "сверху вниз" – это пример неполной совместимости, когда каждый новый процессор "понимает" все команды своих предшественников, но не наоборот. Это естественно, поскольку двадцать лет назад разработчики процессоров не могли предусмотреть систему команд, нужную для современных программ.

Благодаря такой совместимости на современном компьютере можно выполнять любые программы, созданные в последние десятилетия для любого из предшествующих компьютеров, принадлежащего той же аппаратной платформе.

Монитор

Монитор – устройство визуального представления данных. Это не единственно возможное, но главное устройство вывода. Его основными потребительскими параметрами являются: размер и шаг маски экрана, максимальная частота регенерации изображения, класс защиты.

Размер экрана монитора измеряется между противоположными углами трубки кинескопа по диагонали. Единица измерения – дюймы. Стандартные размеры: 14"; 15"; 17"; 19"; 20"; 21". В настоящее время наиболее универсальными являются мониторы размером 15 и 17 дюймов, а для операций с графикой желательны мониторы размером 19 – 21 дюйм.

Изображение на экране монитора получается в результате облучения люминофорного покрытия остронаправленным пучком электронов, разогнанных в вакуумной колбе. Для получения цветного изображения люминофорное покрытие имеет точки или полоски трех типов, светящиеся красным, зеленым и синим цветом. Чтобы на экране все три луча сходились строго в одну точку и изображение было четким, перед люминофором ставят маску – панель с регулярно расположенными отверстиями или щелями. Часть мониторов оснащена маской из вертикальных проволочек, что усиливает яркость и насыщенность изображения. Чем меньше шаг между отверстиями или щелями (*шаг маски*), тем четче и точнее полученное изображение. Шаг маски измеряют в долях миллиметра. В настоящее время наиболее распространены мониторы с шагом маски 0,25 – 0,27 мм. Устаревшие мониторы могут иметь шаг до 0,43 мм, что негативно сказывается на органах зрения при работе с компьютером. Модели повышенной стоимости могут иметь шаг маски менее 0,25 мм.

Частота регенерации (обновления) изображения показывает, сколько раз в течении секунды монитор может полностью сменить изображение (поэтому её также называют *частотой кадров*). Частоту регенерации измеряют в герцах (Гц). Чем она выше, тем четче и устойчивее изображение, тем меньше утомление глаз, тем больше времени можно работать на компьютере без перерыва. При частоте регенерации порядка 60 Гц мелкое мерцание изображения заметно невооруженным глазом. Сегодня такое значение считается недопустимым. Минимальным считают значение 75 Гц, нормативным – 85 Гц, а комфортным – 100 Гц и более.

Класс защиты монитора определяется стандартом, которому соответствует монитор с точки зрения техники безопасности. В настоящее время общепризнанными считаются следующие международные стандарты: *MPR-II, TCO-92, TCO-95, TCO-99* (приведены в хронологическом порядке). Стандарт MPR-II ограничил уровни электромагнитного излучения пределами, безопасными для человека. В стандарте TCO-92 эти нормы были сохранены, а в стандартах TCO-95 и TCO-99 ужесточены. Эргономические и экологические нормы впервые появились в стандарте TCO-95, а стандарт TCO-99 установил самые жесткие нормы по параметрам, определяющим качество изображения (яркость, контрастность, мерцание, антибликовые свойства покрытия).

Клавиатура

Клавиатура – это клавишное устройство управления компьютером. Она служит для ввода *алфавитно – цифровых (знаковых)* данных, а также команд управления. Клавиатура и монитор обеспечивают простейший *интерфейс пользователя*. С помощью клавиатуры управляют компьютером, а с помощью монитора получают от него отклик.

Клавиатура относится к стандартным средствам персонального компьютера. Ее основные функции не нуждаются в поддержке специальными системными программами (драйверами). Необходимое программное обеспечение для начала работы с компьютером уже имеется в микросхеме ПЗУ в составе базовой системы ввода/вывода (*BIOS – Basic Input Output System*), и потому компьютер реагирует на нажатие клавиш сразу после включения компьютера. Принцип действия клавиатуры заключается в следующем:

1. При нажатии на клавишу (или комбинацию клавиш) специальная микросхема, встроенная в клавиатуру, выдает так называемый *скан-код*.
2. Скан-код поступает в микросхему, выполняющая функции *порта* клавиатуры.

ЗАПОМНИМ Порты – специальные аппаратно – логические устройства, отвечающие за связь процессора с другими устройствами.

3. Порт клавиатуры выдает процессору прерывание с фиксированным номером. Для клавиатуры номер прерывания – 9 (Interrupt 9, Int 9).
4. Получив прерывание, процессор откладывает текущую работу и по номеру прерывания обращается в специальную область оперативной памяти, в которой находится так называемый *вектор прерываний*. Вектор прерываний – это список адресных данных с фиксированной длиной записи. Каждая запись содержит адрес программы, которая должна обслужить прерывание с номером, совпадающим с номером записи.
5. Определив адрес начала программы – обработчика возникшего прерывания, процессор переходит к ее исполнению. Простейшая программа обработки клавиатурного прерывания "зашита" в микросхему ПЗУ, но программисты могут "подставить" вместо нее свою программу обработки данного прерывания, если изменят данные в векторе прерываний.
6. Программа-обработчик прерывания направляет процессор к порту клавиатуры, где он находит скан-код, загружает его в свои регистры, потом под управлением обработчика определяет, какой код символа соответствует данному скан-коду.
7. Далее обработчик прерываний направляет полученный код символа в небольшую область памяти, известную как *буфер клавиатуры* и прекращает свою работу, известив об этом процессор.
8. Процессор прекращает обработку данного прерывания и возвращается к отложенной задаче.
9. Введенный символ хранится в буфере клавиатуры до тех пор, пока его не заберет оттуда та программа, для которой он и предназначался, например, текстовый редактор.

Мышь

Мышь – устройство управления манипуляторного типа. Перемещение мыши по плоской поверхности синхронизировано с перемещением графического объекта (*указателя мыши*) на экране монитора.

В отличие от клавиатуры, мышь не является стандартным органом управления и компьютер не имеет для нее выделенного порта. Для мыши нет и постоянного выделенного прерывания, а базовые средства ввода и вывода (BIOS) компьютера, размещенные в постоянном запоминающем устройстве (ПЗУ), не содержат программных средств для обработки прерываний мыши.

В связи с этим в первый момент после включения компьютера мышь не работает. Она нуждается в поддержке специальной системной программы – *драйвера мыши*.

Драйвер устанавливается либо при первом подключении мыши, либо при установке операционной системы компьютера. Хотя мышь и не имеет

выделенного порта на материнской плате, для работы с ней используют один из стандартных портов, средства для работы с которыми имеются в составе BIOS. Драйвер мыши предназначен для интерпретации сигналов, поступающих через порт. Кроме того, он обеспечивает механизм передачи информации о состоянии и положении мыши операционной системе и работающим программам.

Компьютером управляют перемещением мыши по плоскости и кратковременными нажатиями правой или левой кнопок (клавиш). Эти нажатия называются *щелчками (click)*. В отличие от клавиатуры мышь не может напрямую использоваться для ввода знаковой информации – ее принцип управления является событийным. Перемещения мыши и щелчки ее кнопок являются *событиями* с точки зрения ее программы – драйвера. Анализируя эти события, драйвер устанавливает, когда произошло событие и в каком месте экрана в этот момент находился указатель. Эти данные передаются в прикладную программу, с которой работает пользователь в данный момент. По ним программа может определить команду, которую имел в виду пользователь и приступить к ее исполнению.

Комбинация монитора и мыши обеспечивает наиболее современный тип интерфейса пользователя, который называется *графическим*. Пользователь наблюдает на экране графические *объекты* и *элементы управления*. С помощью мыши пользователь изменяет свойства объектов и приводит в действие элементы управления компьютером, а с помощью монитора получает отклик в графическом виде.

Контрольные вопросы:

1. Краткая история развития средств вычислительной техники.
2. Логические элементы вычислительной техники.
3. Функциональные элементы и типовые функциональные схемы.
4. Принципы фон – Неймана.
5. Классификация компьютеров.
6. Базовая аппаратная конфигурация компьютеров.

ГЛАВА III. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРОВ

§3.1. Структура программного обеспечения

Программное обеспечение составляет вторую неотъемлемую компоненту современных компьютеров, без которого немыслимо их функционирование.

Программное обеспечение – это совокупность программ, позволяющих организовать решение задач пользователя. По назначению программное обеспечение можно условно разбить на три класса:

Системное программное обеспечение организует процесс обработки информации в компьютере. По сути – это программное устройство управления компьютером. Основу системного программного обеспечения составляют операционные системы (ОС).

Основное назначение ОС – управление процессом обработки информации и организация связи пользователя с компьютером. ОС – является неотъемлемой частью компьютера, обеспечивая управление всеми аппаратными средствами, а человеку (пользователю) позволяет управлять работой компьютера с помощью соответствующего графического интерфейса и (или) команд.

Ядро ОС дополняется набором сервисных программ, которые служат разным целям. С их помощью производится например, начальная разметка дисков, установка параметров внешних устройств, выдача информации на печать, тестирование ОП и других устройств, стыковка с другими компьютерами (локальная сеть) и многое другое.

В состав системного программного обеспечения входят также средства диагностики и контроля. Эти программы обеспечивают автоматический поиск неисправностей в компьютере. Средства диагностики и контроля могут входить в состав ОС, но могут существовать и автономно.

Система программирования (СП) позволяет разрабатывать программы на удобном для восприятия человеком языке, а не в машинных кодах. В СП входят также компиляторы – комплекс программ, обеспечивающих автоматический перевод программ написанных на символическом языке на машинный язык.

Системы программирования – это особая категория программных средств. С их помощью создаются другие программы, в том числе и прикладные программы. Следовательно, эта категория программных средств совершенно аналогична средствам производства в промышленности – таким как станки, инструменты, сырье и т.д. При этом роль сырья играет информация – текстовые и числовые данные, закодированные сообщения, графическая информация и др.

Прикладное программное обеспечение составляют те программы ради которых и существует компьютер. Это различные пакеты программ, предназначенные для пользователей различных категорий. Например, это могут быть различные текстовые редакторы, СУБД, табличные процессоры и интегрированные программные средства.

Прикладные программы составляют категорию программных средств, обращенных к пользователям компьютера – людям, которые не обязаны уметь программировать или знать устройство компьютера. Их цель заключается либо в том, чтобы с помощью компьютера решать свои повседневные задачи, включая профессиональные, либо учиться определенным навыкам (не обязательно относящихся к компьютеру), либо проводить свой досуг, играя в компьютерные игры.

Прикладные системы могут иметь общий характер, например, обеспечивать составление документов и их печать или хранение и выдачу различных справок.

Другие классы прикладных программ ориентированы на автоматизацию конкретных видов деятельности, например, обучение определенным предметам в школе, проектирование зданий и сооружений, анализ электрокардиограмм, проведение бухгалтерских расчетов и т.д. и т.п.

§3.2. Основы Windows-95

Графический интерфейс Windows – 95

Операционная система Windows – 95, разработанная фирмой Microsoft, обеспечивает большое количество возможностей и удобств для пользователей и программистов. Прежде всего, Windows – 95 обеспечивает удобный и наглядный графический интерфейс пользователя (так называемый стандарт GUI – Graphical User Interface), а также обеспечивает многозадачный режим работы, что означает возможность одновременно

запускать и работать с несколькими программами. Каждая программа работает в своем собственном окне. Управление окнами в Windows – 95 стандартизовано. Это значительно облегчает изучение самых разных программных средств. Рассмотрим некоторые элементы графического интерфейса:

Рабочий стол Windows – 95

На рисунке 1 изображен вид рабочего стола сразу после включения компьютера.

Рабочий стол можно ассоциировать с настоящим рабочим столом в Вашем офисе или дома. На нем можно расположить всё, что нужно Вам для

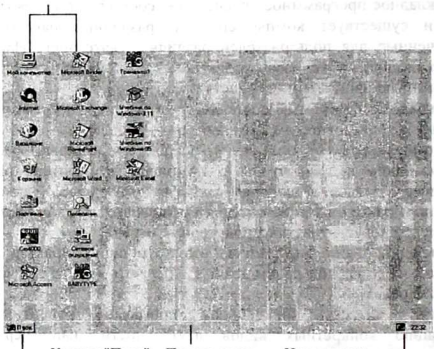


Рис. 1

работы. На рабочем столе расположены значки папок и значки ярлыков наиболее часто используемых программ. Количество значков на рабочем столе зависит от способа установки Windows и от Вашей настройки рабочего стола. Значки ярлыков отличаются от значков папок наличием в левом нижнем углу небольшой стрелочки. (См. рис. 2). Ярлык относится к какому – либо приложению или документу. (Обратите внимание, что понятия программа и приложение означают одно и тоже). В дальнейшем мы будем использовать оба этих термина.

Чтобы запустить приложение или документ достаточно дважды щелкнуть по его ярлыку на рабочем столе, при этом, если вы открываете документ, то автоматически запускается приложение, которое обрабатывает этот документ.

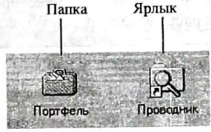


Рис. 2

Существует и другой способ запуска программ. Для этого надо воспользоваться кнопкой "Пуск". Если нажать мышью на эту кнопку, то появится ***меню***.

Меню это такой механизм, который предоставляет пользователю возможность выбора дальнейших действий. Каждое меню состоит из отдельных пунктов (которые в свою очередь могут также быть меню – так называемые вложенные меню, говорят ещё подменю) или команд.

Итак, чтобы запустить приложение необходимо нажать кнопку "Пуск" и пропутешествовав по нужным пунктам меню и подменю добраться до нужной программы и нажать левую кнопку мыши или клавишу Enter.

Рис. 3

На рисунке 3 показан пример запуска программы Калькулятор.

Внимание! При выключении компьютера всегда следует использовать кнопку "Пуск". Если выключить компьютер "просто так", то работа Windows завершится некорректно, могут быть потеряны некоторые данные и может нарушиться работоспособность Windows!

Панель задач

Панель задач отображает значки работающих в данный момент приложений и позволяет быстро переключаться с одной задачи на другую. Для этого надо просто щелкнуть по соответствующему значку программы на панели задач. После закрытия приложения его значок также исчезает с панели задач.

Индикаторы

Индикаторы дают некоторую дополнительную информацию для пользователя. Например, в правом нижнем углу отображаются часы. Рядом слева находится индикатор используемого языка.

En (English) – означает, что в данный момент используется английский язык.

Ru (Russian) – означает, что в данный момент используется русский язык.

Для переключения с одного языка на другой используются клавиши Shift + Alt (знак + означает, что эти клавиши нужно нажимать одновременно). Если индикатор языка виден на экране, то можно просто щелкнуть по индикатору.

Элементы окон Windows – 95

Как уже отмечалось, управление окнами в Windows – 95 стандартизовано. Практически все окна имеют одни и те же элементы управления. Рассмотрим их на рисунке 4.

Любое окно имеет ***строку заголовка*** в котором указывается имя программы и имя документа.

При нажатии кнопки "Свернуть" активное окно уменьшается до размеров значка и помещается на панель задач. Следует иметь в виду, что программа в этом случае продолжает работать, но в свернутом виде. При нажатии кнопки "Развернуть" окно разворачивается до максимально возможных размеров. При этом кнопка "Развернуть" заменяется на кнопку "Восстановить"

Если щелкнуть по этой кнопке, то окно восстановит свой первоначальный размер. Чтобы закрыть окно, щелкните по кнопке "Закрыть".

Рис. 4

Вторая строка окна называется ***строкой меню***. Если щелкнуть по какому – либо пункту, то появится ***подменю*** – вертикальное меню в котором можно выбрать соответствующие команды.

Панель инструментов позволяет выполнять некоторые действия быстрее, используя кнопки на панели и не обращаясь к пунктам меню.

Кнопки на панели инструментов дублируют некоторые, наиболее часто используемые команды меню.

Линейка прокрутки используется в том случае, когда вся информация не помещается в окне. Если нажать на кнопки , то информация в окне будет сдвигаться вверх или вниз.

Бегунок показывает относительное положение в тексте. Например, если бегунок находится вверху линейки прокрутки, то вы находится ближе к началу документа, если внизу, то ближе к концу документа. Можно прокручивать текст и с помощью бегунка. Для этого надо нажать на бегунок мышью и, не отпуская двигать бегунок в нужном направлении. Линейки прокрутки бывают вертикальные и горизонтальные. Методы работы с горизонтальными линейками прокрутки абсолютно идентичны. В тех

случаях, когда информация полностью помещается в окне, линейки прокрутки не выводятся.

Строка состояния отображает состояние различных элементов в данном окне. В дальнейшем мы познакомимся с соответствующими примерами, работая в Word – 97.

Меню

В окне каждой программы находится строка меню. Это так называемое горизонтальное меню. Каждый пункт этого меню состоит в свою очередь из соответствующих подменю – вертикальных меню. Для того чтобы открыть подменю надо щелкнуть по соответствующему пункту горизонтального меню. Каждый пункт меню содержит либо команды с помощью которых можно выполнить какие – то действия, либо режимы (опции), которые можно активизировать или дезактивизировать, либо содержат ещё дополнительные подменю (вложенные меню).

Соглашения по обозначениям в меню

Таблица 1

Соглашения	Значение соглашения
Треугольник ( ) после команды	Означает, что этот пункт содержит вложенное меню (подменю).
Многоточие (...) после команды	После выбора этого пункта появится диалоговое окно. Оно содержит необходимые для выполнения команды параметры, которые нужно выбрать или ввести.
Комбинация клавиш после команды	Эта комбинация клавиш является "быстрыми" клавишами для команды, т.е. для выполнения этой команды можно нажать на указанную комбинацию клавиш, не открывая меню.
Блеклый (или невидимый) пункт	В данный момент этот пункт (команду) нельзя использовать
Галочка (:) перед командой	Означает, что данная команда или режим активизирована. Повторное нажатие отключает команду или режим. Значок : при этом исчезает.

Диалоговые окна

Для выполнения некоторых команд необходимо задать один или несколько параметров. Для этого используются ***диалоговые окна***. Если после команды в меню стоит многоточие, то после её выбора открывается диалоговое окно. (См. рис. 5,6).

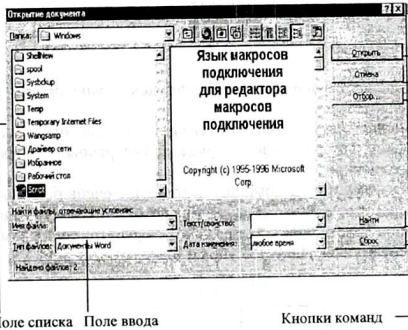


Рис. 5

Диалоговые окна содержат специальные области (называемые элементами управления или параметрами).

Значения параметров необходимо выбрать из некоторого списка или ввести с клавиатуры.

Типы параметров диалоговых окон.

Кнопки команд – используются для выполнения действий. Все диалоговые окна обязательно имеют кнопки **ОК** и **Отмена**. При нажатии кнопки ОК диалог завершается и команда с выбранными вами параметрами выполняется. При нажатии кнопки Отмена окно диалога просто закрывается и команда не исполняется. Наличие остальных кнопок зависит от вида команды.

Поле ввода – используется для ввода значения параметра с клавиатуры.

Поле списка – требуемый параметр выбирается из списка. Выбор осуществляется простым щелчком по нужному элементу списка. При необходимости можно воспользоваться линейкой прокрутки.

Раскрывающийся список – используется в целях экономии места на экране. Чтобы раскрыть этот список нажмите на треугольник. В дальнейшем работа с этим списком полностью аналогична работе с обычным списком.

Флажок – это поле может иметь только два значения: включено/выключено и означает, что некоторый режим включен или выключен. Соглашения здесь таковы:

Чтобы включить или отключить флажок нужно просто щелкнуть по флажку.

Переключатель – позволяет выбрать только один из возможных режимов. В этом его различие от флажка. Выбранный режим отмечается кружочком с точкой внутри.

Вкладки – некоторые команды используют очень большое количество различных параметров. В этом случае диалоговые окна оформляются в виде вкладок или подшивок. Чтобы выбрать некоторую группу параметров нужно нажать на требуемую вкладку. Рисунок 6.

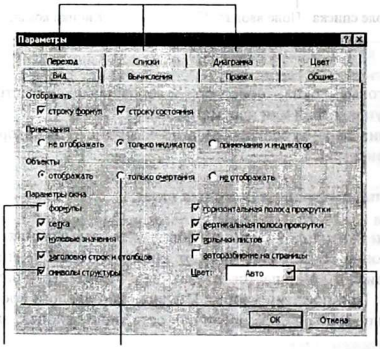


Рис.6

Файловая система Windows – 95

Диски

Любая информация в компьютере хранится на диске. Диск вращается в устройстве подобно компакт – диску в проигрывателе, что позволяет записывать и считывать информацию с диска. Устройство в котором

находится диск называется накопителем или дисководом. Диски бывают гибкие и жесткие, а также компакт – диски (CD – ROM). Жесткие диски находятся внутри компьютера в так называемом системном блоке.

Каждый диск имеет однобуквенное обозначение (имя). Дисковод для гибких дисков имеет обозначение A. Жесткий диск имеет обозначение C. И, наконец, CD-ROM имеет обозначение D.

Большая часть информации в компьютере хранится на жестком диске. Гибкие диски обычно используются для копирования с жесткого диска (или, наоборот, на жесткий диск) информации, например, для перенесения информации с одного компьютера на другой. На жестком диске помещается столько информации, сколько помещается на сотнях (и даже на тысячах) гибких дисках. CD – ROM также имеет очень большой объем. Обычно на CD – ROM поставляется программное обеспечение, игры и т.д.

Файлы, папки, документы

Файл – это область памяти на диске предназначенный для хранения информации. Информацией может быть программа, написанная на каком – либо языке программирования (например, Бейсик, Паскаль, Си), программа готовая к исполнению (в машинном коде), данные, необходимые для работы программы, текстовые документы. Кроме того, в файлах может содержаться: числовые данные, закодированная табличная, графическая и любая другая информация.

Каждый файл имеет ***имя*** и в зависимости от характера содержащейся в файле информации ***тип***.

Имя файла – это последовательность букв, цифр и некоторых знаков, расположенных в произвольном порядке.

Тип (говорят ещё расширение имени файла) – это последовательность из трех букв, цифр или знаков. Между именем файла и типом всегда ставится точка. Например, Годовой отчет.doc

Здесь: Годовой отчет – имя файла, doc – тип файла.

В большинстве случаев пользователю не обязательно присваивать файлу тип. Приложение, в котором обрабатывается данный файл, автоматически присвоит ему нужный тип.

Основной единицей информации, которую обрабатывают приложения Windows – 95 является ***документ***. Каждый документ хранится на диске в виде файла. Программы также хранятся на диске в виде файлов.

Без большой потери строгости можно считать, что файл и документ это одно и тоже. Дело в том, что Windows и Microsoft Office создавались прежде всего для автоматизации учрежденческой деятельности. А в любом офисе или учреждении основная единица информации это документ. В дальнейшем мы будем употреблять оба этих понятия, не делая никаких различий между ними.

Обычно на диске хранятся сотни и тысячи документов. Для упорядочения их используют ***папки***.

Папка – это группа документов, объединенных по какому – либо признаку. Папки можно ассоциировать с выдвижными ящичками стола, в каждом из которых хранятся документы.

Каждая папка должна иметь имя и может, в свою очередь, входить в другую папку как единое целое (вложенные папки). Так образуется иерархическая файловая система Windows. На каждом диске имеется так называемый корневой каталог. В нем регистрируются файлы и папки 1-го уровня. В последних регистрируются файлы и папки следующего 2-го уровня и т.д.

При такой сложной структуре файловой системы для нахождения нужного файла (документа) необходимо указать не только имя документа, но и всю цепочку вложенных папок. Такая цепочка называется ***путем***. При записи пути каждый из вложенных папок отделятся от другого знаком \. Пример записи пути:

C:\Windows\Бухгалтерские документы\Отчеты\Годовой отчет.doc

Здесь C: - имя диска;

Windows – папка верхнего (первого) уровня;

Бухгалтерские документы – вложенная папка (второго уровня);

Отчеты – вложенная папка (третьего уровня);

Годовой отчет.doc – имя документа;

Из примера видно, что папки не имеют расширения имени.

Проводник Windows – 95

С помощью Проводника можно просматривать содержимое дисков и управлять файлами. Под управлением файлами понимается выполнение различных операций над ними как целыми неделимыми объектами (копирование, перемещение, удаление, создание новых папок, удаление ненужных и пр.).

Чтобы открыть окно Проводника выполните одно из следующих действий:

- Если ярлык для Проводника создан и виден на Рабочем столе, щелкните по этому ярлыку;
- Щелкните по кнопке Пуск, выберите пункт Программы, а затем щелкните по пункту Проводник;
- Щелкните правой кнопкой по значку Мой компьютер, затем выберите пункт Проводник;
- Щелкните правой кнопкой по кнопке Пуск, затем выберите пункт Проводник;

Окно Проводника выглядит так, как показано на рисунке 7.

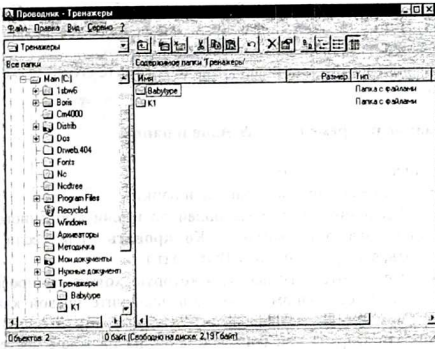


Рис. 7. Окно Проводника

Окно Проводника разбито на две части. Левое окно называется **Все папки**, а правое окно – **Содержимое папки**. В верхней части окна под строкой меню имеется раскрывающийся список, который называется **Переход к другой папке**. В той же строке находится панель инструментов Проводника.

Выбор файла или папки

Чтобы выбрать папку или файл просто щелкните по имени файла или папки. Выделенный объект будет подсвечен. Если вы выделили папку в левом окне, то он откроется автоматически.

Чтобы открыть документ или папку в правом окне нужно дважды щелкнуть по имени документа или папки.

Чтобы перейти на один уровень вверх в иерархии папок щелкните по кнопке с изображением желтой папки с маленькой стрелкой вверх на панели инструментов.

Создание новой папки

Документы близкие по смыслу и одинаковое предназначение удобнее хранить в одной папке (например, документы с отчетами). Для создания новой папки необходимо:

- Открыть папку, в которую нужно вложить новую папку.
- В окне ***Содержимое папки*** (правое окно) щелкнуть в свободном месте правой кнопкой.

- Из появившегося контекстного меню выбрать пункт **Создать**, а затем пункт **Папку.** В списке файлов и папок появится новая папка с именем, присвоенным ей по умолчанию.
- Чтобы изменить имя папки, просто наберите его в строке ввода и нажмите Enter.

Копирование и перемещение файлов и папок

Для выполнения этих действий:

- Выделите нужный файл или папку.
- Щелкните правой клавишей по имени выбранного файла или папки. Выберите команду **Копировать** (или, если хотите переместить, выберите команду **Вырезать**).
- Перейдите в ту папку, в которую хотите копировать или переместить. В свободном месте окна щелкните правой клавишей. Выберите команду **Вставить**

При копировании файл или папка остается на прежнем месте (копия отправляется на новое место). При перемещении файл или папка физически перемещается на новое место.

Удаление файла или папки

Если Вам некоторые файлы и (или) папки не нужны, то можно их удалить, чтобы они не занимали лишнего места на диске.

Для удаления файла:

- Открыть папку, в которой находится удаляемый файл.

Выделить нужный файл. Нажать на правую кнопку мыши

- Из появившегося контекстного меню выбрать пункт **Удалить.**
- На экран выведется диалоговое окно **Подтверждение удаления файла**. Если вы уверены, что хотите действительно удалить этот файл, нажмите кнопку **Да**

Для удаления папки:

Выделить нужную папку. Нажать правую кнопку мыши.

- Из контекстного меню выбрать пункт **Удалить.**
- На экран выведется диалоговое окно **Подтверждение удаления папки**. Если вы уверены, что действительно хотите удалить эту папку – нажмите кнопку **Да.**

К Вашему сведению:

1. При удалении папки удаляются и все вложенные в нее папки и файлы.
2. Все удаляемые папки и файлы помещаются в так называемую *корзину*.
3. Если Вы все же передумали, то ещё можете восстановить папки и файлы, взяв их обратно из корзины. Для этого на рабочем столе откройте корзину, дважды щелкнув по значку корзины. Выберите нужные папки и файлы. В меню **Файл** выберите пункт **Восстановить**.
4. Время от времени не забывайте очищать корзину от "мусора". Для этого в меню **Файл** выберите команду **Очистить**.

Переименование файла или папки

Чтобы переименовать файл или папку надо щелкнуть по имени правой клавишей и выбрать команду **Переименовать**

Выделение группы файлов или папок

Чтобы выделить группу файлов или папок:

1. Если файлы или папки расположены подряд, то нажмите клавишу Shift и, не отпуская её, щелкните по имени первого файла или папки, затем последнего. После чего отпустите клавишу Shift.
2. Если файлы или папки расположены не подряд, то нужно нажать клавишу Ctrl и, не отпуская её щелкать по именам нужных файлов и папок.

С выделенной группой файлов и папок можно выполнять операции, так как будто это один файл. Однако это могут быть только операции копирования, перемещения и удаления.

Создание ярлыков

Ярлыки – это одно из удобств, которые предоставляет Windows – 95. Они позволяют быстро запустить программу или быстро открыть документ простым двойным щелчком по ярлыку на рабочем столе, без долгих поисков нужного файла и копания в папках. Чтобы создать ярлык, надо щелкнуть по имени файла правой клавишей и из контекстного меню выбрать команду **Создать ярлык**.

Присвойте ярлыку имя. Для того чтобы переместить ярлык на рабочий стол нажмите на ярлык мышью и, не отпуская клавишу, перенесите ярлык на рабочий стол, затем отпустите клавишу.

Если вы хотите, чтобы ярлык оставался и в той папке, где он был создан, то прежде чем переносить его на рабочий стол нажмите на клавишу

Ctrl и, не отпуская его, нажмите мышью на ярлык и "тащите" его на рабочий стол.

Кстати, это ещё один способ копирования и перемещения – так называемый метод "перенести – оставить"

Контрольные вопросы:

1. Перечислите и объясните значение элементов управления окнами в Windows – 95;
2. Где находится панель задач и что она отображает?
3. Что такое меню?
4. Как переключиться с русского языка на английский и наоборот?
5. Чем отличается флажок от переключателя?
6. Чем отличается сворачивание окна от её закрытия?
7. Что такое файл?
8. Что такое путь?
9. Как создать новую папку?
10. Чем отличается перемещение от копирования?

ГЛАВА IV. ОСНОВЫ ПРОГРАММИРОВАНИЯ

§4.1. Понятие алгоритма.

Компьютер - это устройство для решения задач. Не обязательно задач чисто математического характера. Это могут быть и шахматные задачи, и задачи управления станками или ракетами, и задачи планирования производства, и задачи информационно- справочного обслуживания. Чтобы решить какую - либо задачу на компьютере необходимо сначала придумать как ее вообще решить, т.е. придумать алгоритм ее решении.

Что такое алгоритм?

Само по себе слово "алгоритм" (algorithm) очень интересно: на первый взгляд может показаться, будто кто-то намеревался написать слово "логарифм" (logarithm), но перепутал порядок первых четырех букв. Имеется старая форма этого слова "algorism", что означает правила выполнения арифметических действий с использованием арабских цифр. В средние века абакисты считали на счетах, алгоритмики использовали зачатки математической символики (арабские цифры и операции +, -, *, / по правилам предложенным аль- Хорезми).

И слово algorism произошло от имени этого ученого Abu Za far Mohammed ibn Musa al- Khowarizmi означающее буквально: "Отец Джафара, Магомет, сын Мусы, уроженец Хорезма". Вышеназванный уроженец Хорезма написал знаменитую книгу: "Kitab al jabr W'al- muqabala" (правила действий над числами). Название книги дало начало другому слову "алгебра", хотя книга было совсем не алгебраической. Постепенно форма и значения слово "algorism" исказилось на "algorithm".

Итак, что понимается под алгоритмом?

Алгоритм - это строгая и четкая, конечная система правил, которая определяет последовательность действий над некоторыми объектами и после конечного числа шагов приводит к достижению поставленной цели.

Из определения алгоритма следует, что он должен удовлетворять следующим требованием:

1) конечность (финитность)

Алгоритм всегда должен заканчиваться после конечного числа шагов. Процедуру обладающую всеми характеристиками алгоритма, за исключением конечности, вызывают вычислительным методом.

2) определенность (детерминированность)

Каждый шаг алгоритма должен быть строго определен. Действия, которые необходимо произвести, должны быть строго и недвусмысленно определены в каждом возможном случае, так чтобы если дать алгоритм нескольким людям, то они действуя по этому алгоритму получали один и тот же результат. Поскольку обычный язык полон двусмысленностей, то чтобы преодолеть это затруднение, для описания алгоритмов разработаны формально определенные языки программирования, или машинные языки, в которых каждое утверждение имеет абсолютно точный смысл.

Запись алгоритма на языке программирования называется программой.

3) Алгоритм должен иметь некоторое число входных данных, т.е. величин, объектов заданных ему до начала работы. Эти данные берутся из некоего конкретного множества объектов.

4) Алгоритм имеет одну или несколько выходных величин, т.е. величин, имеющих вполне определенное отношение ко входным данным.

5) Эффективность

От алгоритма требуют, чтобы он был эффективным. Это означает, что все операции, которые необходимо произвести в алгоритме, должны быть достаточно простыми, чтобы их в принципе можно было выполнить точно и за конечный отрезок времени с помощью карандаша и бумаги.

Попытаемся сравнить алгоритм с рецептом в кулинарной книге. Рецепты обладают свойством финитности, имеют входные данные (соль, мука, яйца и пр.) и выход (званый обед и т.п.), но отличаются печально известным отсутствием определенности. Возьмем например инструкцию: "Добавьте щепотку соли". Они не вполне определена, т.к. неясно сколько означает щепотка. Инструкции типа "Слегка потрясите смесь, пока она не станет комковатой", "подогрейте коньяк в маленькой кастрюльке" являются, возможно, вполне достаточными для опытного повара, но алгоритм должен быть четко определен, настолько чтобы даже компьютер смог следовать инструкции.

Следует отметить, что для практических целей ограничение "финитности" на самом деле не является жестким: используемый алгоритм должен иметь не просто конечное, а предельно конечное, разумное число шагов. Например, в принципе имеется алгоритм, определяющий, является ли начальное положение в игре в шахматы форсированно выигранным для белых или нет. Но для выполнения этого алгоритма требуется фантастически огромный промежуток времени. Пусть имеется компьютер, обладающий быстродействием 100 млн. операций в секунду. Тогда этот компьютер будет выполнять алгоритм в течение 10^{23} лет. Для сравнения: период времени с начала возникновения жизни на земле и до наших дней меньше 10^{23} лет.

Пример алгоритма.

Алгоритм Евклида нахождения наибольшего общего делителя двух целых чисел, т.е. наибольшее целое число, которое делит нацело заданные числа.

1. Рассмотреть А как первое число и В как второе число. Перейти к п.2.

2. Сравнить первое и второе числа. Если они равны, то перейти к п.5. Если нет, то перейти к п.3.

3. Если первое число меньше второго, то переставить их местами. Перейти к п.4.

4. Вычесть из первого числа второе и рассмотреть полученную разность как новое первое число. Перейти к п.2.

5. Рассмотреть первое число как результат.

Стоп.

Этот набор правил является алгоритмом, т.к. следуя ему любой человек умеющий вычитать, может получить наибольший общий делитель для любой пары чисел.

544 *и* 119

$A = 544 \quad B = 119$

$$\begin{array}{r} 544 \\ - \\ \underline{119} \\ 425 \end{array}$$

$A = 425 \quad B = 119$

$$\begin{array}{r} 425 \\ - \\ \underline{119} \\ 306 \end{array}$$

$A = 306 \quad B = 119$

$$\begin{array}{r} 306 \\ - \\ \underline{119} \\ 187 \end{array}$$

$A = 187 \quad B = 119$

$$\begin{array}{r} 187 \\ - \\ \underline{119} \\ 68 \end{array}$$

$A = 68 \quad B = 119$

$A < B$

Меняем местами

$A = 119 \quad B = 68$

$$\begin{array}{r} 119 \\ - \\ \underline{68} \\ 51 \end{array}$$

$A = 51 \quad B = 68$

$A < B$

$A = 68 \quad B = 51$

$$\begin{array}{r} 68 \\ - \\ \underline{51} \\ 17 \end{array}$$

$A = 17 \quad B = 51$

$A < B$

$A = 51 \quad B = 17$

$$\begin{array}{r} 51 \\ - \\ \underline{17} \\ 34 \end{array}$$

$A=34 \quad B=17$

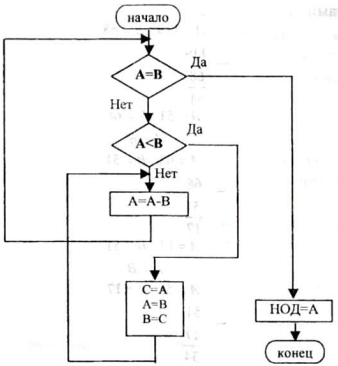
$$\begin{array}{r} -\ 34 \\ \hline 17 \\ \hline 17 \end{array}$$

A=B=17=НОД

Данный способ записи алгоритмов возможен, но неудобен. Во-первых, нет наглядности, во-вторых, "многословен". Одним из способов записи алгоритма являются блок-схемы. В них используют математическую символику для обозначения действий, но разрешается записывать и словами. Блок-схемой называется такое графическое изображение структуры алгоритма, в котором каждый этап или шаг процесса переработки данных представляется в виде прямоугольника, ромба, овала или другой геометрической фигуры, называемой блоком.

Эти фигуры соединяются между собой линиями со стрелками, отображающими последовательность выполнения алгоритма. Внутри каждой фигуры разрешается писать произвольный текст, в котором на понятном человеку языке сообщаются о нужных вычислениях в соответствующей части программы.

Приняты определенные стандарты графических обозначений. Так, прямоугольник, обозначает вычислительные действие в результате которых изменяются значения данных. Ромбом обозначают этап разветвления алгоритма. Выбор одного из двух возможных направлений дальнейшего счета производится в зависимости от выполнения условия, записанного в ромбе. Овалом обозначают начало и конец алгоритма. Запишем алгоритм Евклида в виде блок-схемы:



Итак, чтобы решить какую либо задачу, нужно придумать алгоритм ее решения. Затем следует представить этот алгоритм в таком виде, чтобы данный компьютер мог его выполнить. Для этого нужно, во-первых, разбить алгоритм на элементарные операции которые умеет выполнять компьютер, и, во- вторых, записать каждую такую операцию на языке понятом компьютеру. Такая запись алгоритма на языке компьютера называется программой. Процесс разработки программы называется программированием. А человек, выполняющий эту работу - программистом.

Программирование – это научная дисциплина. Если бы процессы программирования разных задач не имели между собой ничего общего, то программирование, как таковое, не было бы научной дисциплиной. Но дело обстоит не так. Существуют общие методы, которые позволяют, постепенно расчленяя задачи на подзадачи, сводить их решение, в конечном счете к некоторым элементарным операциям (чаще всего к элементарным арифметическим операциям), подобно тому, как разбирая совершенно непохожие сложные механизмы, мы обнаруживаем, что они состоят из одинаковых деталей и узлов, только по разному соединенных (напр. подшипники, болты, гайки и т.д.). Из этого, конечно, не следует что процесс программирования не содержит в себе элементов творчества. Составление программы, такой же творческий процесс, что и разработка сложного механизма из заданных наборов деталей.

Что же касается компьютеров, то, при всем их разнообразии, с точки зрения разработки алгоритмов все они очень похожи. Алгоритм можно составить один для всех типов машин. Основное назначение алгоритмов заключается в их фактическом выполнении тем или иным исполнителем. Т.е. алгоритм составляется для того, чтобы он был выполнен для решения какой либо задачи. Основное назначение компьютера состоит в фактическом исполнением алгоритмов, разработанных человеком. Таким образом, компьютер выступает как инструмент, приспособление человека для исполнения алгоритма.

Важно понять, что компьютер сам по себе ничего не делает. Он лишь слепо выполняет то, что укажет ему человек. Поэтому, когда говорят, что компьютер управляет станками, сочиняет музыку, играет шахматы «как бы самостоятельно», то это неверно!

Это человек придумывает алгоритм, т.е. способ управления станками, сочинения музыки, игры в шахматы, а компьютер лишь исполняет этот алгоритм. Таким образом, компьютер это помощник, инструмент человека для решения различных задач, но инструмент очень мощный, разноплановый. Это настоящий "пожиратель" алгоритмов, причем ненасытный. С помощью компьютера человек может решать самые разнообразные задачи. При этом человек выполняет наиболее сложную, творческую часть работы – составляет, придумывает, "творит" алгоритм решения задач, а рутинную не творческую часть работы, связанную с большими и трудоемкими вычислениями поручает выполнить компьютеру.

Рассмотрим несколько примеров разработки алгоритмов.

1. Задача о поездах и мухе.

Пусть два города А и В удалены на расстояние d=600 км. В одно и то же время из каждого города отходят поезда в направлении к другому городу. Поезд, вышедший из А имеет скорость v_1=40 км/час, а из В имеет скорость v_2=60 км/час. Одновременно из пункта А вылетает исключительно быстрая муха со скоростью v=200 км/час и летит навстречу поезду, вышедшему из пункта В. При встрече с ним муха делает поворот и летит обратно навстречу с поездом, идущему из пункта А. Когда она его встретит, муха снова делает поворот и летит в обратном направлении, и так продолжается до тех пор, пока поезда не встретятся.

Задание:

1) определить длину различных отрезков пути, которые пролетит муха.
2) определить общее расстояние, которое пролетит муха.

На второй вопрос легко ответить. Поезда встретятся через 600/(40 + 60)=6 часов, а муха за это же время пролетит расстояние 200x6=1200 км. Но оставим в стороне эту хитрость, а будем решать задачу в "лоб". Прежде всего, нужно составить алгоритм, т.е. придумать – как решить задачу с помощью элементарных шагов. Алгоритм будем записывать в виде блок-схемы. С чего нужно составлять блок-схему? Как правило, блок-схема начинается с ввода исходных данных и начальных установок, т.е. определение начальных значений некоторых вспомогательных переменных. Но вначале бывает трудно сразу определить чему будут равны начальные значения вспомогательных переменных, поэтому сначала рисуют центральную часть блок-схемы (ядро) в которой определены те действия, которые решают задачу, причем сначала рисуют укрупненную блок-схему, чтобы как можно яснее представить себе весь процесс решения.

Какие идеи сразу приходят в голову? Самая простая идея заключается в том, чтобы производить действия следующим образом:

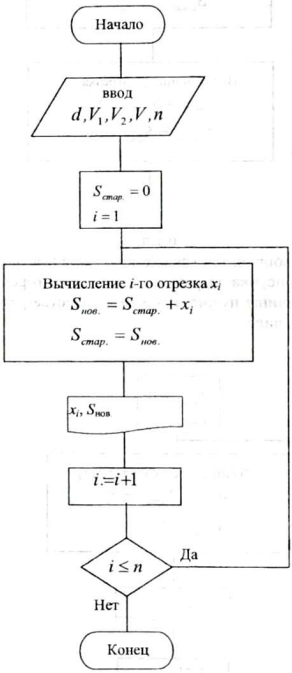
1. вычислить длину первого отрезка пути, который пролетит муха.
2. вычислить длину второго отрезка и прибавить к предыдущему результату.
3. вычислить длину третьего отрезка и сложить с ранее полученной суммой и т.д.

Мы можем этот процесс представить в виде:

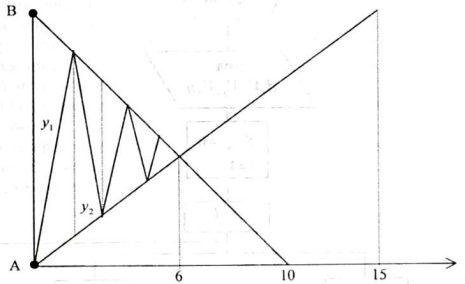
и т.д.

Тут же возникает вопрос: так как схема должна иметь конечную длину, то нужно средствами рисунка выразить понятие повторения, кроме того, надо указать какие величины нужно вывести в качестве результата. Отсюда схему перерисовываем в виде:

Случается, что когда некоторая последовательность операций закончена, необходимо бывает вернуться и повторить эти же операции. Это называется циклом. На этой схеме виден серьезный недостаток – вычислительный процесс по этой схеме никогда не закончится. Необходимо придумать критерий окончания алгоритма. Пока не совсем ясно, будем считать, что нам задано число повторений n, а также заданы все исходные данные. Тогда блок-схема примет вид:



Блок-схема приобрела более или менее законченный вид, но мы так и не продвинулись далеко, т.к. по прежнему не ясно что значит вычисление отрезка x_i. Иногда помогает рисунок. Попробуем графически изобразить процесс.



$$t_1 = \frac{d}{(V + V_2)}, \qquad y_1 = d - t_1(V_1 + V_2), \qquad x = t_1 V;$$

$$t_2 = \frac{d}{(V + V_1)}, \qquad y_2 = d - t_2(V_1 + V_2), \qquad x = t_2 V;$$

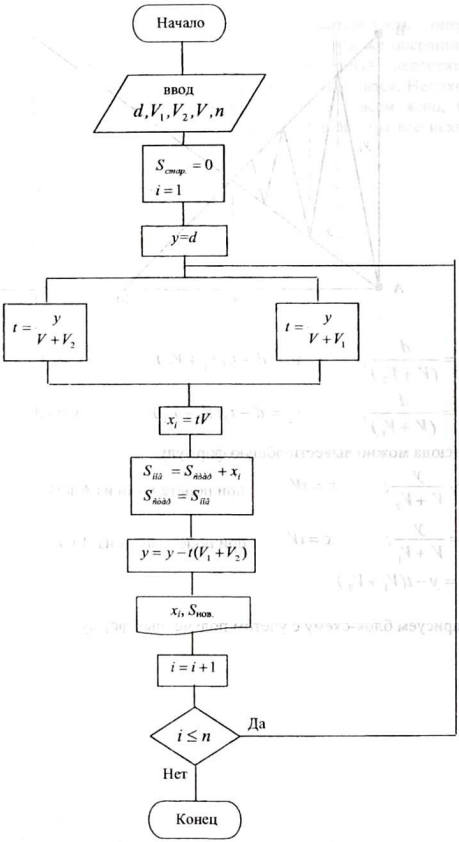
Отсюда можно вывести общую формулу:

$t = \frac{y}{V + V_2}$; $\quad x = tV$ при полете мухи из А к В.

$t = \frac{y}{V + V_1}$; $\quad x = tV$ при полете мухи из В к А.

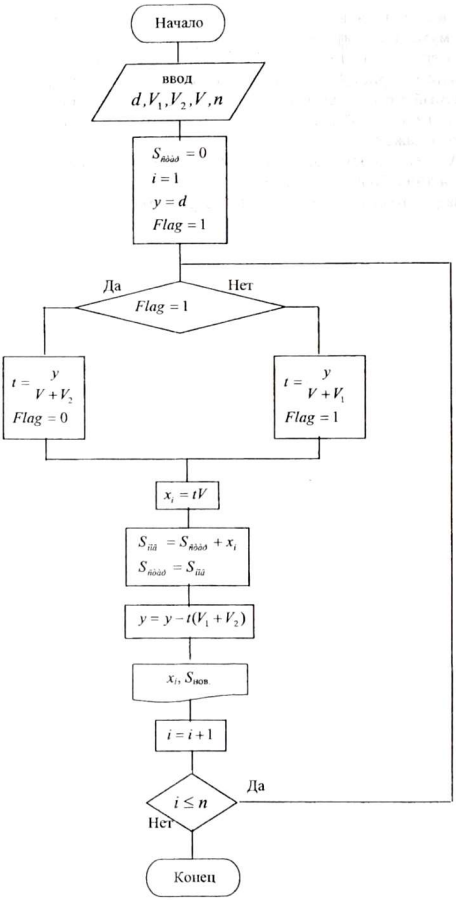
$y = y - t(V_1 + V_2)$

Нарисуем блок-схему с учетом полученных формул:


Начало
ввод
d, V_1, V_2, V, n
$S_{стар.} = 0$
$i = 1$
$y = d$
$t = \frac{y}{V + V_2}$
$t = \frac{y}{V + V_1}$
$x_i = tV$
$S_{щё} = S_{подд} + x_i$
$S_{подд} = S_{щё}$
$y = y - t(V_1 + V_2)$
$x_i, S_{нов.}$
$i = i + 1$
$i \le n$
Да
Нет
Конец

Здесь мы видим, что алгоритм должен разветвляться на две ветви (когда муха летит к поезду В и когда летит к поезду А). Как сообщить компьютеру, что нужно попеременно проходить через эти ветви. Используется прием, который широко используется в программировании, так называемый метод "флажков". Будем считать, что если флажок поднят, то нужно идти по левой веточке, если опущен, то по правой. Но как компьютер "увидит" флажок?

А очень просто. Заведем целочисленную переменную Flag (впрочем, можно и типа Boolean). Если Flag=1, то будем считать, что флажок поднят, если Flag=0, то опущен. Вот и все! Перерисуем блок-схему.


Начало
ввод
d, V_1, V_2, V, n
$S_{\text{общ}} = 0$
$i = 1$
$y = d$
$Flag = 1$
Да
Нет
$Flag = 1$
$t = \frac{y}{V + V_2}$
$Flag = 0$
$t = \frac{y}{V + V_1}$
$Flag = 1$
$x_i = tV$
$S_{\text{ш}} = S_{\text{общ}} + x_i$
$S_{\text{общ}} = S_{\text{ш}}$
$y = y - t(V_1 + V_2)$
$x_i, S_{\text{ш}}$
$i = i + 1$
$i \le n$
Да
Нет
Конец

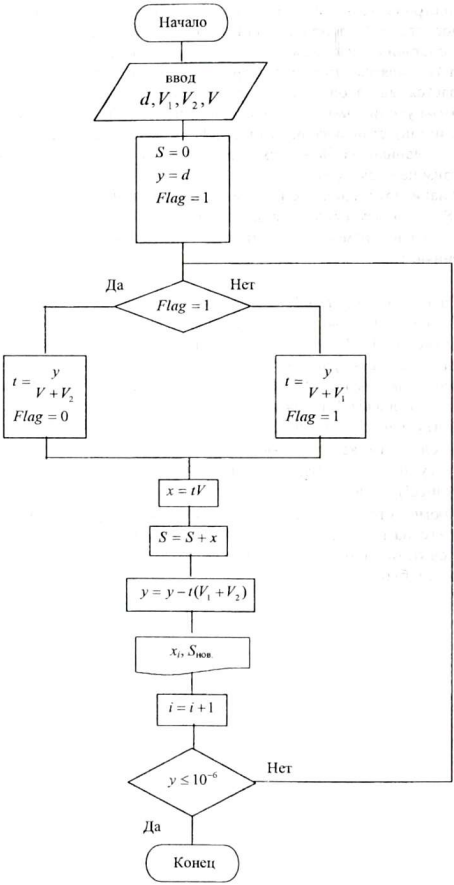
Построив один вариант блок-схемы, всегда нужно посмотреть, нельзя ли упростить ее? Распространенная ошибка начинающих программистов, наспех составив блок-схему алгоритма (а чаще всего, даже не разработав алгоритм), садятся за компьютер и начинают кодировать. В результате порождается масса ошибок и, в конечном счете, времени на разработку программы уходит намного больше, чем нужно. Вы видели, на примере этой достаточно простой задачи, что мы действовали не спеша, и нарисовали уже несколько вариантов блок-схемы алгоритма и все же еще раз, внимательно посмотрим на блок-схему.

Анализируя блок-схему, мы видим, что попеременно используются $S_{стар}$ и $S_{нов}$, причем после вывода на печать $S_{нов}$ его значение нам не нужно, оно все равно изменяется. Отсюда, можно использовать только одну переменную S.

$$S = S + x_i$$

Далее: мы критерий окончания алгоритма определили не совсем хорошо. Действительно, не совсем ясно, чему равно максимальное n. Может 100, а может 1000. Допустим, мы приняли, что n=100, а на самом деле число отрезков оказалось равным 10, тогда 90 раз алгоритм будет работать "впустую", так как полученные результаты будут бессмысленными (поезда давно встретились!). Как быть? Не лучше ли определить конец алгоритма по y. Действительно, из рисунка видно, что $y \to 0$, что соответствует действительности, ведь y означает расстояние между движущимся навстречу друг другу поездами. Будем считать, что поезда встретились (упаси бог столкнулись!), если $y \le 10^{-6}$.

Кроме того, мы видим, что и значение очередного отрезка x_i после вывода его на печать, нам не нужно, т.е. параметр i можно совсем убрать. Будем считать, алгоритм дальше не упростить. Окончательно получаем следующую блок-схему:


Начало
ввод
d, V_1, V_2, V
$S = 0$
$y = d$
$Flag = 1$
Да
Нет
$Flag = 1$
$t = \frac{y}{V + V_2}$
$Flag = 0$
$t = \frac{y}{V + V_1}$
$Flag = 1$
$x = tV$
$S = S + x$
$y = y - t(V_1 + V_2)$
$x_i, S_{нов}$
$i = i + 1$
$y \le 10^{-6}$
Нет
Да
Конец

§4.2. Введение в язык Турбо – Паскаль

Язык Паскаль, изобретенный в начале 70-х годов 20 века Н. Виртом и названный в честь французского математика и философа Блеза Паскаля, является одним из наиболее распространенных языков программирования. От других языков он выгодно отличается возможностью более ясно и логично записывать программы.

Здесь мы рассмотрим лишь основные конструкции и возможности языка. Более подробно изучение языка Паскаль предусмотрено в специальном курсе "Алгоритмические языки и программирование", читаемого на факультете кибернетики и информационных технологий Ошского технологического университета для студентов специальности "Программное обеспечение вычислительной техники и автоматизированных систем".

Для первого знакомства с языком достаточно тех сведений, которые изложены в этой книге.

Программа на языке Паскаль состоит из двух частей: описание действий, которые должны быть выполнены и описание данных, над которыми они выполняются. В тексте программы описание данных предшествует описанию действий. В этом выражается общее правило языка – каждый встречающийся в программе объект должен быть предварительно описан.

Описание данных состоит из описания переменных. Операторами называются действия над данными. В общем виде любая Паскаль – программа имеет вид:

заголовок программы
раздел описания переменных
раздел операторов

Заголовок программы имеет вид:

PROGRAM имя программы;

Здесь слово PROGRAM – это так называемое ключевое (или служебное или еще говорят зарезервированное) слово. Оно должно записываться именно так, а не иначе. Допускается использовать как строчные, так и прописные буквы. Записи PROGRAM, Program, ProgRam – разрешены и означают одно и то же.

Так начинаются все программы, написанные на языке Паскаль. Здесь нечего понимать, просто так принято разработчиком языка.

Имя программы – это любая последовательность букв, цифр и некоторых знаков, начинающихся с буквы. Такие последовательности называются идентификаторами. Идентификатор не должен совпадать ни с одним из ключевых слов. Из знаков допускаются использовать при записи

идентификаторов знак _ (подчеркивание), знаки $, @. В идентификаторе не должно быть (.) – точки, (,) – запятой, ( ) – пробелов, знаков операций.

Примеры правильных идентификаторов:

```
x1
Summa
VOLVO
Select_screen_color
```

Примеры неправильных идентификаторов:

```
3x   начинается с цифры
JAN.3   внутри идентификатора есть точка
VOL  VO   есть пробел
```

Переменные. Стандартные типы.

Каждая переменная имеет имя и тип. Имя переменной – это произвольный идентификатор. В дальнейшем будем говорить "переменная x", вместо "переменная с именем x"

Тип переменной определяет множество её возможных (допустимых) значений.

В Паскале существуют следующие стандартные типы переменных: `integer` (целый), `real` (вещественный), `boolean` (логический), `char` (символьный), `string` (строковый).

Значениями переменных целого типа являются целые (и только!) числа.

Примеры целых чисел:

25 +150 -200 10000

Операции над целыми числами таковы:

+ (сложение), - (вычитание), * (умножение), div (деление нацело), mod (остаток от деления двух целых чисел).

Значениями переменных вещественного типа являются вещественные числа. Определены следующие операции над вещественными числами:

+ (сложение), - (вычитание), * (умножение), / (деление).

Запись вещественных чисел похожа на общепринятую, только вместо запятой используется точка и вместо степени 10 используется буква E.

Пример.

Общепринятая	на Паскале
5,30	5.30
-1,0	-1.0
41000	41000 или 4.1E4
-0,0073	-0.73E-2

Значениями переменных логического типа является true (истина), false (ложь). Определены операции: not (не), and (и), or (или).

Значения переменных символьного типа – символы из таблицы ASCII, о котором речь пойдет позже.

Значения переменных строкового типа – цепочка символов. При записи констант символьного и строкового типа используют одиночные кавычки.

Пример.

'A' - это символ A

'Это цепочка символов'

Операции отношения

Существуют следующие операции отношения:

= равно, <> не равно, < меньше , > больше, <= меньше или равно, >= больше или равно.

Результатом этих операций являются логические значения true или false.

Следует отметить, что над символьными переменными определены лишь операции = и <>.

Раздел описаний переменных

Этот раздел имеет вид:

var описание 1; описание 2; ...; описание n;

var – ключевое слово (от английского variable – переменная)

описание имеет вид:

переменная 1, переменная 2, ..., переменная m: тип;

переменная 1, переменная 2, ..., переменная k: тип;

.....

переменная 1, переменная 2, ..., переменная s: тип;

тип – одно из ключевых слов: `integer, real, boolean, char, string`

Пример раздела описаний:

```
var a,b,c,x,y: real;
    i,j,k,m,n: integer;
    FLAG: boolean;
    symbol: char;
```

Выражения. Порядок выполнения операций.

Совокупность переменных и констант, соединенных знаками операций и скобками, называется выражением.

Пример.

```
(a+b)/c
((n+q1)*dx+(i+j)*dy)/(x1-2)*(y1-2)
```

Правила выполнения операций в Паскале:

1. Умножение и деление выполняются раньше, чем сложение и вычитание. Говорят также, что умножение и деление имеют более высокий приоритет, чем сложение и вычитание.

2. Операции, имеющие одинаковый приоритет выполняются слева направо.

Умножение и деление имеют одинаковый приоритет, сложение и вычитание имеют также одинаковый приоритет. Исходя из этих правил выражение

4/8*5

будет вычисляться следующим образом:
Сначала будет вычислено 4/8 (=0.5), а затем результат будет умножен на 5. Получится 0.5*5=2.5

Всякое отклонение от этих правил должно регламентироваться скобками, т.к. действия над переменными стоящими в скобках выполняются в первую очередь.

Константы

В Паскале есть возможность присвоить константе имя, при этом в последующем тексте программы всюду вместо этой константы можно использовать её имя. Все определения констант перечисляются в специальном разделе – разделе описания констант, имеющем вид:

```
const
      имя 1=значение 1;
      имя 2=значение 2;
      ……..
      имя n=значение n;
```

Пример.

```
const
      r=1.87E+5;
      g=981E-2;
      atmosph=760;
      pi=3.14159;
```

Давая имена константам, мы делаем программу более понятной. Запись `2*pi*r` гораздо понятнее и информативнее, нежели запись 2*3.14159*1.87E+5

Кроме того, при внесении изменений в программу нам будет достаточно изменить только значение константы.

Основные операторы Паскаля

Основная часть программы на Паскале – раздел операторов. Он начинается ключевым словом `begin` и заканчивается ключевым словом `end`, за которым следует точка. Рассмотрим основные операторы:

Оператор присваивания

Элементарное действие над переменной – изменение её значения. Для этого применяется оператор присваивания, имеющий вид:

имя переменной:= выражение;

В нем переменная и выражение должны быть одного типа.

Пример.

Пусть x – переменная целого типа.

```
x:=x+1;
```

В левой части оператора x обозначает переменную, а в правой части – число, являющаяся её текущим значением. Выполнение этого оператора приводит к увеличению значения переменной x на единицу.

Операторы ввода/вывода

Во время исполнения программы она обменивается информацией с "внешним миром". Например, она может выдавать информацию на экран или получать информацию с клавиатуры. Для этого используются операторы ввода и вывода.

Оператор вывода имеет вид:

```
write(выражение);
```
или

```
writeln(выражение);
```

В результате выполнения этого оператора значение соответствующего выражения будет выведено на экран. Выражение может быть любым из указанных выше типов.

Пример.

```
write(2+2);
```
будет выведено на экран 4

```
write(x=y);
```
будет выведено true или false в зависимости от значений `x, y`

Оператор `writeln` отличается от оператора `write` тем, что выведет значение выражения с начала новой строки, а оператор write с той позиции строки, где находится курсор.

В операторе вывода можно указывать несколько выражений, разделяя их запятыми, а также любой текст, заключенный в одинарные кавычки.

Пусть значение A=5. Тогда при выполнении оператора

```
writeln('Значение A=', A);
```

будет выведено на экран с новой строки

Значение A=5

Оператор ввода имеет вид:

read (имя переменной 1, имя переменной 2,..., имя переменной n);
readln (имя переменной 1, имя переменной 2,..., имя переменной n);

В результате его выполнения переменной (переменным) присваивается считанное с клавиатуры значение. Вводимое значение должно записываться при вводе так, как описана переменная, т.е. если например, переменная целого типа, значит вводимое число должно быть целым. При этом, если используется read, то значение вводится в то место на экране, где в данный момент находится курсор, если же используется readln, то с новой строки и с первой колонки экрана.

Пример программы с использованием операторов ввода/вывода.

```
Program summa;
var result, A, B: integer;
begin
        read(A,B);
        result:=A+B;
writeln('Сейчас появится результат');
writeln('1-е введенное число=',A);
writeln('2-е введенное число=',B);
writeln('Сумма двух чисел=',result);
end.
```

Операции с целыми числами

До сих пор мы рассматривали лишь одну операцию с целыми числами – сложение. Естественно в Паскале разрешены и другие операции. Рассмотрим программу, показывающую все возможные операции с целыми числами:

Пример.

```
Program operation;
var result, A, B, C: integer;
begin
        read(A,B);
        writeln('A=',A,'B=',B);
        C:=A+B;
        writeln('Демонстрация сложения, C=',C);
        C:=A*B;
        writeln('Демонстрация умножения, C=',C);
        C:=A div B;
        writeln('Демонстрация деления нацело, C=',C);
        C:=A mod B;
        writeln('Остаток от деления, C=',C);
        C:=A-B;
```

```
    writeln('Демонстрация вычитания, C=',C);
end.
```

Из этой программы видно, что знак умножения * (а не х). Тем самым избегают возможности спутать его с буквой х.

Далее с целыми числами определены операции:

`div` (от английского divide – делить) – деление нацело.

Пусть A=17, B=3, отсюда 17:3=5*3+2 и `A div B` дает результат 5

`mod` (от английского modulo – определить остаток) – определение остатка от деления нацело.

`A mod B` дает результат 2

Стандартные функции с целыми переменными (аргументами)

Рассмотрим программу:

```
Program func;
var a, b, c: integer;
begin
    a:=-2;
    b:=abs(a);
    writeln('abs(-2)=',b);
    c:=sqr(b);
    writeln('sqr(b)=',c);
    c:=sqr(b+b);
    writeln('sqr(b+b)=',c);
end.
```

Оператор `b:=abs(a);` присваивает переменной `b` абсолютное значение числа `a`.

Под абсолютным значением числа понимается значение этого числа, если отбросить знак.

```
abs(-2)=2
abs(-10)=10
abs(5)=5
```

Оператор `c:=sqr(b)` - присваивает переменной с квадрат числа b, т.е. $c = b^2$, sqr (от английского square – квадрат). Число в скобках называется аргументом функции. В качестве аргумента может быть выражение, например, sqr($b^2 - 4ac$).

Операции с вещественными числами (тип Real).

С вещественными числами можно выполнять различные операции. Все возможные операции иллюстрируются следующей программой:

```
Program Real_Ex;
var a, b, c: real;
begin
     a:=17.3;
     b:=3.4;
     c:=a*b;
     writeln('a*b=',c);
     c:=a/b;
     writeln('a/b=',c);
     c:=a+b;
     writeln('a+b=',c);
     c:=a-b;
     writeln('a-b=',c);
end.
```

Одновременное использование вещественных и целых чисел.

В программе могут встречаться переменные разных типов:

```
Program Int_Real;
var n, k: integer;
       a, b: real;
begin
     a:=3.6;
     n:=4;
     b:=n;
     writeln('b=', b);
     n:=trunc(a);
     writeln('trunc(3.6)=',n);
     k:=round(a);
     writeln('round(3.6)=',k);
end.
```

В программе мы видим запись `b:=n;` где вещественной переменной `b` присваивается значение целой переменной `n`. Кроме того, в таких записях как `b:=n+4.6;` или `b:=3*7.2+n;` встречаются вещественные и целые числа, стоящие в правой части выражения. Такие записи разрешены. И наоборот, присвоение вещественных значений целым переменным просто запрещены. Для этого используются стандартные функции `trunc` и `round`. С помощью функции `trunc` производится преобразование вещественного числа в целое путем отбрасывания всех цифр, стоящих после десятичной точки (`truncate` – усекать), `round` – позволяет преобразовать вещественное число в целое путем округления (`round` – округлять).

Другие стандартные функции с вещественными аргументами

Функция	Запись на Паскале
x^2	sqr(x)
$\sqrt{x}$	sqrt(x)
$\sin x$	sin(x)
$\cos x$	cos(x)
$arctgx$	arctan(x)
$\ln x$	ln(x)
e^x	exp(x)

Еще раз напоминаю, что аргументом функции (т.е. то, что стоит в скобках) может быть выражение. Например,

```
s:=sin(a+b*2/5.2-sqr(a));
```

Булевые переменные и условные операторы

Булевые переменные (логические переменные) – это переменные имеющие только два значения `false` и `true`. Над булевыми переменными определены логические операции `not, and, or`.

Кроме того, определены следующие операции отношения:

= равно, <> не равно,

< меньше, > больше

<= меньше или равно, >= больше или равно

Результатом этих операций отношения являются логические `false` или `true`.

Рассмотрим следующее выражение:

`x>3`

В зависимости от значения x это выражение будет либо истинным (`true`), либо ложным (`false`).

Пример.

```
PROGRAM LOGICA;
VAR X: INTEGER;
BOL: BOOLEAN;
BEGIN
     X:=4;
     BOL:=X>3;
     WRITELN('BOL=',BOL);
     BOL:=X<3;
     WRITELN('BOL=',BOL);
END.
```

Допускается размещать справа и слева от знаков отношений арифметические выражения: X+6.5<X+5→ такие выражения называются логическими или булевыми выражениями.

Рассмотрим программу где используются логические функции:

```
PROGRAM  LOG1;
VAR X: INTEGER;
L1, L2, RESULT: BOOLEAN;
BEGIN
     X:=4;
     L1:=X>3;
     L2:=X<3;
     WRITELN ('БУЛЕВАЯ ПЕР.L1=',L1);
     WRITELN ('БУЛЕВАЯ ПЕР.L2=',L2);
     RESULT: =L1 AND L2;
     WRITELN('L1 AND L2 РАВНО',RESULT);
     RESULT : =L1 OR L2;
     WRITELN('L1 OR L2 РАВНО',RESULT);
     RESULT: = NOT RESULT;
     WRITELN('NOT RESULT РАВНО',RESULT);
END.
```

Условные операторы

Условные операторы – это такие операторы, с помощью которых можно изменять последовательность выполнения операторов программы.

Оператор IF ....THEN

Этот оператор имеет вид:

```
IF   условие THEN  оператор;
```

где оператор – любой оператор Паскаля;
условие – логическое выражение.

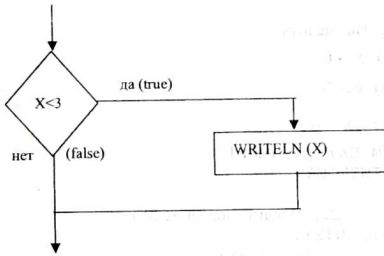
Если это условие выполняется (т.е. это условие дает значение TRUE), то будет выполнен оператор стоящий после слова THEN.

Рассмотрим пример:

```
IF X<3 THEN WRITELN (X);
```

Здесь записано : "Если X<3 , то вывести на экран значение X".

Оператор IF ....THEN можно представить в виде структурной схемы, с помощью которой показывается ход выполнения программы:



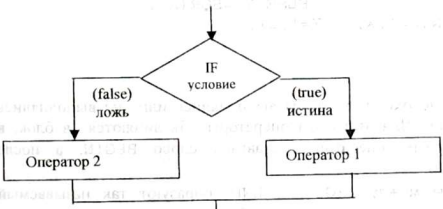
Оператор IF …THEN…..ELSE

Этот оператор записывается следующим образом:

```
IF   условие THEN оператор 1
             ELSE оператор 2;
```

В этом операторе будет выполняться либо оператор 1, либо оператор 2, что иллюстрирует структурная схема.

Обратите внимание, что перед ELSE ; не ставится



Из этой структурной схемы видно, что выбирается либо один, либо другой вариант продолжения программы.

Например:

```
IF X<2 THEN   X1:=0
       ELSE   X1:=1;
```

Пример. Вычислить:

$$y = \begin{cases} x, x > 0 \\ 0, x = 0 \\ x^2, x < 0 \end{cases}$$

```
PROGRAM EX7;{Вариант 1}
VAR X:INTEGER;
BEGIN
     WRITELN('Введите значение X')
     READLN(X);
     IF X>0 THEN Y:=X;
     IF X=0 THEN Y:=0;
     IF X<0 THEN Y:=SQR(X)
     WRITELN('X=',X,'Y=',Y);
     READLN;
 END.
 PROGRAM EX8;{Вариант 2}
 VAR X:INTEGER;
 BEGIN
      WRITELN('Введите значение X')
      READLN(X);
      IF X>0 THEN Y:=X
            ELSE IF X=0 THEN Y:=0
                        ELSE Y:=SQR(X);
      WRITELN(X=',X,'   Y=',Y);
 READLN;
 END;
```

Часто бывает необходимо, чтобы выполнялись или не выполнялись несколько операторов. Для этого эти операторы объединяются в блок, в котором перед первым оператором ставится слово BEGIN, а после последнего слово END.

Все операторы между BEGIN и END образуют так называемый составной оператор, а BEGIN и END как бы выполняют роль скобок. Их часто так и называют – операторные скобки.

Пример:

```
PROGRAM DEMO_IF;
VAR  X: INTEGER;
BEGIN
     READ(X);
   IF X<2 THEN
     BEGIN
```

```
            WRITELN('ВЫПОЛНЯЕТСЯ ВЕТКА ПРОГРАММЫ ПО
                    УСЛОВИЮ TRUE');
          WRITELN('X = ', X);
    END;
  ELSE WRITELN('X = ', X) ;
END.
```

Иллюстрация в общем виде:

```
BEGIN
S1;
S2;
S3;
S4;
S5;
END.
```

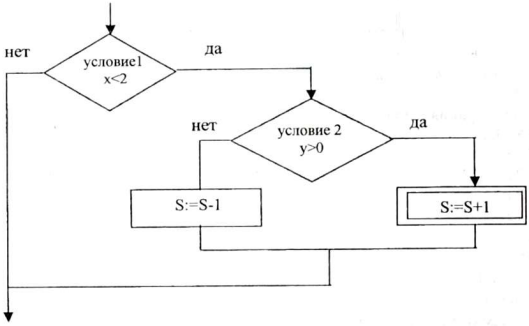
где S1, S2, S3, S4, S5- операторы Паскаля , в том числе и составные операторы, т. е. составные операторы могут быть вложены друг в друга .Схематично это выглядит так:

```
BEGIN                    BEGIN
 .........                S1;
     BEGIN                S2;
      .........           S3;
      ........               BEGIN
     END;                      P1;
      .........                 P2;
      .......                   P3;
END;                          END;
                          S5;
                         END;
```

Здесь S4 составной оператор.

Условные операторы также могут быть вложены друг в друга.

```
IF X<2 THEN
     IF Y>0 THEN S:=S+1
            ELSE  S:=S-1;
```



Операторы цикла

Операторы цикла используются для организации многократного повторения выполнения одних и тех же операторов. В языке Паскаль существует три типа операторов цикла.

Оператор цикла с предусловием. Этот оператор имеет вид:

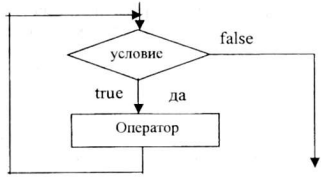
```
while  условие
  do   оператор;
```

условие – булевское выражение;

оператор – любой оператор Паскаля, в частности может быть и составным оператором. Слова `while` и `do` являются служебными словами. Выполняется этот оператор следующим образом: сначала вычисляется значение булевского выражения. Если это значение есть `true`, то выполняется оператор после слова `do` и снова происходит возврат к вычислению булевского выражения. Так повторяется пока булевское выражение имеет значение `true`. Как только значение булевского выражения станет `false`, то происходит выход из цикла, т. е. оператор после служебного слова `do` уже не выполняется, а будет выполняться следующий после оператора цикла оператор.

В данном операторе вычисление выражения происходит раньше, чем будет выполняться оператор после `do`, поэтому он и называется оператор цикла с предусловием. Может так случиться, что оператор после `do` не будет выполнен вообще, если значение условия с первого раза будет `false`.

Структурная схема цикла с предусловием



Оператор цикла с постусловием

Этот оператор имеет вид:

```
repeat оператор
until  условие;
```

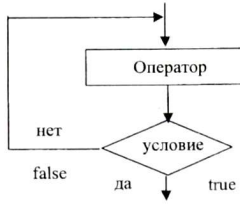
где оператор – любой оператор Паскаля, в том числе и составной.

условие – булевское выражение.

`repeat` и `until`-служебные слова.

Этот оператор выполняется следующим образом: сначала выполняется оператор следующий за служебным словом `repeat`, затем вычисляется значение булевского выражения (условия). Если значение условия `false`, то происходит возврат к выполнению оператора и после этого снова вычисляется значение булевского выражения. Так повторяется до тех пор, пока значение булевского выражения `false`. Как только условие станет `true`, выполнение оператора цикла прекращается.

Структурная схема оператора



В отличие от оператора цикла `while - do` здесь оператор будет выполнен хотя бы один раз, независимо от значения условий.

Предупреждение! Чтобы рассмотренные выше операторы цикла выполнялись конечное число раз, при построении цикла необходимо предусмотреть, чтобы среди выполняемых операторов обязательно был оператор, который изменял бы значение условия, таким образом, чтобы когда-нибудь значение условия принимало бы `false` (для оператора `while-do`) или `true` (для оператора `repeat- until`).

В противном случае цикл будет повторяться бесконечное число раз и компьютер "зациклится". Ответственность за правильное применение этих операторов цикла несет на себе программист!

Пример: Вычислить $S = \sum_{X=1}^{100} X$

```
PROGRAM SUM1; {Вариант 1}
VAR X, SUM : INTEGER;
  BEGIN
       SUM:=0;
       X:=1;
  WHILE X<=100 DO
       BEGIN
            SUM:=SUM+X;
            X:=X+1;
       END;
       WRITELN('SUM=',SUM);
       READLN;
  END.

PROGRAM SUM2; {Вариант 2}
VAR X, SUM : INTEGER;
  BEGIN
       SUM:=0;
       X:=1;
       REPEAT
            SUM:=SUM+X;
            X:=X+1;
       UNTIL X>100
       WRITELN('SUM=',SUM);
       READLN;
  END.
```

Пример: Вычислить функцию:

$$y = \begin{cases} x^2 + 1, x > 0 \\ 0, x = 0 \\ x^2 - 1, x < 0, \end{cases}$$ для x ∈[-10, 10] с шагом 1

```
PROGRAM EX9;{Вариант 1}
VAR X,Y:INTEGER;
BEGIN
    X:=-10;
    WHILE X<=10 DO
      BEGIN
           IF X>0 THEN Y:=SQR(X)+1;
           IF X=0 THEN Y:=0;
           IF X<0 THEN Y:=SQR(X)-1;
         WRITELN(X=',X,'   Y=',Y);
         X:=X+1;
      END;
READLN;
END.
```

```
PROGRAM EX10;{Вариант 2}
VAR X,Y:INTEGER;
BEGIN
    X:=-10;
    WHILE X<=10 DO
      BEGIN
           IF X>0 THEN Y:=SQR(X)+1;
                  ELSE IF X=0 THEN Y:=0
                  ELSE Y:=SQR(X)-1;
           WRITELN('X=',X,'Y=',Y);
           X:=X+1;
      END;
 READLN;
END.
```

В программах EX9 и EX10, как вы могли заметить, использовались составные операторы.

```
PROGRAM EX11;{Вариант 3 с использованием оператора цикла с постусловием }
VAR X,Y:INTEGER;
BEGIN
    X:=-10;
```

```
    REPEAT
    IF X>0 THEN Y:=SQR(X)+1;
    IF X=0 THEN Y:=0;
    IF X<0 THEN Y:=SQR(X)-1;
    WRITELN(X=',X,'  Y=',Y);
    X:=X+1;
    UNTIL X>10;
  READLN;
END.
```

Заметим, что в операторе цикла с постусловием, если после слова REPEAT используется не один, а несколько операторов, то не обязательно использовать операторные скобки BEGIN и END, поскольку служебные слова REPEAT и UNTIL сами выполняют роль операторных скобок.

Оператор цикла с параметром

Иногда заранее точно известно, сколько раз должно быть выполнено определенное действие. Для задач такого типа в языке Паскаль, имеется оператор цикла с параметром. Этот оператор имеет вид:

FOR переменная : = выражение 1 TO выражение 2 DO оператор;

где FOR, TO, DO – служебные слова; переменная- переменная целого типа, называемая индексом или параметром цикла.

Выражение 1, выражение 2 – арифметические выражения целого типа, т.е. значения выражений должны быть целыми;

оператор – простой или составной оператор.

Для того, чтобы оператор цикла выполнялся хотя бы один раз, значение выражения 1 должно быть меньше или равно значению выражения 2 (на практике значения выражения 1 всегда меньше значения выражения 2). Оператор работает следующим образом: вначале переменной (параметру цикла) присваивается значение выражения 1, затем сравнивается значение параметра цикла и значение выражения 2. Если параметр цикла меньше значения выражения 2, то выполняется оператор после слова do. Затем параметр цикла увеличивается на 1, после этого вновь сравнивается значение параметра цикла и выражение 2, если параметр цикла меньше, то вновь выполняется оператор после слова do. И так продолжается до тех пор, пока параметр цикла не станет больше выражения 2. Как только это происходит, оператор цикла заканчивается.

Пример:

```
PROGRAM EX12;
VAR X,Y:INTEGER;
BEGIN
    FOR X:=-10 TO 10 DO
    BEGIN
    IF X>0 THEN Y:=SQR(X)+1
           ELSE IF X=0 THEN Y:=0
           ELSE Y:=SQR(X)-1;
        WRITELN('X=',X,'Y=',Y);
     END;
 READLN;
END.
```

Второй вариант оператора цикла с параметром

Оператор цикла с параметром может быть записан и в таком виде:

FOR переменная:= выражение 1 DOWNTO выражение 2 DO оператор;

В этом варианте параметр цикла (переменная) после каждого повторения не увеличивается, а уменьшается на 1. Значение выражения 1 больше или равно (на практике всегда больше) значения выражения 2.

Оператор цикла заканчивается как только параметр цикла станет меньше выражения 2.

Пример:

```
PROGRAM EX13;
VAR X,Y:INTEGER;
BEGIN
    FOR X:=10 DOWNTO -10 DO
    BEGIN
    IF X>0 THEN Y:=SQR(X)+1
           ELSE IF X=0 THEN Y:=0
           ELSE Y:=SQR(X)-1;
        WRITELN(X=',X,'   Y=',Y);
     END;
 READLN;
END.
```

Пример: Вычислить $S=\sum_{x=1}^{10} x$,

```
PROGRAM EX14;{Вариант 1}
VAR X,Y:INTEGER;
BEGIN
   S:=0;
FOR X:=1 TO 10 DO S:=S+X;
    WRITELN('X=',X,'Y=',Y);
     READLN;
END.
```

```
PROGRAM EX15;{Вариант 2}
VAR X,Y:INTEGER;
BEGIN
   X:=1;S:=0;
   WHILE X<=10 DO
   BEGIN
     S:=S+X;
     X:=X+1;
   END;
    WRITELN('X=',X,'Y=',Y);
 READLN;
END.
```

Вычислить $S=\frac{1}{x}+\frac{1}{x^2}+\frac{1}{x^3}+....+\frac{1}{x^n}+...$,

x∈[1.5,2] с шагом h=0.5, точность $\varepsilon=10^{-5}$

```
PROGRAM EX16;
VAR X,S,H,EPS:REAL;
          T:REAL;
BEGIN
      X:=1.5; H:=0.5;
      EPS:=0.00001;
      WHILE X<=2 DO
      BEGIN
       S:=0;T:=1;
       REPEAT
       T:=T*1/X;
       S:=S+T;
       UNTIL T<=EPS;
      WRITELN('X=',X,'S=',S);
      X:=X+H;
      END;
```

```
    READLN;
END.
```

Вычислить sinx используя его разложение в ряд Тейлора:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots.$$

$x \in [0,1]$, h=0.2, точность $\varepsilon = 10^{-3}$

Вывести также "точное" значение, вычисленное с помощью функции SIN(X)

```
PROGRAM EX20;
VAR X,Y,S,H,EPS,T:REAL;
            N:INTEGER;

BEGIN
     X:=0; H:=0.2; EPS:=0.001;
     WHILE X<=1 DO
     BEGIN
      S:=X; T:=X; N:=2;
      REPEAT
      T:=-T*(SQR(X)/N*(N+1));
      S:=S+T;
      N:=N+2;
      UNTIL ABS(T)<EPS;
      Y:=SIN(X);
      WRITELN('X=',X, 'Приближенное значение синуса=',S,
    'Точное значение синуса=',Y);
     X:=X+H;
     END;
  READLN;
END.
```

Символьные и строковые типы данных

Список символов, т.е. список цифр, букв и других знаков содержит таблица ASCII. Номер, который символ имеет в таблице ASCII, называется кодом этого символа. Символ можно представить указав его в кавычках, а можно использовать значок # , за которым следует код символа . Например, буква 'A' , в таблице ASCII имеет номер 65, т е. его код равен 65, тогда можно указать # 65 и это будет означать 'A'. Для обозначения типа "символ" в Паскале используется зарезервированное слово CHAR. Для хранения переменной типа "символ" требуется один байт памяти.

Строкой называется, некоторая последовательность символов: 'AABB', 'ВАСЯ-ЭТО КОТ'.

Строка может состоять из одного символа, а может быть совсем пустой, т. е. состоять из пробелов. Максимальная длина строковой переменной –255 байтов. Тип строковой переменной указывается зарезервированным словом `string`. При этом память для строковой переменной резервируется так, как если бы она содержала строку максимальной длины. Но если заранее точно известна длина строки, например 30 символов, то можно указать `string` [30].

Пример описания строк и символов.

```
VAR SYMBOL:CHAR;
    NAME: STRING[10];
    MESSAGE: STRING;
```

Переменная SYMBOL переменная типа символ (CHAR) – для него отводится 1 байт памяти.

Переменная NAME - типа STRING (строковая переменная), её отводится 10 байт памяти.

MESSAGE - также строковая переменная , её отводится 255 байт памяти, поскольку для нее не указана длина.

Какие операции допустимы для строк и символов? Только операция (сложения). При этом в результате получается строка.

Пример:

```
PROGRAM EX21;
VAR SYMBOL:CHAR;
    ANY,BODY: STRING;
BEGIN
    SYMBOL:='A';
    BODY:= 'Ш';
    ANY:='MAMA';
    BODY:=BODY+SYMBOL;
    {теперь BODY='ША'}
    ANY:=ANY+BODY;
    {ANY='MAMAША'}
    WRITELN(ANY);
    BODY:='+ПАПАША=ПРЕДКИ';
    ANY:=ANY+BODY;
    WRITELN(ANY);
 READLN;
END.
```

Пример:

```
PROGRAM EX22;
VAR ANSW,PASSW: STRING[4];
BEGIN
      PASSW:='BOSS';
      REPEAT
      WRITELN('Введите пароль');
      READLLN(ANSW);
      IF PASSW<> ANSW THEN
           WRITELN('Вы не пользователь');
      ELSE
           WRITELN('Здравствуйте, мой друг!');
      UNTIL ANSW = PASSW;
      READLN;
END.
```

Строковые процедуры и функции

Функция Concat- выполняет конкатенацию последовательности строк, т. е. объединения нескольких строк.

Описание: `Concat(S1[,S2,…,Sn]:string):string;`

Имя функции Concat

S1,S2,…,Sn – формальные параметры, имеют тип string. Квадратные скобки указывают, что параметры S2,…,SN могут отсутствовать. Результат функции строка символов, т.е. имеет тип также string.

Пример:

```
PROGRAM EX23;
VAR A,C: STRING;
BEGIN
       A:='КОЛЯ+ ОЛЯ';
      C:='РАВНЯЕТСЯ ЛЮБОВЬ';
      WRITELN(CONCAT(A,C)); READLN;
END.
```

Функция COPY –возвращает подстроку строки.

Описание:

`COPY(S:STRING,INDEX,COUNT:INTEGER):STRING;`

S - исходная строка,

INDEX - номер символа, начиная с которого выделяется подстрока из S,

COUNT - количество символов в подстроке.

Пример:

```
PROGRAM EX24;
VAR A,B: STRING;
BEGIN
      A:='head&shoulders';
      B:=COPY(A,6,9);
      WRITELN(B);READLN;
END.
```

Процедура DELETE- удаляет из строки подстроку

Описание :

DELETE(S:STRING, INDEX:INTEGER,COUNT:INTEGER);

S – исходная строка, INDEX – номер символа, начиная с которого удаляется подстрока из S, COUNT- количество символов в подстроке.

Пример:

```
PROGRAM EX25;
VAR A: STRING;
BEGIN
      A:='123XYZ';
      DELETE(A,4,3);
      WRITELN(A); READLN;
END.
```

Процедура INSERT – добавляет подстроку в строку .

Описание:

INSERT(S1:STRING,S2:STRING,INDEX:INTEGER);

S2 - исходная строка

S1 - добавляемая строка

INDEX - номер символа строки S2, начиная с которого добавляется подстрока S1.

Пример:

```
PROGRAM EX26;
VAR S: STRING;
BEGIN
      S:='123678';
      INSERT('45',S,4);
      WRITELN(S);
      READLN;
END.
```

Функция LENGTH - возвращает динамическую длину строки.

Описание :

```
LENGTH(S:STRING):INTEGER;
```

Пример:

```
PROGRAM EX27;
VAR S: STRING;
    I: INTEGER;
BEGIN
      S:='ABCDE';
      I:= LENGTH(S);
      WRITELN(S,'ДЛИНА СТРОКИ=',I);
      S:=''; I:=LENGTH(S);
      WRITELN(S,'ТЕПЕРЬ ДЛИНА СТРОКИ =',I);
      READLN;
END.
```

Функция POS- производит поиск подстроки в строке.

Описание :

```
POS(Substr, S:ISTRING):BYTE
```

Substr –подстрока, поиск которой производится в строке S. Функция POS возвращает номер символа, начиная с которого подстрока Substr найдена в S. Если такой подстроки нет, то возвращается 0.

Пример:

```
PROGRAM EX28;
VAR S: STRING;
    I: INTEGER;
BEGIN
      S:='МАМАША';
      I:=POS('ША',S);
      WRITELN(S,  'Номер символа, с которого начинается
подстрока ША=',I);
      READLN;
END.
```

Пример: Дана строка символов длиной 30. Определить, входит ли в эту строку подстрока 'ША'. Строку ввести с клавиатуры.

```
PROGRAM EX29;
VAR S: STRING[30]; Substr:STRING[2];
    I: INTEGER;
BEGIN
      WRITELN ('Введите строку символов') ;
      READLN(S); Substr:='ША';
```

```
      I:=POS(Substr,S);
      IF I=0 THEN WRITELN('В данной строке', S,'подстрока',
           Substr,'отсутствует')
      ELSE
      WRITELN('подстрока',Substr,'входит в данную
   строку начиная с номера символа ',I);
  READLN;
END.
```

Процедура `Str` – преобразует численное значение в его строковое представление.

Описание :

`Srt(X[:width[:decimals]],s:string);`

`X-` числовая переменная (типа `integer` или `real`)

`width-` длина строки `S;decimals-` количество знаков после запятой.

Пример:

```
PROGRAM EX30;
VAR S: STRING; X:REAL;
BEGIN
X:=123.565;
str(X:6:2,S);
{теперь S='123.56'}
      WRITELN(S);
READLN;
END.
```

Процедура `Val` преобразует строковое значение в его численное представление.

Описание :

`VAL(S:string,V,code:integer);`

`S-` исходная строка символов

`V-` числовая переменная – результат преобразования `S`

`code-` целая переменная

Если невозможно преобразовать S в число, то после выполнения процедуры `Val` `code≠0`, если преобразование произошло удачно, то `code=0`.

Пример:

```
PROGRAM EX31;
VAR S: STRING;
    X:REAL;
    C :INTEGER;
BEGIN
     S:='234.12';
     VAL(S,X,C);
     WRITELN(X);
     READLN;
END.
```

Массивы

В некоторых случаях приходится сталкиваться с ситуацией, когда должно использоваться относительно много переменных одного типа.

Представим себе программу, которая позволяет вычислить определенные метеорологические данные и в которой всегда будет не меньше 365 переменных. Таким образом можно присваивать отдельным переменным метео-данные по отдельным дням всего года.

В принципе сейчас нет никакой проблемы, чтобы объявить тип REAL у 365 переменных, используя запись.

```
VAR DAY1, DAY2, DAY3 и т.д. DAY365:REAL;
```

Однако в Паскале нельзя писать и "т.д.". Нужно перечислить все переменные. Для этого нужна по крайней мере страница. Если теперь нужно подсчитать среднее арифметическое на один день, то нужно записать:

```
SRED:=(DAY1+DAY2+и т.д.DAY365)
```

и опять как быть с "и т.д."?

Для таких случаев Паскаль предоставляет возможность введения большого числа переменных одного и того же типа, используя простые выражения.

Эта возможность в Паскале реализуется с помощью так называемых массивов. Используется служебное слово ARRAY, которая указывает на ряд переменных одного и того же типа и имеющих одно имя.

Например :

```
VAR DAY:ARRAY[1..365] OF REAL;
```

С помощью этого описания определяется массив DAY состоящий из 365 элементов (переменных), т.е. каждый элемент массива может использоваться как отдельная переменная. Можно записать:

```
DAY[1]:=1.25;
DAY[2]:=0.34;
```

Другими словами, каждый элемент массива определяется именем массива и номером элемента в квадратных скобках. Говорят еще индекс

массива. Причем в качестве индекса можно использовать арифметическое выражение целого типа.

В описании массива после имени в квадратных скобках указывается минимальное значение индекса. В нашем случае массив DAY состоит из 365 элементов, нумерация идет от 1 до 365 т.е. шаг нумерации по умолчанию всегда равен 1.

При использовании массивов запись программы становится намного короче и наглядней.

Пусть например, в начале все элементы массива DAY должны быть приравнены 0.

Вместо того, чтобы писать

```
DAY[1]:=0;
DAY[2]:=0;
```

и т.д.

```
DAY[365]:=0;
```

мы можем поступить следующим образом:

```
VAR DAY:ARRAY[1..365] OF REAL;
      K : INTEGER;
 BEGIN
      FOR K:=1 TO 365 DO
           DAY [K]:=0;
```

Пример. Пусть имеются 2 массива типа STRING. В одном массиве содержатся фамилии людей, а в другом номера их домашних телефонов. Написать программу, которая в цикле по номеру телефона выводит на экран фамилию этого абонента.

```
PROGRAM EX32;
VAR NAME:ARRAY[1..50]OF STRING[30];
    TEL:ARRAY[1..50]OF STRING[7];
    K :INTEGER;PHONE:STRING[7];
BEGIN
  WRITELN('Введите фамилию и номер телефона');
  FOR K:=1 TO 50 DO
     READLN(NAME[K],TEL[K]);
     PHONE:=' ';
     WHILE PHONE<>'*******' DO
    BEGIN WRITELN('Введите номер телефона');
          READLN(PHONE);
          K:=1;
          WHILE K<= 50 DO
          BEGIN
            IF TEL[K]=PHONE THEN
            WRITELN('Фамилия этого абонента', NAME[K]);
              ELSE K:=K+1;
          END;
```

```
      END;READLN;
  END.
```

Недостаток этой программы в том, что если указанный номер телефона в массиве не существует, то никаких сообщений на этот счет не выводится на экран. Кроме того, после того как найден абонент, например для k=2, программа ещё 48 раз прокрутит цикл. Как модифицировать программу?

```
PROGRAM EX33;
VAR NAME:ARRAY[1..50]OF STRING[30];
    TEL:ARRAY[1..50]OF STRING[7];
    K :INTEGER;PHONE:STRING[7];
    FLAG : BOOLEAN;
BEGIN
  WRITELN('Введите фамилию и номер телефона');
  FOR K:=1 TO 50 DO
    READLN(NAME[K],TEL[K]);
    PHONE:=' ';
    WHILE PHONE<>'*******' DO
    BEGIN
     WRITELN('Введите номер телефона');
     READLN(PHONE);
     K:=1;
     WHILE K<= 50 DO
     BEGIN
       IF TEL[K]=PHONE THEN
         BEGIN
           FLAG:=TRUE;K:=51;
           WRITELN('Фамилия этого абонента',NAME[K]);
         END
           ELSE BEGIN  FLAG:=FALSE;K:=K+1;END;
     END;
     IF FLAG:=FALSE AND PHONE<>'*******' THEN
      WRITELN('Абонента с таким номером телефона нет в
      справочнике');
    END; READLN;
END.
```

Модули в Турбо Паскале

Модули в Турбо Паскале содержат ряд полезных процедур, функций и констант, которые дают возможность использовать почти всю мощь компьютеров. Существуют следующие модули в Турбо Паскале:

SYSTEM – содержит стандартные функции и процедуры "классического" Паскаля, например, вычисления функции COS, SIN,

LN, EXP и др. Модуль SYSTEM автоматически доступен всем программам. Фактически мы широко пользовались этим модулем.

DOS – содержит процедуры и функции, позволяющие использовать средства операционной системы MS-DOS.

CRT – набор процедур для работы с экраном, клавиатурой.

GRAPH – содержит обширный набор программ, который позволяет использовать графические возможности компьютера.

Для использования модуля достаточно в программе после строки заголовка вставить строку.

USES имя модуля;

Если используется несколько модулей то пишется так :

USES имя модуля 1 , имя модуля 2, ….., имя модуля N;

Модуль CRT

Модуль CRT, как уже говорилось, содержит ряд процедур и функции для работы с экраном в текстовом режиме, клавиатурой и динамиком. Рассмотрим экран компьютера в текстовом режиме. Обычно на нем размещается 25 строк текста и 80 символов в каждой строке. Для того, чтобы полностью владеть экраном, нужно знать координаты курсора, менять цвет символов, текста и фона. Для этого существуют следующие процедуры:

- GOTOXY(I,J)-позиционирует курсор в I строку и J столбец экрана.
- WRITE(S)- выводит строку S начиная с текущей позиции курсора.
- процедура Textcolor(COLOR)- устанавливает цвет выводимого текста.

Существуют следующие константы цветов:

Black- черный;
Blue-синий;
Green-зеленый;
Cyan-бирюзовый;
Red-красный;
Magenta- малиновый;
Brown-коричневый;
LightGray-светло- серый;
DarkGray- темно-серый;
LightBlue- ярко-голубой;
LightGreen- ярко- зеленый;
LightCyan-ярко-бирюзовый;
LightRed- ярко- красный;
LightMagenta- ярко- малиновый;
Yellow-желтый;
White-белый;

Blink- мерцание (мигание);

Для вывода на экран мигающего текста надо использовать процедуру TEXTCOLOR(COLOR+BLINK) .

Например TEXTCOLOR(BLUE+BLINK);

WRITELN('Это мигающий текст ')

- Процедура TEXTBACKGROUND(COLOR) - устанавливает цвет фона.
- Процедура CLRSCR-очищает экран, одновременно окрашивая его в цвет установленного фона.
- Процедура WINDOW(X1,Y1,X2,Y2)- определяет на экране окно где X1,Y1- координаты левого верхнего угла , а X2,Y2- координаты правого нижнего угла окна

Функция KEYPRESSED - возвращает значение TRUE, если клавиша нажата и FALSE – в противном случае.

Функция READKEY – считывает символ с клавиатуры.

Пример: Написать программу, которая очищает экран и устанавливает синий фон. Затем рисует окно красного цвета и выводит текст белыми буквами.

```
PROGRAM EX33;
USES CRT;
BEGIN
      TEXTBACKGROUND(BLUE);
      CLRSCR;TEXTBACKGROUND(RED);
      WINDOW(5,5,15,65);
      CLRSCR;GOTOXY(5,6);TEXTCOLOR(WHITE);
      WRITE('ПРИВЕТ, ВАСЯ!');
      READLN;
END.
```

Пример. В некой компании по продаже компьютеров имеется 20 менеджеров, фамилии которых, хранятся в массиве NAME. В другом массиве имеются данные о количестве проданных компьютеров каждым менеджером за месяц. Написать программу, которая выводит таблицу, в которой следующие графы: фамилия, количество проданных компьютеров, сумма выручки, сумма премии. Итоговая строка должна содержать общее количество проданных компанией компьютеров за месяц, общую сумму выручки, сумму премиальных. Стоимость компьютера принять равной 1000$, размер премии за каждый проданный компьютер 25$

```
PROGRAM EX34;
USES CRT;
VAR NAME:ARRAY[1..20] OF STRING[30];
     KOL: ARRAY[1..20] OF INTEGER;
     SUM,COST,PREM,I,K:INTEGER;
```

```
      PROGPREM,SUM1:INTEGER;
BEGIN
      WRITELN('Введите фамилию и количество реализованных
           компьютеров');
      FOR K:=1 TO 20 DO
      READLN(NAME[K],KOL[K]);
      TEXTBACKGROUND(BLUE);CLRSCR;
      TEXTCOLOR(WHITE);GOTOXY(0,15);
      WRITE('СВЕДЕНИЯ');
      WRITE('О РЕАЛИЗАЦИИ КОМПЬЮТЕРОВ');
      TEXTBACKGROUND(RED);GOTOXY(1,12);
      WRITE('За ЯНВАРЬ 2002 г.');
      TEXTBACKGROUND(GREEN);
      WINDOW(2,10,24,70);TEXTCOLOR(YELLOW);
      GOTOXY(2,10);
      WRITE('ФАМИЛИЯ','КОЛИЧЕСТВО','СУММА ВЫРУЧКИ',
      'СУММА ПРЕМИИ');
      TEXTCOLOR(BLACK);GOTOXY(3,10);
      COST:=1000;PROGPREM:= 25;
      FOR K:=1 TO 20 DO
      BEGIN GOTOXY(K+2,10);
            SUM:=COST*KOL[K];
            PREM:=PROGPREM* KOL[K];
            WRITE(NAME[K],KOL[K],SUM,PREM);
      END;
      SUM:=0;SUM1:=0;I:=0;
      FOR K:=1 TO 20 DO
      I:=I+KOL[K];
      SUM:=I*COST;{СУММА ВЫРУЧКИ}
      PREM:=I*PROGPREM;
      GOTOXY(23,10);
      TEXTCOLOR(YELLOW);
      WRITE('ИТОГО');
      WRITE(I,SUM,PREM);
      READLN;
END.
```

Создание и обработка последовательных файлов

Файл – некоторый участок на диске, где хранятся данные. Каждому файлу присваивается уникальное имя, благодаря которому становится возможным обращаться к различным данным на диске. Файлы состоят из отдельных записей – совокупности данных, имеющих свои имена и тип, объединенных одним именем. Запись - комбинированный тип данных.

Каждая запись состоит из отдельных полей, которые могут иметь разные типы.

Общий вид объявления записи:

```
TYPE
  <имя записи>=RECORD
  <имя1>:<тип1>;
  <имя2>:<тип2>;
   ..............
  <имяN>,<имяN+1>:тип;
END;
VAR <имя переменной>:<имя записи>;
```

Пример. Пусть имеется запись следующей структуры:

Фамилия	Имя	Отчество	Группа	Год рождения

```
TYPE
  STUD=RECORD
  family,name,otch,group:string[20];
  birthday:integer;
END;
VAR   A,B:STUD;
```

Обращение к элементам записи выполняется через уточненное (составное) имя.

Пример:

```
A.family:='Иванов'; B.group:='ПОВТАС-1/02';
```

Чтобы не удлинять слишком текст программы, в Паскале предусмотрен оператор присоединения WITH. Это дает возможность определить кусок программы, внутри которого можно просто указывать требуемые поля записи, не указывая каждый раз при этом имя самой записи.

Синтаксис оператора:

```
WITH <список имен записей> DO
begin
 ... {в этом блоке при обращении к полям записи не обязательно каждый раз указывать имя записи}
   ...
end;
```

Пример:

```
TYPE
  STUD=RECORD
```

```
    family,name,otch,group:string[20];
END;
VAR   A,B:STUD;
begin
  with B do
    begin
      family:='Иванов';
      group:='ПОВТАС-1/02';
    end;
end.
```

Файловые типы

В программе для работы с файлами вводятся файловые переменные. Переменные файловых типов необходимы тем программам, которым требуется читать данные с диска или записывать данные на диск. Для описания таких переменных используется зарезервированное слово FILE. Синтаксис описания файловой переменной:

VAR <имя файловой переменной>: FILE of <тип компонентов>;

Пример.

```
VAR NAME: FILE OF STRING[30];
    KOL: FILE OF INTEGER;
```

Файл NAME состоит из последовательности строк длиной 30 символов, файл KOL из целых чисел.

При работе с файлами используются следующие процедуры:

- Для связи файловой переменной с файлом во внешней памяти используется процедура:

```
assign(f,fname);
```

где f - файловая переменная, fname - переменная типа string, содержащая имя файла. Процедура ASSIGN – присваивает имя внешнего файла на диске файловой переменной.

Общий вид имени файла:

<диск>:\<имя каталога>\<имя подкаталога>\...\<имя файла>

Если опущены <диск>,<имя подкаталога>,то принимается текущий каталог и текущее логическое устройство. Только после выполнения ASSIGN можно обращаться к другим процедурам по обработке файлов.

```
ASSIGN(NAME,'NAME.DAT')
ASSIGN(KOL,'C:\TP\WORK\KOL.DAT')
```

- Процедура REWRITE – создает и открывает новый файл. Описание:

REWRITE(f)

f- является файловой переменной. Перед использованием REWRITE переменная f должна быть связана с дисковым файлом с помощью процедуры ASSIGN. Если файл с таким именем уже существует, то он удаляется, а на его месте создается новый пустой файл.

- Процедура RESET- открывает существующий файл для чтения или записи. Описание:

RESET(f) – где f-файловая переменная. Перед использованием RESET переменная f должна быть связана с файлом на диске с помощью процедуры ASSIGN.

- Процедура READ(f,v) - считывает в переменную v очередной элемент файла. Тип переменной v должен соответствовать файловой переменной f.
- Процедура WRITE(f,v)- записывает переменную v в файл. f- файловая переменная, v- переменная того, же типа, что и элемент файла f.
- Процедура CLOSE(f) - закрывает открытый файл.
- Функция EOF в случае, если достигнут конец файла возвращает значение TRUE и FALSE, если конец файла не достигнут.
- rename(f,newname); - переименовать файл; newname - переменная типа string, содержащая новое имя файла;
- erase(F); - удалить файл.

Рассмотрим снова программу Ex34(стр. 175). С этой программой, было не удобно работать, т.к. постоянная информация (фамилии, количество проданных компьютеров) не сохранялись и при каждом запуске программы приходилось заново их вводить.

Проблема решается, если постоянную информацию сохранить на диске.

Напишем программу создания файлов на диске.

```
PROGRAM CREATE;
USES CRT;
VAR NAME: FILE OF STRING[30];
    KOL: FILE OF INTEGER;
BEGIN
    ASSIGN(NAME,'NAME.DAT');
    ASSIGN(KOL,'KOL.DAT');
    REWRITE(NAME);REWRITE(KOL);
    WRITELN('ФАЙЛЫ НА ДИСКЕ СОЗДАНЫ');
    READLN;CLOSE(NAME);CLOSE(KOL);
END;
```

Напишем программу записи информации в созданные файлы.

```
PROGRAM INPUT;
USES CRT;
VAR NAME: FILE OF STRING[30];
    KOL: FILE OF INTEGER;
    COMP,K:INTEGER; FAM:STRING[30];
BEGIN
    ASSIGN(NAME,'NAME.DAT');
    ASSIGN(KOL,'KOL.DAT');
    RESET(NAME);RESET(KOL);
    WRITELN('Введите фамилию и количество проданных компьютеров
    ');
     FOR K:=1 TO 20 DO
        BEGIN READLN(FAM,COMP);
              WRITE(NAME,FAM);
              WRITE(KOL,COMP);
        END;
    WRITELN('Информация на диск записана');
    CLOSE(NAME);CLOSE(KOL);
    READLN;
END;
```

Напишем главную программу

```
PROGRAM MAIN;
USES CRT;
VAR NAME: FILE OF STRING[30];
    KOL: FILE OF INTEGER;
    FAM: STRING[30];
    COMP,COST,PREM,SUM,SUM1,SUMC,SUMV,SUMP:INTEGER;
    FAM:STRING[30];
BEGIN
    ASSIGN(NAME,'NAME.DAT');
    ASSIGN(KOL,'KOL.DAT');
    RESET(NAME);RESET(KOL);
    COST:=1000;PREM:=25;
    SUMC:=0;SUMV:=0;SUMP:=0;
    WRITELN('Фамилия количество сумма выручки сумма премии');
    WHILE NOT EOF(NAME) DO
      BEGIN READ(NAME,FAM);
            READ(KOL,COMP);
            SUM:=COMP*COST;
            SUM1:=COMP*PREM;
            SUMC:=SUMC+COMP;
            SUMV:=SUMV+SUM;
```

```
            SUMP:=SUMP+SUM1;
            WRITELN(FAM,COMP,SUM,SUM1);
    END;
  WRITELN('ИТОГО:',SUMC,SUMV,SUMP);
  READLN;
  END.
```

В этом примере мы создали два файла, каждая запись которых состоят только из одного поля.

Файл Name

Иванов
Петров
.....
Сидоров
Шанкар

Файл Kol

2
10
15
0

На практике, конечно, записи имеют более сложную структуру, т.е. состоят из нескольких полей, имеющих разные типы. Создадим один файл с именем Managers, имеющий следующую структуру записи:

Name	Kol
Иванов	2
Петров	10
.....	
Сидоров	15
Шанкар	0

```
Program Create;
Uses CRT;
Type
   New_Type=record
   Name:string[30];
   Kol:integer;
end;
New_File=file of New_Type;
var
  Fam&comp:New_Type;
  Managers:New_File;
  k:integer;
begin
  assign(Managers, 'Managers');
  rewrite(Managers);
  writeln('Введите фамилию менеджера и количество
```

```
проданных им компьютеров');
   with Fam&comp do
   begin
      for k:=1 to 20 do
      begin
         read(name, kol);
         write(Managers,Fam&comp);
      end;
   end;
end.
```

В этом примере количество менеджеров фиксировано и равно 20. Напишем программу, в которой количество менеджеров вводится с клавиатуры.

```
Program Create2;
Uses CRT;
Type
   New_Type=record
   Name:string[30];
   Kol:integer;
end;
New_File=file of New_Type;
var
  Fam&comp:New_Type;
  Managers:New_File; k,n:integer;
begin
  with Fam&comp do
  begin
     writeln('Укажите количество менеджеров');
     readln(n);
     assign(Managers, 'Managers');
     rewrite(Managers);
     writeln('Введите фамилию менеджера и количество
проданных им компьютеров');
     for k:=1 to n do
     begin
        read(name, kol);
        write(Managers,Fam&comp);
     end;
  end;
end.
```

Общая структура Паскаль – программы

Общая структура Паскаль – программы имеет вид:

```
Program < Имя программы >;
Label
   < Метки >;
Const
   < Константы >;
Type
   < Новые типы данных >;
Var
   < Описание переменных >;
   < Процедуры >;
   < Функции >;
Begin
   < Тело главной программы >;
End.
```

С основными элементами этой структуры мы уже знакомы. Нами остались не рассмотренными <Метки>, <Процедуры>, <Функции>.

Метки нужны для организации перехода на какой – либо оператор. Для этого оператор помечается меткой. Оператор перехода имеет вид:

```
GoTo <Метка>;
```

и позволяет передать управление оператору с меткой <Метка>.

Автор не советует читателю привыкать писать программы с метками. Программа, содержащая метки обычно плохо читается и, как правило, является не структурированной. Вполне можно писать программы вообще не используя метки. Итак, забыли про метки и поехали дальше!

Паскаль позволяет часто повторяющиеся части большой программы оформлять в виде отдельной программы, называемой процедурой или функцией. Каждая процедура или функция должна иметь имя. Текст процедуры или функции записывается в разделе описаний, после описания переменных. В дальнейшем для того, чтобы использовать эту процедуру или функцию достаточно указать ее имя. В некоторых случаях процедура (функция) использует некоторые значения, которые должны передаваться из главной программы или из других процедур (функций). Эти значения называются параметрами. Параметры указываются в заголовке процедуры (функции) в скобках. Если их несколько, то они разделяются запятыми.

Отличие функции от процедуры в том, что функция обязательно должна возвратить некоторое вычисленное значение вызывающей программе.

Структура процедуры

```
procedure <имя процедуры>[(параметры)];
Label
   < Метки >;
Const
   < Константы >;
Type
   < Новые типы данных >;
Var
   < Описание локальных переменных >;
   < Описание внутренних процедур >;
   < Описание внутренних функций >;
Begin
   < Тело процедуры >;
End;
```

Параметры, указанные в заголовке называются формальными параметрами. Для вызова процедуры необходимо просто указать ее имя и параметры (если они имеются). Параметры, указанные при вызове процедуры называются фактическими параметрами.

Структура функции

```
function <имя функции>[(параметры)]:тип функции;
Label
   < Метки >;
Const
   < Константы >;
Type
   < Новые типы данных >;
Var
   < Описание локальных переменных >;
   < Описание внутренних процедур >;
   < Описание внутренних функций >;
Begin
   < Тело функции >;
End;
```

В заголовке функции указывается ее имя, параметры (если они есть) и тип значения, которую функция должна возвратить. Для вызова функции необходимо указать ее имя и фактические параметры (при их наличии). Причем имя функции можно указывать в выражениях!

В теле функции должен присутствовать хотя бы один оператор, который присваивает имени функции некоторое значение. В противном случае значение функции остается неопределенным, что может привести к

непредсказуемым последствиям. При вычислении выражений, имя функции замещается вычисленным значением.

Пример использования функции.

Вычислить значение $y = \sin x$ путем разложения $\sin x$ в ряд Тейлора с точностью $\varepsilon = 10^{-3}$, $x \in [0,1]$, $\Delta x = 0.1$ Вычисление оформить в виде функции. Вычисленное значение $\sin x$ для каждого x сравнить с "точным" значением путем применения стандартной функции Паскаля SIN(X).

Напомню, что мы уже писали эту программу (программа № 20, стр. 165), но без применения функций. Разложение $\sin x$ в ряд Тейлора имеет вид:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots.$$

```
Program Ex;
var x,y,dx:real;
function No_standard_sin (x: real): real;
var eps,s,t: real;
    n: integer;
begin
  s:=x; t:=x; n:= 2; eps:= 0.001;
  repeat
    t:=-t*(SQR(x)/n*(n+1));
    s:=s+t;
    n:=n+2;
  until ABS(t)< eps;
  No_standard_sin:=s;
end;
begin
  x:=0; dx:=0.1;
  writeln('Значение синуса, которое вычислено с
помощью');
  writeln('моей функции','стандартной функции');
  while x<=1 do
  begin
    y:=No_standard_sin(x);
    writeln(y,'                                    ',SIN(x));
    x:=x+dx;
  end;
readln
end.
```

Пример. Рассмотрим пример разработки достаточно большой и сложной программы, реализующую методы работы с последовательными файлами, в которой используются записи сложной структуры, процедуры и

функции. Структура записи:

Фамилия	Группа	Предмет	Оценка

```
program pr;
uses Crt;
```
{ Предложение uses Crt необходимо для подключения процедур: clrscr - очистка экрана;
gotoxy - установка курсора в точку (x,y),
1<=X<=80 и 1<=Y<=25
Точка отсчета (1,1) - левый верхний угол экрана }

```
TYPE str=string[16];
     rec=RECORD
     fio,sub:str;
     gr,m:integer;
END;
     ft=file of rec;
VAR
     symd:str;  ans:char;
     f,v:ft; z:rec; i,ig,dig:integer;
     fname:string[8];
```

{ f,v - файловые переменные, где f - имя основного файла; v - имя вспомогательного файла; fname - строковая переменная, предназначенная для ввода имени внешнего файла;
z - переменная комбинированного типа (типа RECORD-END);
ans, ig - переменные, предназначенные для выбора режима;
i, symd, dig - вспомогательные переменные; }

```
{ ========== Ввод данных ========== }
procedure indata;
BEGIN
    writeln('  Фамилия    ');
    readln(z.fio);
    writeln('  Предмет    ');
    readln(z.sub);
    writeln('  Оценка     ');
    readln(z.m);
    writeln('  Группа     ');
    readln(z.gr);
 end;
{ ========== Создание файла ========== }
procedure create;
BEGIN
  rewrite(f);
```

```
    writeln('  Вводите информацию ');
    repeat
    indata;
    write(f,z);
    writeln('  Продолжить, ответ - да/нет(Y/N) ');
    readln(ans);
    until (ans='N') or (ans='n');
  END;
 { ================ Вывод содержимого файла ================ }
  procedure out;
  var j:integer; p:text;
  Begin
    reset(f);  clrscr;
     { j - счетчик записей в файле }
    writeln(' Вывод информации на печать/экран(Y/N)');
    write(' Ваш ответ='); readln(ans);
    gotoxy(1,5); j:=0;
    if (ans='Y') or (ans='y') then
     assign(p,'prn')
    else
     assign(p,'con');
     rewrite(p);
     writeln(p,'    *      фамилия     *        группа      *
предмет   *    оценка   *');
     writeln(p,'================================');
     while not eof(f) do
     begin
     read(f,z);  j:=j+1;
     gotoxy(1,6+j);writeln(p,z.fio:16,z.gr:15,'
',z.sub:17,z.m:10);
     end;
     writeln(p,'================================');
     writeln(p,' Число студентов=',j:2);
     writeln('Для продолжения нажмите клавишу Enter ');
     readln;
   end;
    { ========= счетчик записей ========== }
  function krecf: integer;
  var j:integer;
    begin
       reset(f); j:=0;
       while not eof(F) do
       begin
         read(f,z); j:=j+1;
       end;
```

```
      krecf:=j; close(f);
    end;
  { ========= Поиск записей по заданным полям ========== }
  procedure select;
  Begin
    Repeat
    reset(f);
    clrscr;
    gotoxy(10,10); write  ('Выбор информации по:');
    gotoxy(10,11); write  ('  группе              - 1');
    gotoxy(10,12); write  ('  предмету            - 2');
    gotoxy(10,13); write  ('  оценке              - 3');
    gotoxy(10,14); writeln('  отказ от выбора     - 4');
    readln(i);
     clrscr;
     case i of
  1: begin write(' Группа -'); readln(dig);
     writeln(' Сведения по группе ',dig:5) end;
  2: begin write(' Предмет -'); readln(symd);
     writeln(' Сведения по предмету ',symd:15) end;
  3: begin write(' Оценка ='); readln(dig);
     writeln(' Сведения по оценке  ',dig:1) end;
    end; { end of case }
  while not eof(f) do
    begin
      read(f,z);
    case i of
     1:    if    z.gr=dig    then    writeln(z.fio:15,'
',z.sub:15,' ',z.m:1);
     2:    if    z.sub=symd    then    writeln(z.fio:15,'
',z.gr:15,' ',z.m:1);
     3: if z.m=dig then writeln(z.fio:15,' ',z.sub:15,'
',z.gr:5);
    end; { end of case }
  end; { end of while }
  gotoxy(5,24);   writeln('   Для   продолжения   нажмите
клавишу Enter ');
  readln;
  until I=4;
  end;
  { ======== Восстановление файла под основное имя f ========= }
  procedure restorefile;
  BEGIN
       close(f); close(v); erase(f);
       rewrite(f); reset(v);
```

```
      while not eof(v) do
            begin read(v,z); write(f,z) end;
      close(f); close(v); erase(v);
    { удален вспомогательный файл v под внешним именем s.dat }
  END;
{ ============== Добавление записей в файл ============== }
  procedure add;
  BEGIN
       assign(v,'s.dat'); rewrite(v);
   { "s.dat" - имя вспомогательного файла }
      reset(f);
    { копирование содержимого файла f в файл v }
      while not eof(f) do
      begin read(f,z); write(v,z) end;
       writeln('  Вводите информацию ');
    { записи добавляются в конце файла }
      repeat
      indata; write(v,z);
      writeln('  Продолжить, ответ - да/нет(Y/N) ');
      readln(ans);
      until (ans='N') or (ans='n');
   { добавлены новые записи в файл v }
      restorefile;
  END;
{ ============== Удаление записей из файла ============== }
  procedure del;
  BEGIN
    assign(v,'s.dat'); rewrite(v); reset(f);
    clrscr;
    gotoxy(10,10); writeln('Удаление информации по:');
    gotoxy(10,11); writeln('  группе              - 1');
    gotoxy(10,12); writeln('  фамилии             - 2');
    gotoxy(10,13); writeln('  предмету            - 3');
    gotoxy(10,14); writeln('  оценке              - 4');
    gotoxy(10,15); writeln('  отказ от выбора     - 5');
    gotoxy(10,16);   write('  выбор режима =');
    readln(i);
    case i of
     1: begin write(' Группа  - '); readln(dig); end;
     2: begin write(' Фамилия - '); readln(symd); end;
     3: begin write(' Предмет - '); readln(symd); end;
     4: begin write(' Оценка  - '); readln(dig);   end;
     5: exit; { выход в основную программу }
    end; { end of case }
```

```
    { ========= поиск записи для удаления ========== }
    while not eof(f) do
    begin
      read(f,z);
      case i of
       1: if z.gr<>dig then write(v,z);
       2: if z.fio<>symd then write(v,z);
       3: if z.sub<>symd then write(v,z);
       4: if z.m<>dig then write(v,z);
          else
           begin writeln(' Ошибка при вводе ');
           writeln('Для продолжения нажмите клавишу
Enter');
           readln;
           end;
      end; { end of case }
    end; { end of while }
    restorefile;
  END;
{ ============== корректировка записей в файле ================ }
  procedure vary;
  var r:rec;
  Begin
    reset(f); assign(v,'ss.dat'); rewrite(v);
    clrscr;
    gotoxy(10,9);   writeln('Укажите ключ (поле)для
поиска');
    gotoxy(10,10);writeln('корректируемой записи -
по:');
    gotoxy(10,11); writeln(' группе - 1');
    gotoxy(10,12); writeln(' фамилии - 2');
    gotoxy(10,13); writeln(' предмету - 3');
    gotoxy(10,14); writeln(' оценке - 4');
    gotoxy(10,15); writeln(' отказ от выбора - 5');
    gotoxy(10,16);  write(' выбор режима =');
    readln(i); clrscr;
    gotoxy(10,9); writeln(' Замена информации ');
    case i of   { поиск записи }
1:begin gotoxy(10,10); write('группа='); readln(dig);
        indata
  end;
2:begin gotoxy(10,10); write('фамилия='); readln(symd);
        indata
  end;
3:begin gotoxy(10,10); write('предмет='); readln(symd);
```

```
        indata
  end;
4: begin gotoxy(10,10); write('оценка='); readln(dig);
        indata
  end;
5:exit; { выход в основную программу }
    end; { end of case }
    while not eof(f) do
    begin
      read(f,r);
      case i of
1: begin if dig=r.gr then write(v,z) else write(v,r)
   end;
2: begin if z.fio=r.fio then write(v,z) else write(v,r)
   end;
3: begin if z.sub=r.sub then write(v,z) else write(v,r)
   end;
4: begin if z.m=r.m then write(v,z) else write(v,r)
   end;
      end; { end of case }
    end; { end of while }
    restorefile;
  end;
    { ========== основная программа ==========}
begin
    clrscr;
    write(' введите имя файла не более 8 символов -');
    readln(fname);   { ввод произвольного имени }
    assign(f,fname+'.dat'); { .dat - расширение для
данных }
    repeat
    clrscr;
    { Формирование меню работы с основным файлом f }
gotoxy(10,10); writeln('  ВЫБОР: ');
gotoxy(10,11); writeln('создание файла  - 1');
gotoxy(10,12); writeln('вывод содержимого файла - 2');
gotoxy(10,13); writeln('поиск записей по заданным полям
- 3');
gotoxy(10,14);writeln('добавление записей в файл - 4');
gotoxy(10,15);writeln('удаление записей из файла - 5');
gotoxy(10,16);writeln('корректировка записей в файле -
6');
gotoxy(10,17);writeln('  выход из программы - 7');
    readln(ig);
    case IG of
```

```
  { IG - значение для выбора режима работы с файлом f }
    1: create;
    2: out;
    3: select;
    4: add;
    5: del;
    6: vary;
  end; { end of case }
 until IG=7;
end.
```

Литература

1. Фигурнов В.Э.
   " IBM PC для пользователя", -М., 1995 г.
2. Левин А.
   " Самоучитель работы на компьютере", -М., 1996 г.
3. Брябрин В.М.
   " Программное обеспечение ПЭВМ", -М., 1989 г.
4. Фролов Г.Д., Кузнецов Э.И.
   " Элементы информатики", -М., 1989
5. Дж. Мартин
" Организация баз данных в вычислительных системах", -М., 1976 г.
6. Громов Г.Р.
   " Национальные информационные ресурсы", -М., 1985 г.
7. Глушков В.М.
   "Основы безбумажной технологии", -М., 1982 г.
8. Г. Симпсон
   "Профессиональная работа на персональном компьютере", -М.: "Финансы и статистика", 1988 г.
9. А. Файсман
   "Профессиональное программирование на Турбо Паскале", -М.: "Инфомэкс", 1992 г.

ОГЛАВЛЕНИЕ

Подписано в печать 20.09.2005 г.

Формат: 60x84 1/16 Объем: 8,6 п.л.
Заказ: 30 Тираж: ___ экз.

ОшГУ, Издательский центр «Билим»
г. Ош, ул. Ленина, 331, каб.135., тел.: 7-20-61