50 коп.



МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РСФСР

МОСКОВСКИЙ ОРДЕНА ЛЕНИНА
И ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ
ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ
имени В. И. ЛЕНИНА

МГПИ

# Методические разработки по курсу «ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ»

(машинный вариант)

Разделы: основы алгоритмизации и основы вычислительной техники

Москва 1986

МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РСФСР

МОСКОВСКИЙ ОРДЕНА ЛЕНИНА И ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ
ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ
ИМЕНИ В.И.ЛЕНИНА

# Методические разработки по курсу "ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ"

/машинный вариант/

Раздели: основы алгоритмизации и основы вычислительной техники

Москва - 1986

УДК 378.147

Печатается по постановлению редакционно - издательского совета Московского ордена Ленина и ордена Трудового Красного Знамени государственного педагогического института имени В.И.Ленина

Э.И.Кузнецов. Методические разработки по курсу "Основы информатики и вычислительной техники" /машинный вариант/. Разделы: основы алгоритмизации и основы вычислительной техники. -М.: МГПИ им. В.И.Ленина, 1986, с. 100 с ил.

Методические разработки по разделам: основы алгоритмизации и основы вычислительной техники курса "Основы информатики и вычислительной техники" содержат сведения об алгоритмах, их структурах, средствах записи, структурах данных, а также общие сведения об электронных вычислительных машинах, в том числе, о персональных ЭВМ. Они предназначены для будущих учителей информатики.

Рецензенты: Г.А.Шадрин, кандидат физ. - мат. наук, доцент, заведующий кафедрой математической физики МГПИ им. В.И.Ленина.

П.П.Кольцов, кандидат физ. - мат. наук, заведующий сектором научного совета АН СССР по комплексной проблеме "Кибернетика".







СОДЕРЖАНИЕ

## ПРЕДИСЛОВИЕ

Настоящие методические разработки по курсу "Основы информатики и вычислительной техники" предназначены для будущих учителей информатики общеобразовательной средней школы. Они охватывают два раздела курса: основы алгоритмизации и основы вычислительной техники. Третий раздел – основы программирования излагается в работе [1], являющейся продолжением данной работы.

В настоящее время в условиях безмашинного обучения раздел основы алгоритмизации составляет основу программы курса IX класса, а раздел основы вычислительной техники включен в программу курса X класса. В проекте перспективной программы курса оба эти раздела предполагается включить в программу IX класса[2].

Предлагаемые вниманию читателей материалы были разработаны в течение 1985/86 учебного года при проведении экспериментального курса информатики в общеобразовательной средней школе № 57 г. Москвы в объеме часов трудового обучения. Несмотря на это они вполне могут быть использованы и при проведении обязательного школьного курса информатики в IX и X классах в объеме 102 часов.

Эти материалы дополняют пробное учебное пособие [3] и являются продолжением работ [4] и [5], которые были разработаны в процессе экспериментальной работы в школе № 57 в 1984/85 учебном году. Необходимо отметить, что в работе [4] по вине авторов отсутствует ссылка на работу Годуновой Е.К. и МихайловойМ.А. "ПМК в играх и развлечениях", которая в настоящее время находится в печати в издательском отделе МГПИ имени В.И.Ленина и материалы которой были частично использованы при подготовке темы 21 в работе [4].

Предлагаемые материалы могут быть использованы как в условиях безмашинного обучения, так и при "машинной поддержке" курса школьной информатики.

Москва, 1986г. Э.И.Кузнецов

Содержание курса
"Основы информатики и вычислительной техники"
/машинный вариант/

РАЗДЕЛ I. Основы алгоритмизации

ТЕМА I. АЛГОРИТМЫ
Понятие алгоритма. Средства записи алгоритмов. Структуры алгоритмов.

ТЕМА 2. СТРУКТУРЫ ДАННЫХ
Простые переменные. Массивы. Очереди. Стеки. Строки. Списки. Таблицы.

РАЗДЕЛ 2. Основы вычислительной техники

ТЕМА 3. ОБЩИЕ СВЕДЕНИЯ ОБ ЭВМ
Структура ЭВМ и принципы работы. Развитие структуры ЭВМ. Режимы использования ЭВМ. Персональные ЭВМ.

РАЗДЕЛ 3. Основы программирования

ТЕМА 4. ПЕРСОНАЛЬНАЯ ЭВМ "АГАТ" И ЯЗЫКИ ПРОГРАММИРОВАНИЯ
Основные технические характеристики и принципы работы персональной ЭВМ "Агат". Описание языка "Бейсик - Агат". Описание языка Рапира.

ТЕМА 5. ФАЙЛЫ
Фиксированная и переменная длина элементов. Хранение файлов. Открытие и закрытие файла. Запись и считывание последовательных записей. Запись и считывание записей с устройств с прямым доступом. Методы работы с файлами в системе "Школьница". Языковые средства обработки файлов. Упражнения.



ТЕМА 6. СОРТИРОВКА
Сортировка извлечением. Метод "пузырька". Метод слияния.

ТЕМА 7. ЗАДАЧИ ОБРАБОТКИ ТЕКСТОВОЙ ИНФОРМАЦИИ

ТЕМА 8. ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ ПЕРСОНАЛЬНОЙ ЭВМ "АГАТ"
Управление графикой и диалогом в Рапире. Процедура РЖМ. Графический комплект "Шпага".

ТЕМА 9. ИСПОЛЬЗОВАНИЕ ЭВМ ПРИ РЕШЕНИИ НАУЧНО - ТЕХНИЧЕСКИХ И ЭКОНОМИЧЕСКИХ ЗАДАЧ. ОСНОВНЫЕ ЭТАПЫ РЕШЕНИЯ ЗАДАЧИ НА ЭВМ
Постановка задачи и построение алгоритма. Разработка программы. Трансляция программы. Отладка программы. Эксплуатация программы.

РАЗДЕЛ I. Основы алгоритмизации

ТЕМА I. АЛГОРИТМЫ

## I.I. Понятие алгоритма

Понятие алгоритма относится к числу фундаментальных концепций информатики. Оно возникло задолго до появления электронных вычислительных машин и вошло в число основных понятий математики.

Слово "алгоритм" произошло от имени арабского математика аль-Хорезми (IX в.) и использовалось в математике для обозначения правил выполнения четырех типов арифметических действий - сложения , вычитания , умножения и деления. В настоящее время под алгоритмом в самом общем смысле понимают систему правил , с помощью которых за конечное число шагов можно достичь желаемой цели или получить решение поставленной задачи.

При выяснении понятия алгоритма важное значение играет также понятие исполнителя алгоритма. Алгоритм формулируется в расчете на конкретного исполнителя , который будет этот алгоритм исполнять. Поэтому значение слова "алгоритм" близко по смыслу к значению слов "указание" или "предписание". Иными словами можно сказать , что алгоритм - это понятное и точное предписание (указание) исполнителю совершить определенную последовательность действий для достижения указанной цели или решения поставленной задачи.

Понятно , что сказанное выше не является определением в строгом смысле слова , а лишь отражает интуитивное понимание алгоритма , сложившееся за долгие годы

Алгоритмы применяются в нашей жизни для решения самых разнообразных задач. Например , можно сформулировать алгоритм управления производственным процессом , алгоритм пользования бытовым прибором , алгоритм игры в шахматы , алгоритм поиска пути в лабиринте , алгоритм решения геометрической задачи на построение с помощью циркуля и линейки и т.п.

Алгоритм имеет ряд важных особенностей , выделяющих его среди очень схожих понятий таких , как рецепт , процедура , метод и др. Эти особенности алгоритма лучше всего пояснить на конкретном примере.

Предположим , что необходимо подсчитать число гласных букв в тексте на русском языке , который заканчивается специальным



символом ☒ , больше нигде в тексте не встречающимся. Для решения этой задачи можно предложить такой способ подсчета числа гласных: читать последовательно символ за символом и каждый раз когда очередной символ есть гласная буква русского алфавита , прибавлять к счетчику единицу. Счет начинать с нуля и продолжать до тех пор , пока не будет прочитан символ ☒ , обозначающий конец текста.

Приведенное описание дает лишь идею алгоритма , но чтобы превратить его в понятное и четкое предписание исполнителю , его необходимо уточнить.

Во-первых , отметим , что объектами алгоритма являются символы текста и числа. Над символами текста производятся операции чтения , поиска следующего символа , сравнения символа с символом ☒ конца текста и сравнения символа на совпадение с одной из гласных букв русского алфавита. Над числами производится операция прибавления единицы.

Во-вторых , символы текста читаются последовательно один за другим , начиная с первого символа до символа ☒ включительно. Чтобы не ошибиться при чтении , будем предполагать , что наш абстрактный исполнитель имеет в своем распоряжении указатель , который можно перемещать по тексту и который указывает на очередной читаемый символ. Поэтому поиск следующего символа сводится к перемещению указателя на следующий по порядку символ текста.

В-третьих , счет числа гласных букв начинается с нуля , поэтому до начала работы счетчик числа гласных должен содержать нуль.

Учитывая эти замечания сформулируем алгоритм в виде такой последовательности действий:

1. Записать в счетчик 0.
2. Установить указатель на первый символ текста.
3. Если символ не есть ☒ , то перейти к п.4 , иначе перейти к п.7.
4. Если символ - гласная буква русского алфавита , то увеличить счетчик на единицу.
5. Перевести указатель на следующий символ текста.
6. Перейти к п.3.
7. Взять число , находящееся в счетчике , в качестве ответа. Стоп.

Обратим внимание на следующие основные особенности алгоритма.

I. Дискретность. Алгоритм представлен в виде конечной последовательности шагов. Говорят , что алгоритм имеет дискретную структуру. Следовательно , выполнение алгоритма расчленяется на выполнение его шагов , причем выполнение очередного шага начинается после завершения предыдущего,

2. Конечность. Выполнение алгоритма заканчивается после конечного числа шагов. Заметим , что при выполнении алгоритма некоторые его шаги могут повторяться многократно. В нашем примере многократно повторяются шаги с третьего по шестой. Однако выполнение алгоритма все же закончится за конечное число шагов , так как текст имеет конечную длину , и в нем имеется символ $ . При чтении этого символа будет выполнен седьмой шаг алгоритма , завершающий его исполнение.

В математике однако существуют вычислительные процедуры , имеющие алгоритмический характер , но не обладающие свойством конечности. Например , можно сформулировать процедуру вычисления числа $\pi$ , которая описывала бы процесс получения $\pi$ с любым числом десятичных знаков. Такая процедура описывает бесконечный процесс и никогда не завершится , если не прервать ее искусственно. Подобные процедуры обычно называют вычислительными методами.

3. Понятность. Чтобы алгоритм можно было выполнить , он должен быть понятен исполнителю. Приведенный выше алгоритм подсчета числа гласных букв понятен человеку , знающему русский алфавит. Чтобы этот алгоритм мог быть также выполнен человеком , не знающим русского алфавита , необходимо сообщить такому исполнителю дополнительные сведения , именно , перечислить явно гласные буквы русского алфавита.

Таким образом , при формулировке алгоритма всегда необходимо учитывать возможности и особенности исполнителя , на которого расчитан алгоритм.

4. Определенность. Каждый шаг алгоритма должен быть четко и недвусмысленно определен и не должен оставлять места для произвольной трактовки его исполнителем. При выполнении алгоритма исполнитель должен действовать строго в соответствии с его правилами и у него не должно возникать потребности предпринимать



какие-нибудь самостоятельные действия. Иными словами , алгоритм рассчитан на чисто механическое выполнение.

Это очень важная особенность означает , в частности , что , если один и тот же алгоритм поручить для исполнения разным исполнителям , то они придут к одному и тому же результату. Именно определенность в формулировке алгоритма позволяет поручить его исполнение автомату , не обладающему " здравым смыслом".

В приведенном выше примере правила обладают определенностью, если в качестве исполнителя выступает человек. Для исполнителя-автомата некоторые правила могут оказаться не достаточно точно определенными. Например , действие перемещения указателя на следующий символ определено , если текст расположен на длинной узкой ленте и вытянут в одну строку ( например , напечатан на узкой телетайпной ленте). Если же это печатный текст , разбитый на строки и страницы , как это обычно делается в книге , то перемещения указателя на следующий символ должно быть дополнено указаниями автомату,как поступать , когда закончится очередная строка или очередная страница текста.

Примером неопределенного указания другого рода может служить такое указание:"если x намного больше 1 , то напечатать число 5". Различные кулинарные рецепты , имея много общего с алгоритмами , часто страдают отсутствием определенности , так как используют , например , такие указания: " добавить сахар по вкусу" или " подогреть смесь , слегка помешивая ее ложкой" и т.п.

5. Ввод. Алгоритм имеет некоторое число входных величин (аргументов) , которые задаются до начала работы. Возможность задания различных наборов исходных значений обеспечивает свойство массовости алгоритма , то есть применения его для решения целого класса однотипных задач , различающихся исходными данными. В нашем примере алгоритма входными данными являются символы текста.

6. Вывод. Целью выполнения алгоритма является получение определенного результата (результатов) , имеющего вполне определенное отношение к исходным данным. Можно сказать , что алгоритм перерабатывает исходные данные в результаты. В нашем примере результат - число , обозначающее число гласных букв в исходном тексте.

7. Эффективность. От алгоритма требуется , чтобы каждый его шаг исполнитель мог выполнить точно и за конечное время. Поэтому например , указание "если любое целое число , большее 6 , можно представить в виде суммы трех простых чисел , то напечатать число I , иначе - число 0" не является эффективным , так как нам не известен способ доказательства истинности или ложности утверждения , содержащегося в указании. Поэтому исполнитель не может выполнить этот шаг алгоритма , пока не найдет доказательства истинности или ложности содержащегося в нем утверждения.

Кроме того эффективность означает , что алгоритм может быть выполнен не просто за конечное , а за разумное конечное время , так как использование любого алгоритма на практике включает требование его окончания именно за разумное число шагов.

Выше уже говорилось , что основные особенности алгоритма позволяют использовать в качестве его исполнителя не только человека , но и автомат , лишь бы такой автомат мог воспринять и выполнить указания алгоритма. В качестве таких исполнителей могут использоваться роботы , составляющие основу современных гибких автоматизированных производств , или электронные вычислительные машины (ЭВМ) , осуществляющие автоматическую обработку информации.

Обычно отдельные указания исполнителю , содержащиеся в каждом шаге алгоритма , называют командами. Следовательно , возможность выполнения алгоритма конкретным исполнителем связана с возможностью исполнения им каждой команды алгоритма. Разные исполнители отличаются друг от друга своими возможностями , то есть репертуаром команд , которые они "понимают" и могут исполнять. Совокупность таких команд , которые могут быть выполнены конкретным исполнителем , называется системой команд исполнителя.

Следовательно , алгоритм , рассчитанный на конкретного исполнителя , должен быть сформулирован так , чтобы содержать только те команды , которые входят в систему команд этого исполнителя. В этом смысле можно сформулировать алгоритм для некоторого абстрактного исполнителя с заданной системой команд. Для того же чтобы этот алгоритм мог быть реально исполнен , необходимо реализовать этого абстрактного исполнителя на каком-то конкретном физическом исполнителе , например , на конкретной ЭВМ. В этом случае говорят , что на данной ЭВМ реализован виртуальный ( абстрактный ) исполнитель.



Для разработчика алгоритма , вообще говоря , достаточно знать только систему команд исполнителя , но это вовсе не означает , что ему безразлично , как будет практически выполняться алгоритм. Ведь обычно для нас важно , чтобы задача по разработанному алгоритму решалась как можно быстрее. Вот почему при разработке алгоритмов нас должны интересовать и возможности конкретных физических исполнителей.

Возможности реальных исполнителей алгоритмов находят также свое отражение и в способах записи алгоритмов , к рассмотрению которых мы сейчас переходим.

## I.2. Средства записи алгоритмов.

В информатике сложились вполне определенные традиции в представлении алгоритмов , расчитанных на разных исполнителей. Средства , которые используются для записи алгоритмов , в значительной степени определяются тем , для кого предназначается алгоритм. Если алгоритм предназначен для человека , то его запись может быть не полностью формализована , на первое место здесь выдвигается понятность и наглядность , поэтому для записи таких алгоритмов может использоваться естественный или графический язык , лишь бы запись отражала все основные особенности алгоритма. Для записи алгоритмов , предназначенных для автоматов ( в частности , для ЭВМ) необходима формализация , поэтому в таких случаях применяют специальные формальные языки.

Ниже кратко обсуждаются основные средства , используемые для записи алгоритмов.

### I.2.I. Словесная запись алгоритма

С этой формой записи алгоритма мы уже познакомились в пункте I.I. Обычно так записывают алгоритмы , ориентированные на человека. Команды алгоритма нумеруют , чтобы иметь возможность на них ссылаться. Приведем в качестве еще одного примера словесной формы записи алгоритма классический алгоритм Евклида для нахождения наибольшего общего делителя двух натуральных чисел:

I. Если числа равны , то взять первое число в качестве ответа и закончить выполнение алгоритма , иначе перейти к п.2.
2. Определить большее из двух чисел.
3. Заменить большее число на разность большего и меньшего чисел.
4. Перейти к п.I.

Команды такого алгоритма выполняются в естественной последовательности , если не оговорено противного. Так , после второй команды будет выполняться третья , а вот после четвертой команды необходимо вернуться снова к выполнению первой команды, так как это явно оговорено в четвертой команде.

Команды перехода нарушают естественный порядок выполнения команд алгоритма. В нашем примере команда перехода обеспечивает многократное повторение команд алгоритма. Такое повторение будет происходить до тех пор , пока числа не станут равными.

Проанализировав этот алгоритм , можно видеть , что он обладает всеми перечисленными ранее особенностями алгоритма. В частности , конечность алгоритма Евклида определяется тем , что в процессе его выполнения мы будем получать убывающую последовательность целых положительных чисел , которая должна кончиться.

I.2.2. Блок-схемы

Блок-схемы представляют алгоритм в наглядной графической форме. Команды алгоритма изображаются в виде плоских геометрических фигур , соединенных стрелками , показывающими очередность выполнения команд алгоритма.

На рис.I.I. представлен в виде блок-схемы рассмотренный выше алгоритм Евклида/см. стр. 30/.

Команды обработки информации изображаются в виде прямоугольников , условия , которые приходится проверять в ходе выполнения алгоритма ,- в виде ромбов. Для записи самой команды , которая помещается внутри геометрической фигуры , используется естественный язык с элементами математической символики , если это необходимо.

Заметим , что в результате проверки условия возникают два возможных пути для продолжения алгоритма. Эти пути изображаются стрелками "+" и "-". Переход по стрелке "+" происходит в том случае , если условие соблюдено , а переход по стрелке "-" в том случае , если условие не соблюдено.

Ясно , что блок-схемы обладают большей наглядностью при изображении алгоритма. Однако эта наглядность быстро теряется при изображении сколько-нибудь большого алгоритма - в этом случае блок-схема получается плохо обозримой.



I.2.3. Псевдокоды.

Псевдокод представляет систему обозначений и правил , предназначенную для единообразной и точной записи алгоритмов. Он занимает промежуточное место между естественным языком и формальным языком программирования. С одной стороны , он близок к обычному естественному языку , поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны , в псевдокоде используются некоторые формальные конструкции и математическая символика , что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде отсутствуют строгие синтаксические правила , присущие формальным языкам программирования , что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд , рассчитанный на абстрактного исполнителя.

С другой стороны , в псевдокоде имеются конструкции и формы записи , присущие формальным языкам программирования , что облегчает дальнейшую запись алгоритма на языке программирования в виде, предназначенном для исполнения на ЭВМ. В частности , в псевдокоде так же , как и в языках программирования есть служебные слова , смысл которых определен раз и навсегда. Они выделяются в печатном тексте жирным шрифтом , а в рукописном тексте - подчеркиваются.

Возможны различные псевдокоды , различающиеся набором служебных слов и основных (базовых) конструкций. Покажем , как на одном из таких псевдокодов может быть представлен алгоритм пункта I.I:

```
алгоритм числогласных ;
начало
        записать в счетчик 0;
        установить указатель на первый символ текста;
        пока символ не есть ¶
          повторять
              начало
                    если символ есть гласная буква русского
                                алфавита
                      то
                        счетчик увеличить на 1
```

все ;
перевести указатель на следующий символ текста
конец;
взять число , находящееся в счетчике в качестве ответа ;
стоп
конец.

В силу своих особенностей псевдокоды , как и другие описанные выше средства записи алгоритмов , ориентированы на человека.

1.2.4. Языки программирования

Мы уже отмечали , что при записи алгоритма в виде блок-схемы или на псевдокоде допускается определенный произвол при изображении команд. Вместе с тем такая запись настолько точна , что позволяет человеку понять суть дела и , при необходимости , исполнить алгоритм.

Однако на практике алгоритмы создаются в расчете на их исполнение с помощью ЭВМ. Поэтому в конечном итоге алгоритм должен быть записан на языке , "понятном"ЭВМ. И здесь на первый план выдвигается необходимость точной и определенной записи команд , не оставляющей места для произвольного толкования. Поэтому язык для записи алгоритма , предназначенного для исполнения на ЭВМ , должен быть формализован. Такой язык принято называть языком программирования , а запись алгоритма на этом языке - программой для ЭВМ.

В настоящее время насчитывается несколько сотен различных языков программирования , расчитанных на разные сферы применения ЭВМ , то есть на разные классы решаемых с помощью ЭВМ задач. Эти языки классифицируют по разным уровням , учитывая степень зависимости языка от конкретной ЭВМ.

На самом нижнем уровне классификации находится машинный язык , то есть внутренний язык машины , на котором в конечном итоге представляется и исполняется программа для ЭВМ. Однако непосредственная запись алгоритма на машинном языке требует от разработчика чрезмерной детализации задачи , в результате запись получается не наглядной и трудной для понимания. Поэтому разработчики алгоритмов используют , как правило , языки программирования более высокого уровня , в которых принята символическая



форма записи , близкая в общепринятой математической.

При исполнении алгоритма на ЭВМ программа транслируется с языка высокого уровня на машинный язык , а затем уже исполняется. В силу того , что и язык программирования высокого уровня и машинный язык формализованы , трансляция программы может быть автоматизирована и выполнена с помощью той же ЭВМ.

Язык программирования высокого уровня вместе с соответствующим транслятором превращает реальную ЭВМ в виртуального исполнителя. Для человека дело обстоит так , как будто ЭВМ непосредственно понимает язык высокого уровня и исполняет алгоритм , записанный на этом языке. Поэтому можно говорить о Фортран-машине или Паскаль-машине в зависимости от используемого языка программирования. Подробно о этих языках программирования пойдет речь дальше.

## 1.3. Структуры алгоритмов.

Алгоритмы , предназначенные для решения широкого круга практических задач , связаны с преобразованием информации. При этом под информацией мы понимаем вполне конкретные величины - числовые , текстовые , графические и т.п. Именно такие алгоритмы , работающие с величинами и предназначенные для исполнения на ЭВМ , будут нас интересовать в дальнейшем. Процесс разработки алгоритма, предназначенного для исполнения на ЭВМ , называют программированием для ЭВМ.

Алгоритмы можно представлять как некоторые жесткие структуры , состоящие из отдельных базовых элементов. Такое представление об алгоритмах позволяет говорить о программировании , как о деятельности по систематическому конструированию алгоритмов в соответствии с определенной дисциплиной , предполагающей , что при построении алгоритма используются только конструкции из заранее фиксированного конечного набора базовых конструкций.

Естественно , что при таком подходе к программированию изучение основных принципов конструирования алгоритмов должно начинаться с изучения базовых элементов алгоритма. Для описания основных структурных элементов алгоритма мы будем использовать псевдокод и язык блок-схем.

I.3.I. Простые команды.

Наиболее элементарной структурной единицей любого алгоритма является простая команда , обозначающая один элементарный шаг переработки информации.

При исполнении алгоритма переработка информации состоит в изменении значений некоторых величин , с которыми работает алгоритм. Величины можно разбить на две группы - постоянные и переменные. Значения постоянных величин остаются неизменными в процессе исполнения алгоритма , а значения переменных - изменяются. С каждой величиной помимо значения связано также имя , используемое для обозначения величины. В качестве имени используется идентификатор , то есть последовательность букв и цифр , начинающаяся с буквы, например, X, X1, alfa, B25CD, Y81K и т.п.

Значение переменной величины может быть изменено с помощью команды присваивания , имеющей следующий вид:

< идентификатор > := < выражение >.

Здесь и далее в угловых скобках записываются основные понятия , которые в реальных командах заменяются на конкретные имена и конкретные выражения.

Знак присваивания (:=) обозначает указание исполнителю выполнить действие , состоящее в том , чтобы , во-первых , вычислить значение выражения , стоящего в правой части команды присваивания и , во-вторых , присвоить это значение переменной , имя которой стоит в левой части команды. Например , команда X:= 1 означает , что переменной X присваивается значение 1 , а команда У:= У+1 означает , что переменной У присваивается значение , которое на 1 больше ее прежнего значения.

Переменной величине может быть присвоено значение и с помощью команды ввода , которая передает исполнителю значение переменной из некоторого внешнего источника. Например , команда ввод (x,y) означает , что исполнитель получает из внешнего источника два значения , которые должны быть присвоены переменным x и y.

Аналогичная команда вывода: вывод(x,y) означает , что исполнитель должен выдать для обозрения значения величин x и y.

Перечисленные виды простых команд на языке блок-схем могут быть изображены в виде элементарного функционального блока , имеющего один вход и один выход (рис.I.2) , при этом внутри



блока записывается команда , осуществляющая переработку информации/см. стр. 3I/.

I.3.2. Составные команды.

Из простых команд и проверки условий образуются составные команды , имеющие более сложную структуру. Ниже рассматриваются основные типы составных команд алгоритма.

Следование.

Эта составная команда образуется из последовательности команд , следующих одна за другой. При записи на псевдокоде команды последовательности отделяются одна от другой с помощью точек с запятой. При исполнении алгоритма команды последовательности выполняются одна за одной в том порядке , как они записаны.

Для обозначения начала и конца команды следования используются служебные слова начало и конец .

В общем виде команда следования может быть представлена так:

начало < действие > ; <действие > ; ... <действие > конец

Под действием понимается либо простая , либо составная команда . Эти команды могут записываться либо в строчку , либо в столбец одна под одной. Мы в дальнейшем будем использовать второй способ записи.

Служебные слова начало и конец играют роль скобок. Наличие скобок позволяет рассматривать команду следования как единое действие , распадающееся на последовательность более простых действий.

Приведем пример команды следования:

начало
ввод (x);
y := $x^2+5$;
z := $\sqrt{x^2+y^2}$
конец

Запись команды следования на языке блок-схем будет иметь вид , показанный на рис. I.3/стр. 3I/.

Развилка (ветвление).

С помощью этой команды осуществляется выбор одного из двух возможных действий в зависимости от условия.

На псевдокоде эта команда в общем виде записывается так:

```
если < условие >
  то      < действие 1 >
  иначе   < действие 2 >
все
```

Действия , указанные после служебных слов то и иначе , могут быть простыми или составными командами. При исполнении команды ветвления выполняется только одно из действий: если условие соблюдено , то выполняется действие I , в противном случае - действие 2.

На рис. I.4 команда ветвления изображена на языке блок-схем. В случае , когда условие соблюдено , продолжение исполнения алгоритма происходит по стрелке "+" , в противном случае - по стрелке "-" /см. стр. 32/.

Команда ветвления может использоваться в сокращенной форме ( такую форму развилки называют также коррекцией) , когда в случае несоблюдения условия никакое действие не выполняется. На псевдокоде коррекция записывается так:

```
если  < условие >
  то    < действие >
все
```

Изображение коррекции в виде блок-схемы показано на рис. I.5.

В качестве примера команды ветвления рассмотрим вычисление значения функции y , которая задана формулой:

$$y = \begin{cases} x^2 , & \text{если } x < 0 \\ x+1 , & \text{если } x \geqslant 0 \end{cases}$$

На псевдокоде команда ветвления для вычисления значения y будет иметь вид:

```
если    x ⩾ 0
  то    y:=x +1/2
  иначе y:= x^2
все
```

В команде ветвления после служебных слов то или иначе может стоять составная команда , например:



```
если    x ⩾ 0
  то
     начало
          y:=x
          z:= x+y
     конец
  иначе      y:=x^2
все
```

Цикл.

Большинство алгоритмов содержат серии многократно повторяемых команд. Если такие команды записывать в виде составной команды следования , то каждую повторяемую команду пришлось бы выписать ровно столько раз , сколько раз она повторяется. Ясно , что это очень неэкономный способ записи. Поэтому для обозначения многократно повторяемых действий используют специальную конструкцию , называемую циклом. Составная команда цикла , называемая также командой повторения , содержит условие , которое используется для определения числа повторений. Мы рассмотрим два типа команды повторения.

Команда цикла с предусловием записывается на псевдокоде в следующем виде:

```
пока < условие >
  повторять    < действие >
```

Под действием мы , как и прежде , понимаем простую или составную команду. Исполнение команды цикла состоит в том , что сначала проверяется условие ( отсюда и название - цикл с предусловием) и , если оно соблюдено , то выполняется команда , записанная после служебного слова повторять. После этого снова проверяется условие. Выполнение цикла завершается , когда условие перестает соблюдаться. Для этого необходимо , чтобы команда , выполняемая в цикле , влияла на условие.

Запись команды повторения с предусловием на языке блок-схемы приведена на рис. I.6 /см. стр. 33/.

Второй тип команды повторения - цикл с постусловием выполняется аналогично , только условие проверяется после выполнения команды , а повторение выполнения команды происходит в том случае , когда условие не соблюдено , то есть повторение произво-

дится до соблюдения условия ( поэтому этот тип цикла называют также циклом "до").

На псевдокоде цикл с постусловием записывается в виде:

повторять < действие >
до < условие >

Блок-схема этого цикла приведена на рис. I.7/стр. 33/.

Приведем фрагмент алгоритма , в котором используется команда повторения. Пусть векторы x и y заданы своими координатами: $x_1, x_2, \ldots, x_n$ и $y_1, y_2, \ldots, y_n$ . Требуется найти их скалярное произведение: сумма $= \sum_{i=1}^{n} x_i \cdot y_i$ . Запишем это вычисление на псевдокоде , используя команду цикла с предусловием:

```
сумма := 0 ;
  i := 1 ;
пока  i ≤ n
  повторять
          начало
                сумма := сумма + x_i * y_i
                i := i + 1
          конец ;
```

Этот же фрагмент может быть записан с использованием команды цикла с постусловием:

```
сумма := 0;
  i := 1 ;
повторять
          начало
                сумма := сумма + x_i * y_i ;
                i := i + 1
          конец
до   i > n
```

I.3.3 Комбинации базовых команд .

Известно , что любой алгоритм может быть построен с использованием только трех базовых конструкций: следования , развилки и цикла. Это превращает разработку алгоритма в "сборку" его конструкции из имеющего набора базовых конструкций подобно тому , как конструктор собирает механизм из конечного набора имеющихся в его распоряжении деталей.



"Сборка"алгоритма может происходить двумя путями. Во-первых, базовые элементы могут соединяться в последовательность , образуя конструкцию следования. Это возможно , так как каждая базовая конструкция имеет один вход и один выход. Это особенно хорошо видно на соответствующих блок-схемах. Во-вторых , одна базовая конструкция может вкладываться в другую конструкцию , образуя "вложенные" конструкции. Это также возможно , так как внутри составных команд могут быть снова составные команды.

Таким образом при построении алгоритма он может развиваться как "вширь" путем подключения в цепочку новых конструкций , так и "вглубь" путем включения одних конструкций в другие. Такое конструирование обычно и применяется на практике. Алгоритм строится в несколько этапов - сначала он формулируется в самых общих чертах , а затем уточняется путем детализации более крупных действий через более мелкие. Этот метод известен под названием метода пошаговой детализации или метода "сверху - вниз".

Строгая дисциплина конструирования алгоритма позволяет получить алгоритм с ясно выраженной структурой , что облегчает понимание и доказательство правильности алгоритма. При записи алгоритма на псевдокоде получается текст , который можно читать без перерыва сверху вниз , как обычный текст. Кроме того , последовательная структура полученного при конструировании алгоритма означает , что при исполнении его команды будут выполняться в том порядке , в котором они появляются в тексте алгоритма. Это значительно облегчает переход от статического объекта - текста алгоритма к динамическому процессу его исполнения.

I.3.4. Примеры .

Запись алгоритма на псевдокоде начинается с заголовка алгоритма , содержащего служебное слово алгоритм , за которым следует имя алгоритма ( идентификатор). Наличие имени в заголовке позволяет в дальнейшем ссылаться на этот алгоритм.

Пример I. Вычисление факториала.

Известно , что $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1)n$. Поэтому процесс вычисления $n!$ может быть описан с помощью команды цикла.

```
алгоритм  факториал ;
начало
      ввод (n);
     f :=1 ;
     к:=0 ;
     пока   к ≠ n
      повторять
                начало
                        к := к+1 ;
                        f := f · к
                конец ;
      вывод (f)
конец
```

По поводу этой записи сделаем несколько замечаний. В алгоритме используется команда цикла с предусловием. Это имеет принципиальное значение , так как такой алгоритм будет работать верно и при $n=0$. Действительно , в этом случае действия , заданные командой цикла, не будут ни разу выполнены , и в результате будет выдан верный ответ $f = 1$ ( по определению $0!=1$ ).

Мы использовали "ступенчатую" форму записи , позволяющую явно выделить основные структурные элементы алгоритма и облегчающую его понимание.

Запись этого алгоритма в виде блок-схемы приведена на рис. I.8/стр. 84/.

Пример 2. Поиск в таблице

Пусть в линейной таблице записана последовательность чисел $x_1, x_2, \ldots, x_n$ , причем числа этой последовательности никак не упорядочены по величине. Пусть задано также число $m$ . Требуется выяснить, содержится ли это число в таблице и , если содержится , то определить его порядковый номер.

Так как числа в линейной таблице не упорядочены , то поиск числа $m$ сводится к последовательному просмотру чисел и сравнению их с числом $m$. Такой просмотр может быть организован с помощью команды цикла. Заметим , что просмотр должен заканчиваться , как только будет обнаружено число $m$ или когда будет просмотрена вся таблица , поэтому условие окончания цикла должно состоять из двух условий , причем невыполнение любого из условий ведет к завершению цикла.



Алгоритм должен выдавать сообщение о том , содержится ли число $m$ в таблице. Для этого будем использовать переменную с , значением которой будет текст. Такие переменные называют текстовыми или литерными.

Для обозначения типа переменных , в тексте алгоритма может быть предусмотрен специальный раздел описания переменных. Описание типа переменных предусмотрено в большинстве языков программирования. В псевдокоде такое описание может опускаться , так как тип переменной в целом ряде случаев ясен из самого контекста алгоритма.

Для определения порядкового номера числа $m$ будем использовать переменную к. До начала просмотра таблицы переменной к будет присвоено значение $0$ , а переменной с - текстовое значение:"числа $m$ в таблице нет" (текст заключается в апострофы). Учитывая это , получим такую запись алгоритма на псевдокоде:

```
алгоритм поиск ;
начало
      ввод (m, x1, x2, ..., xn );
      к:=0 ;
      с:= 'числа m в таблице нет' ;
      i:= 1;
       пока  i ≤ n  и  xi ≠ m
        повторять
                        i := i +1 ;
       если   i ≤ n
        то начало
                        с:= 'число m есть в таблице' ;
                        к:= i
           конец
       все ;
       вывод ( с,к )
конец
```

Заметим , что команда цикла может завершиться либо при условии $i > n$ ( просмотрена вся таблица и число $m$ не обнаружено), либо при условии $x_i = m$ ( в этом случае $i \le n$ и значение $i$ является порядковым номером числа $m$ в таблице. Поэтому за командой цикла следует команда коррекции , с помощью которой переменным с и к присваиваются соответствующие значения ( когда число

$m$ найдено в таблице). Если число $m$ не будет найдено , то значения с и к останутся такими же , как и до начала поиска.

Блок-схема алгоритма поиска приведена на рис. I .9 /стр. 35/.

Пример 3. Сумма положительных элементов матрицы.

Пусть задана квадратная матрица $A=\|a_{ij}\|$ $1 \le i,j \le n$ . Необходимо найти сумму всех положительных элементов этой матрицы.

Очевидно для нахождения суммы положительных элементов матрицы необходимо просматривать один за одним элементы матрицы и прибавлять каждое положительное число к сумме , первоначальное значение которой - ноль.

Так как матрица представляет двумерную таблицу чисел , расположенных по строкам и столбцам , то такой просмотр можно организовать либо по строкам , либо по столбцам. Примем первый способ просмотра. Тогда исполнитель должен последовательно перебирать элементы каждой строки, пока она не кончится , а при окончании очередной строки - перейти к следующей строке и так поступать до тех пор , пока не кончатся все строки матрицы.

Такой алгоритм может быть описан с помощью вложенных команд цикла:

```
алгоритм сумма ;
начало
    ввод (a_ij , 1≤ i,j ≤ n ) ;
    сумма:=0 ;
    i:= 1 ;
    повторять
        начало
            j := 1 ;
            повторять
                начало
                    если a_ij > 0
                        то сумма:=сумма+ a_ij
                    все ;
                    j :=j + 1
                конец
            до j >n ;
               i:=i+ 1
        конец
    до i >n ;
```



```
вывод ( сумма)
конец
```

Заметим , что в этом алгоритме использованы вложенные команды цикла с постусловием. Однако с таким же успехом можно было бы использовать и вложенные команды цикла с предусловием.

Блок-схема алгоритма изображена на рис. I.I0/стр. 36/.

I.3.5. Вспомогательные (подчиненные) алгоритмы.

Часто при построении алгоритма оказывается возможным использовать уже разработанные ранее алгоритмы. Так , при построении алгоритма по принципу "сверху-вниз" обычно стараются всю задачу разбить на более простые подзадачи , и если для какой-то подзадачи уже построен алгоритм , он может быть включен в состав вновь разрабатываемого алгоритма.

Это важно в том отношении , что позволяет не повторять уже проделанную работу и экономит силы и время на разработку нового алгоритма. Кроме того это позволяет избежать при записи нового алгоритма повторения текста в той его части , которая уже описана в готовом алгоритме.

Готовые алгоритмы , целиком включаемые в состав разрабатываемого алгоритма , называют вспомогательными или подчиненными алгоритмами в отличие от главного или основного алгоритма , в состав которого они включаются.

Использование вспомогательных алгоритмов вызывает необходимость оформлять их специальным образом , чтобы иметь возможность в дальнейшем ссылаться на них в основном алгоритме. Формальные способы оформления таких алгоритмов широко применяются в языках программирования , а сами вспомогательные алгоритмы , записанные на языках программирования , называют подпрограммами или процедурами.

При использовании псевдокода или блок-схем полная формализация в оформлении подчиненных алгоритмов , как правило , не производится. Однако и здесь используются некоторые общие принципы оформления.

В заголовке подчиненного алгоритма следом за именем может указываться в круглых скобках список так называемых формальных параметров. Такой подчиненный алгоритм называют алгоритмом с параметрами. В списке формальных параметров указываются имена входных и выходных величин ( аргументов и результатов) алгоритма.

Это необходимо для того , чтобы при ссылке на подчиненный алгоритм в основном алгоритме можно было задать значения аргументов, а после исполнения подчиненного алгоритма - воспользоваться результатами - значениями соответствующих переменных.

Ссылка на вспомогательный алгоритм в основном алгоритме осуществляется с помощью специальной команды вызова вспомогательного алгоритма , в которой указывается имя подчиненного алгоритма и список фактических параметров , которые должны быть подставлены вместо формальных параметров при исполнении вспомогательного алгоритма. Таким образом команда вызова вспомогательного алгоритма имеет вид:

< имя вспомогательного алгоритма> ( < список фактических параметров>)

Исполнение такой команды эквивалентно исполнению вспомогательного алгоритма.

Сказанное поясним на таком примере. Предположим , что необходимо определить частное и остаток от деления двух натуральных чисел. Для исполнителя , имеющего в системе команд команду деления нацело ( обозначим ее ÷) , алгоритм нахождения частного ($q$) и остатка ($r$) при делении двух натуральных чисел может быть представлен в таком виде:

```
алгоритм частноеостаток (a, b, q, r);
начало
    если   a > b
    то начало
            q := a ÷ b
            r := a - q · b
       конец
    иначе начало
            q := b ÷ a
            r := b - q · a
       конец
    все
конец
```

Это алгоритм оформлен как вспомогательный алгоритм с параметрами.

Предположим теперь , что заданы две последовательности натуральных чисел одинаковой длины: $x_1, x_2, \ldots, x_n$ и $y_1, y_2, \ldots, y_n$. Необходимо подсчитать число ($N$) пар $(x_i, y_i)$, $1 \le i \le n$, соответствующих друг другу кратных чисел этих последовательностей и найти сумму частных от деления большего числа каждого такой пары на меньшее. Очевидно , что эта задача может быть решена путем последовательного перебора всех пар соответствующих чисел этих последовательностей и применения к каждой паре подчиненного алгоритма "частноеостаток".



Основной алгоритм , который мы назовем " кратные пары" в записи на псевдокоде будет иметь вид:

```
алгоритм кратныепары ;
начало
    ввод (x1, x2, ..., xn, y1, y2, ..., yn );
     N := 0 ;
    сумма := 0 ;
     i := 1 ;
    пока   i ≤ n
     повторять
          начало
                частноеостаток (xi, yi, q, r);
                если  r = 0
                  то начало
                          N := N + 1
                          сумма:=сумма + q
                     конец
                все ;
                i := i +1
          конец ;
     вывод ( N, сумма)
конец
```

На блок-схемах вспомогательные алгоритмы обозначаются в виде специальной геометрической фигуры ("флажка") , внутри которой записывается команда вызова вспомогательного алгоритма. Блок-схема алгоритма "кратныепары" приведена на рис.I.II/стр. 37/.

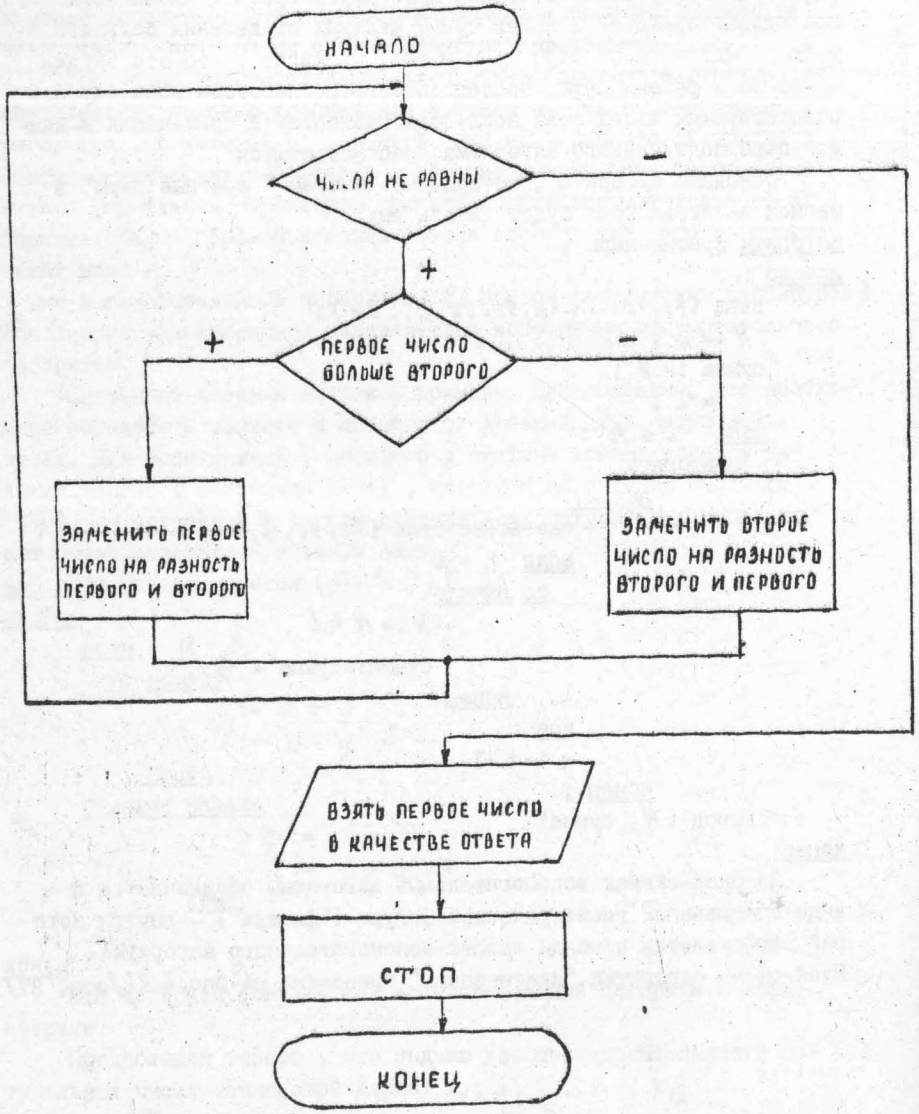



Рис.I.1. Блок-схема алгоритма Евклида.



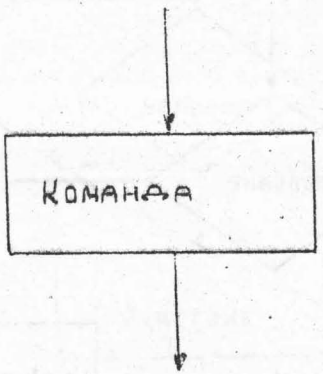



Рис.I.2. Функциональный блок.

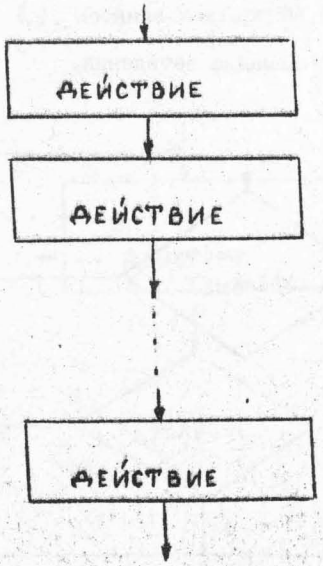



Рис. I.3. Команда следования.

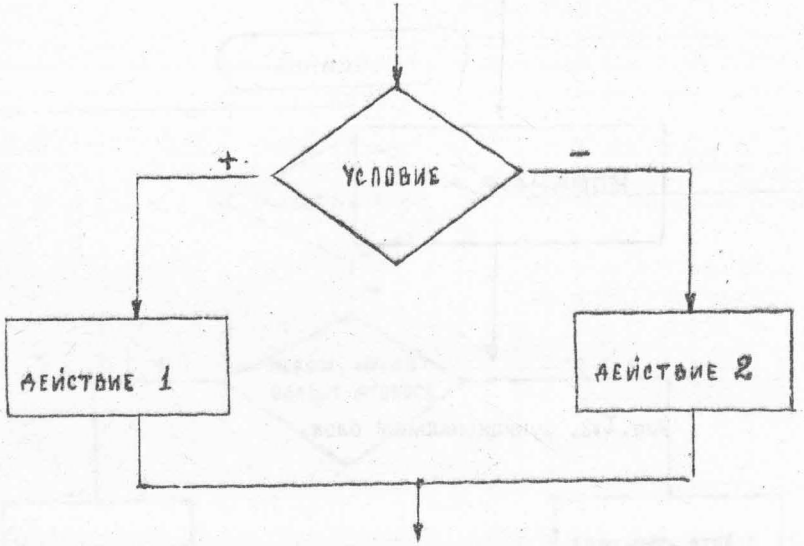



Рис. I.4. Команда ветвления.

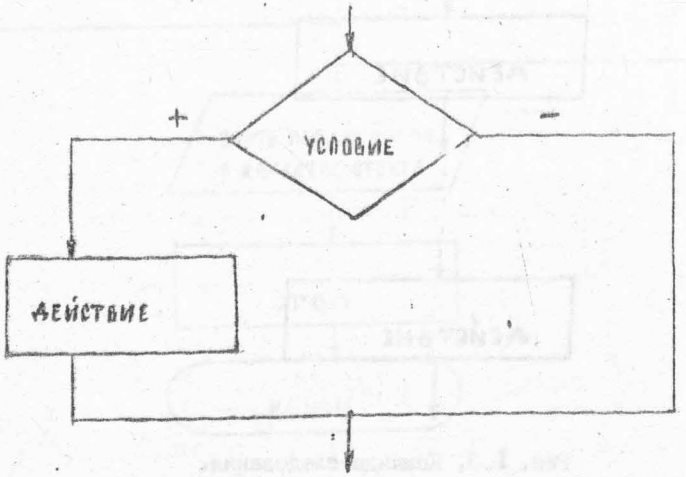



Рис. I.5. Команда ветвления в сокращенной форме (коррекция)



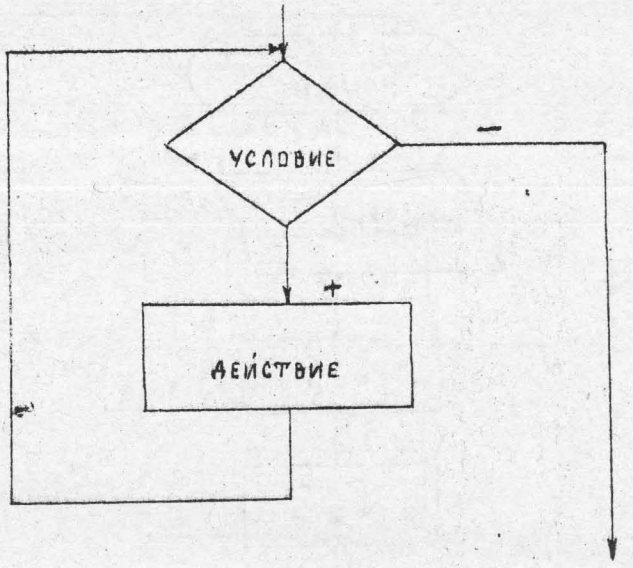



Рис. I.6. Команда повторения с предусловием.

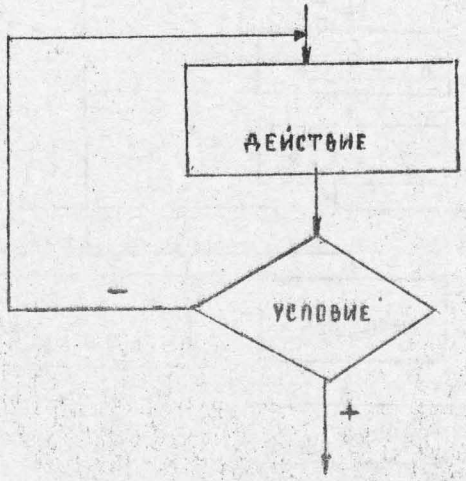



Рис. I.7. Команда повторения с постусловием.

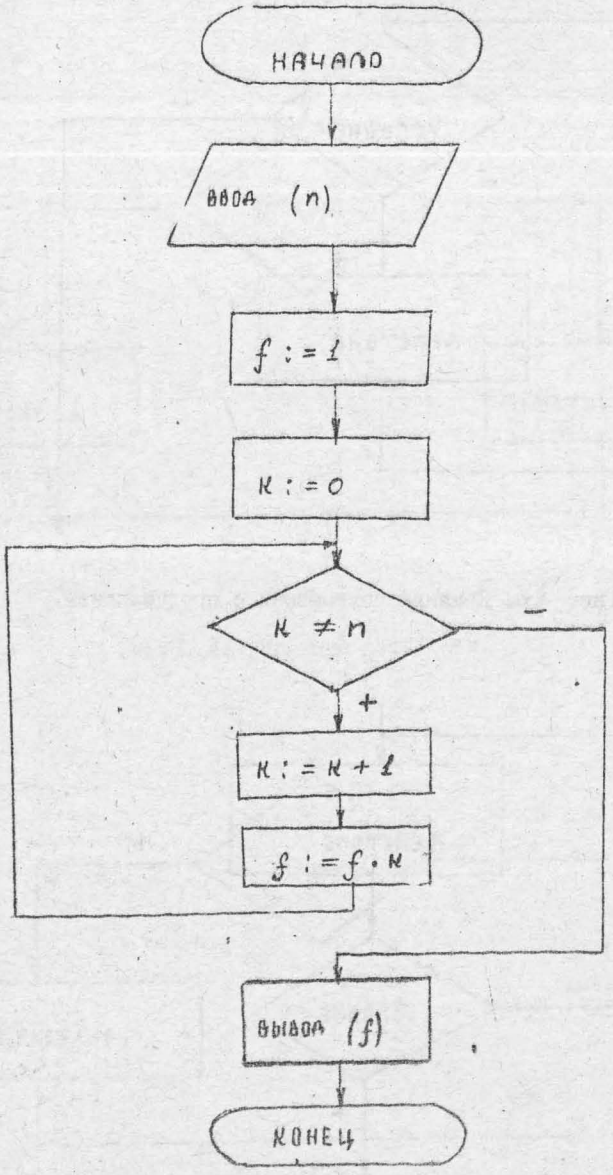



Рис. I.8. Блок-схема алгоритма "факториал".



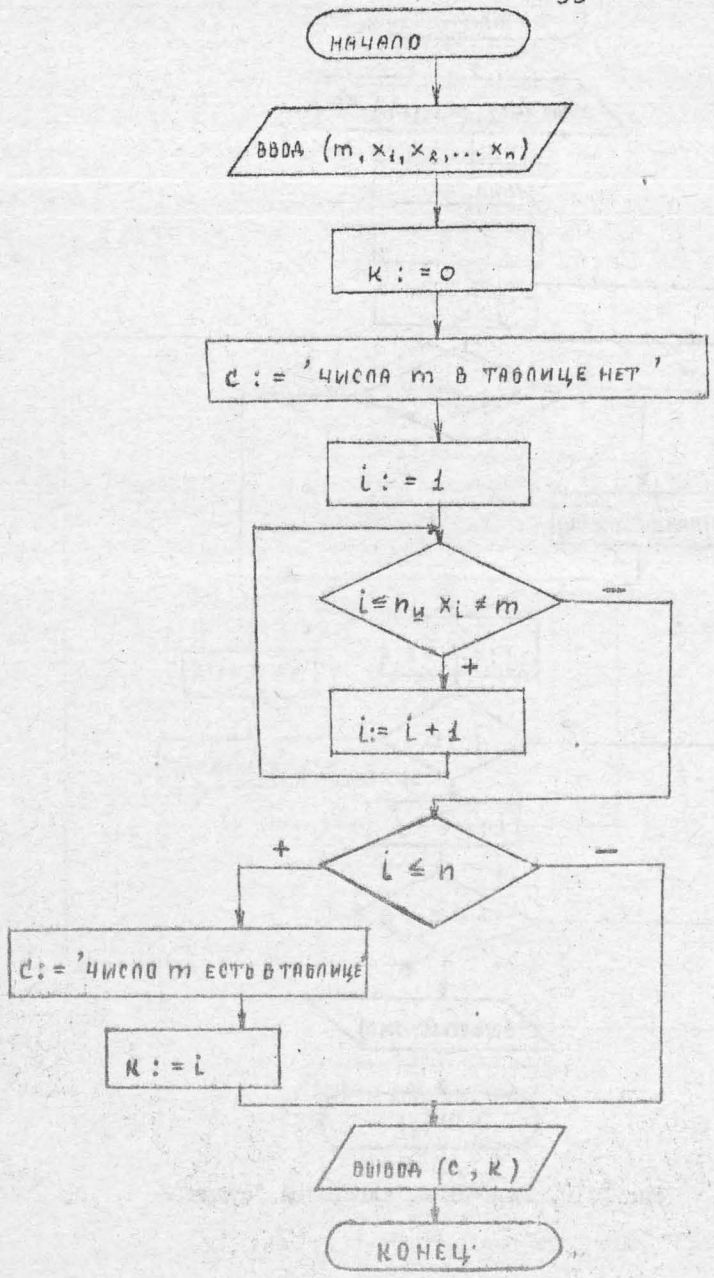



Рис. I.9. Блок-схема алгоритма "поиск "

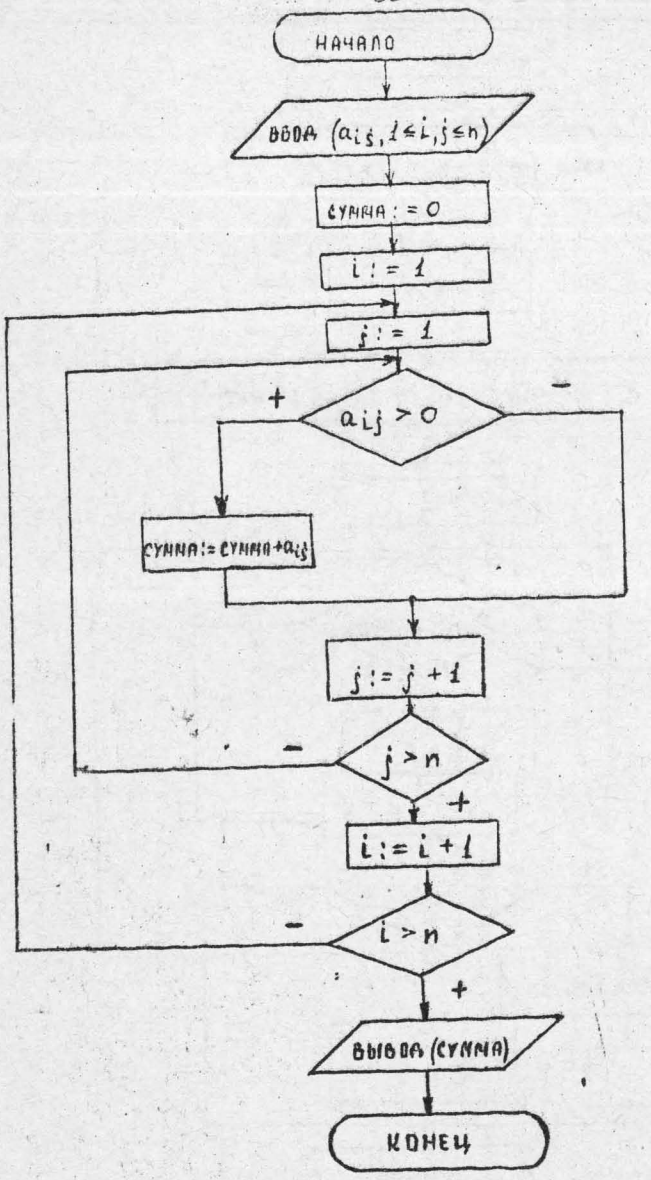



Рис.I.IO. Блок-схема алгоритма "сумма".



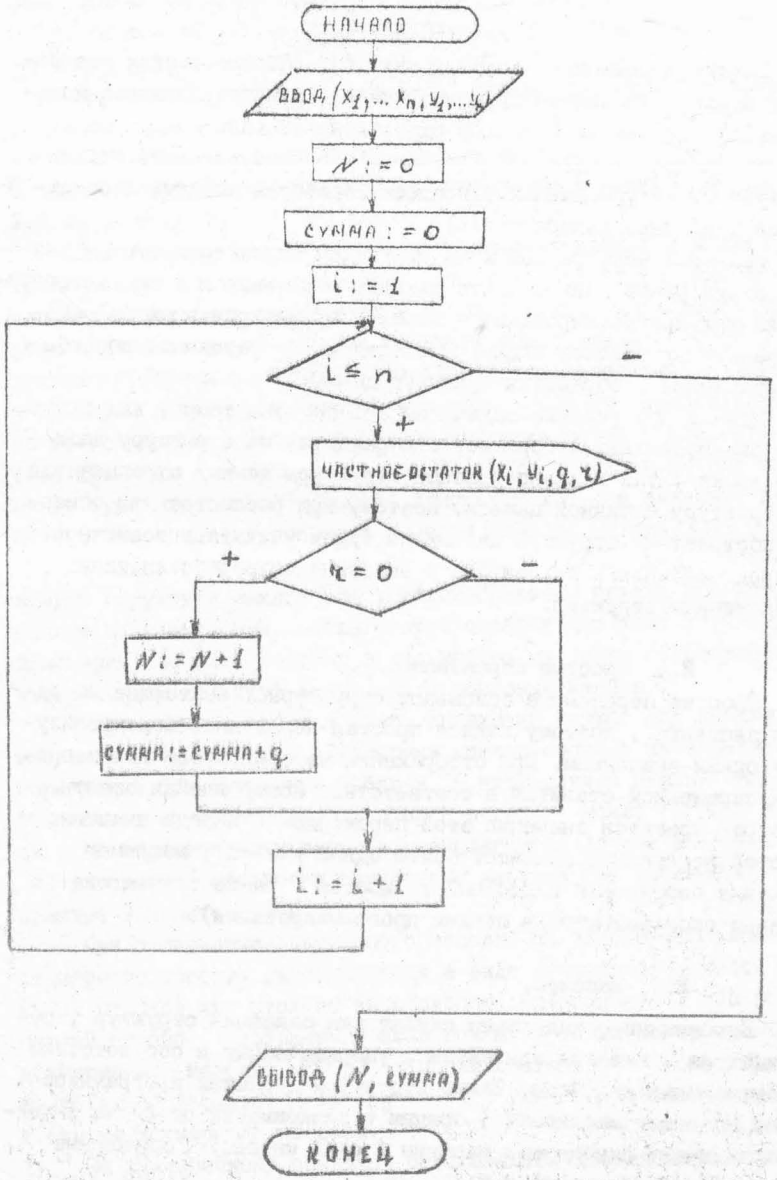



Рис. I.II. Блок-схема алгоритма "кратныепар-ы"

## ТЕМА 2. СТРУКТУРЫ ДАННЫХ

Задачи , решаемые на ЭВМ , являются математическим моделями процессов или явлений реальной жизни. В математической модели находят отражение наиболее существенные связи между реальными объектами. Модели объектов вместе с присущими им связями образуют структуры данных , процесс обработки которых и описывается с помощью алгоритмов.

Разные классы решаемых на ЭВМ задач характеризуются и разными структурами данных , что находит отражение и в соответствующих языках программирования - языки , ориентированные на решение различных классов задач , используют различные способы представления и обработки структур данных.

Однако при решении задачи на ЭВМ подобно тому , как структуры разнообразных алгоритмов отображаются на структуру машинного языка , так и разнообразные структуры данных отображаются на структуру машинной памяти. Поэтому при рассмотрениии основных абстрактных структур данных мы будем учитывать возможность их представления в памяти ЭВМ и особенности обработки таких отображенных структур.

### 2.1. Простые переменные.

Простые переменные описывают структуры , состоящие из одного элемента , поэтому каждая простая переменная характеризуется одним значением. При отображении на память ЭВМ имени простой переменной ставится в соответствие номер ячейки памяти , в которой хранится значение этой переменной ( иногда значение простой переменной занимает более одной ячейки , например , значение переменной с двойной точностью . такие случаи специально оговариваются в языках программирования).

### 2.2. Массивы.

Переменные с индексами служат для описания структур , состоящих из множества элементов , упорядоченных в соответствии со значениями индексов. Такие структуры в языках программирования называют массивами , причем в зависимости от числа индексов различают одномерные массивы ( один индекс) , двумерные массивы ( два индекса) и т.д.

В языках программирования индексы принято записывать в скобках после имени переменной , а не в качестве подстрочных символов , как это характерно для обычной математической записи. Таким же обозначением мы будем пользоваться в дальнейшем и при записи на псевдокоде , например , мы будем писать $X[1]$, $a[i,j]$, $y[i+1]$, $z[j,k]$ и т.п.



Переменная с одним индексом описывает одномерный массив. Элементы одномерного массива образуют последовательность , причем номер элемента последовательности определяется индексом. С такой структурой данных мы уже встречались в примере 2 пункта I.3.4.

Значения элементов одномерного массива хранят в памяти также в виде последовательности , то есть структура одномерного массива однозначно отображается на структуру памяти ЭВМ , при этом индекс элемента массива означает номер ячейки памяти , в которой хранится значение этого элемента , относительно номера той ячейки памяти , в которой хранится значение первого элемента этого массива. Например , если значение первого элемента массива хранится в ячейке с номером I0I , то значение пятого элемента будет храниться в ячейке с номером I05 , а значение двадцатого элемента - в ячейке с номером I20.

Так как структура одномерного массива однозначно соответствует структуре памяти ЭВМ , то отображение других структур данных на память ЭВМ соответствует отображению этих структур на одномерный массив. Поэтому в дальнейшем мы будем пользоваться отображением структур данных на одномерный массив.

Переменные с двумя и более индексами служат для описания многомерных массивов. Двумерный массив состоит из элементов , образующих прямоугольную таблицу. В этом случае первый индекс обозначает номер строки , а второй индекс - номер стрлбца таблицы , на пересечении которых и располагается данный элемент. Один из алгоритмов обработки двумерного массива был рассмотрен в примере 3 пункта I.3.4.

При отображении двумерного массива на одномерный элементы двумерного массива располагаются в виде последовательности строка за строкой или столбец за столбцом. Предположим , что принят первый способ ( отображение по строкам) , тогда номер элемента двумерного массива $X[i,j]$ в последовательности может быть вычислен по формуле: $(i-1)\cdot n+j$ , где $n$ - число элементов в строке двумерного массива.

По аналогичному принципу отображаются на одномерный массив

и массивы большей размерности.

Массив как структура данных характерен тем , что с помощью индексов обеспечивается прямой доступ к любому элементу массива, поэтому операции выбора элемента массива или записи в массив сводятся к вычислению значений индексов.

## 2.3. Очереди

Очереди - хорошо известные структуры данных , организованные по принципу " первым пришел - первым ушел". Это динамические структуры , число элементов которых может меняться в процессе обработки.

Обработка элементов очереди ведется последовательно один за одним , причем элемент , который первым попал в очередь , первым и обрабатывается и при этом покидает очередь. Добавление новых элементов производится в конец очереди.

Элементы , находящиеся в середине очереди между ее первым и последним элементами,не доступны для обработки. Следовательно, в очереди доступны только два элемента - первый ("голова") и последний ("хвост). Над первым элементом выполняются операции чтения и обработки , над последним - операция записи в очередь.

Для отображения очереди используется одномерный массив $a[1], a[2], \ldots, a[n]$ , причем длина( $n$ ) этого массива выбирается с определенным запасом , чтобы длина очереди ( число ее элементов) не превышала бы $n$ .

При отображении очереди на одномерный массив можно поступить , например , так : первый элемент очереди хранить в первом элементе массива , записывая вновь поступающие элементы в следующие свободные элементы массива. После обработки первого элемента он удаляется из очереди и на его место переписывается следующий за ним элемент. Это приведет однако к перемещению и всех следующих элементов очереди: на место второго элемента будет переписан третий , на место третьего - четвертый и т.д. Очевидно , что это не очень эффективный способ представления , хотя он и отражает реальный процесс движения очереди.

На практике для отображения очереди применяют другой способ, использующий два указателя , один из которых указывает на элемент массива , хранящий "голову" очереди , а другой - на элемент



массива , предназначенный для записи очередного элемента очереди ("хвост "). В этом случае динамика движения очереди находит отражение в перемещении указателей по элементам массива.

Итак , пусть $i$ - индекс массива , хранящего "голову" очереди , а $j$ - индекс первого свободного элемента массива , куда поступает новый элемент очереди ("хвост"). Тогда выборка очередного элемента очереди для обработки сводится к выполнению операций: $x = a[i]; i = i + 1$ . После этого индекс $i$ указывает на следующий элемент очереди , то есть " головой" очереди становится следующий по порядку элемент.

Запись нового элемента в очередь сводится к выполнению операций: $a[j] := y; j := j + 1$ . После этого индекс $j$ снова указывает на первый свободный элемент массива.

Элементы очереди могут поступать и обрабатываться неравномерно , поэтому длина очереди будет изменяться , в частности , возможен случай , когда очередь окажется пустой. Признаком этого может служить равенство значений $i = j$ после чтения головного элемента очереди.

В процессе обработки очереди ее элементы будут смещаться к концу массива , так как индекс $j$ увеличивается после каждой записи в очередь. Поэтому возможен случай , когда после записи в очередь очередного элемента $j$ станет больше $n$ . В этом случае необходимо сместить очередь в начало массива , то есть положить $i = 1$ и последовательно переписать все элементы очереди в элементы массива , начиная с $a[1]$ .

Чтобы избежать такой дополнительной работы , можно считать массив замкнутым по кольцу , то есть элементом , следующим за последним элементом массива , считать его первый элемент. Тогда при $j > n$ "хвост" очереди перемещается в начало массива.

Отображение очереди на "кольцевой" массив из пяти элементов показано на рис. 2.1 .

Хотя значение $n$ выбирается достаточно большим , все-таки возможен случай , когда после записи в очередь очередного элемента выполняется условие $j = i$ . Это означает , что "хвост" совпадает с "головой" , то есть происходит ее переполнение. В этом случае , как и в случае пустой очереди , необходимо выдать соответствующее сообщение.

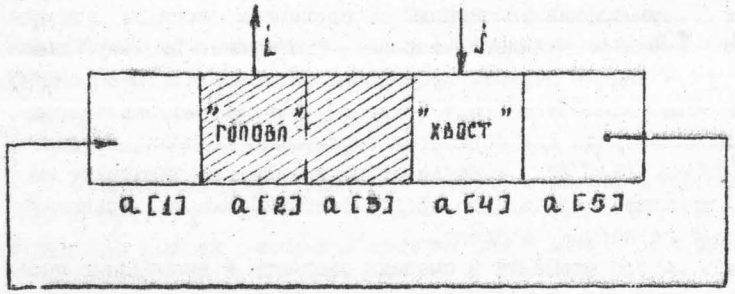



Рис. 2.I. Организация очереди.

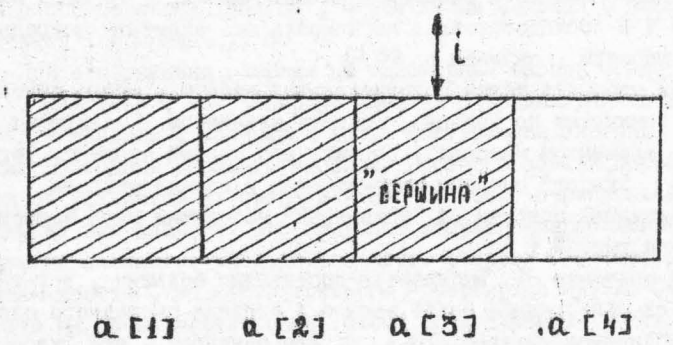



Рис. 2.2. Организация стека.



С учетом сказанного алгоритмы записи в очередь и чтения из очереди могут быть представлены в таком виде:

```
алгоритм запись_в_очередь;
начало
        a[j] := y;
        если   j ≠ n
          то   j := j + 1
          иначе  j := 1
        все;
        если   j = i
          то вывод ("переполнение очереди")
        все
конец

алгоритм чтение_из_очереди ;
начало
        x := a[i];
        если   i ≠ n
          то   i := i + 1
          иначе  i := 1
        все;
        если   i = j
          то вывод ("пустая очередь")
        все
конец
```

## 2.4. Стеки

Стек представляет другой тип очереди , организованной по принципу " последним пришел -первым ушел". В стеке в отличие от очереди доступен только один элемент , называемый " вершиной" стека. При записи в стек очередной элемент заносится в его "вершину" ,а остальные элементы продвигаются вниз без изменения порядка. При выборке из стека выбирается элемент из его "вершины", а все остальные элементы без изменения порядка сдвигаются вверх так , что в "вершину" попадает элемент , поступивший в стек предпоследним.

По принципу стека организована , например , стопка книг на столе. Новую книгу мы кладем поверх стопки. Эту же книгу мы воз-

мем из стопки первой , после этого наверху стопки окажется книга, положенная туда предпоследней. По такому же принципу организована пистолетная или автоматная обойма ( магазин) - патрон , заправленный в обойму последним , попадает из обоймы в ствол первым. По этой причине стек называют также магазинной памятью.

Стек можно отразить на одномерный массив $a[1], a[2], \ldots, a[n]$ по тому же принципу , что и очередь. Для указания " вершины" стека можно использовать индекс $i$ . При записи в стек указатель вершины будет сдвигаться в сторону конца массива , при чтении из стека указатель "вершины" будет перемещаться в сторону начала массива. Значение $i = 0$ перед чтением из стека служит признаком того , что стек пуст , а значение $i = n$ перед записью в стек - признаком того , что стек переполнен.

Отображение стека на массив из четырех элементов показано на рис.2.2 .

Алгоритмы записи в стек и чтения из стека могут быть записаны в таком виде:

```
алгоритм записьвстек ;
начало
      если  i ≠ n
        то начало
                 i := i + 1;
                 a[i] := y
           конец
        иначе   вывод ("переполнение стека")
      все
конец

Алгоритм чтениеизстека ;
начало
      если  i ≠ 0
        то начало
                 x := a[i];
                 i := i - 1
           конец
        иначе вывод ("стек пустой")
      все
конец
```



Стек удобен для вычисления значения арифметического выражения , представленного в так называемой обратной польской (бесскобочной) записи. Такие вычисления с помощью стека реализуются , например, во многих калькуляторах.

## 2.5 Строки.

Строка - это последовательность символов из некоторого алфавита. Основные операции , выполняемые над строками , состоят в последовательном переборе символов строки , включении в строку нового символа , исключении из строки заданного символа , включении или исключении группы подряд идущих символов (подстроки).

При отображении строк на одномерный массив можно поступить так же , как и при отображении очередей , то есть размещать последовательные символы строки в последовательных элементах массива. Но в этом случае включение или исключение символов приведет к раздвижке или сжатию строки , что вызовет перемещение других символов строки , начиная с места разрыва , по элементам массива. Поэтому такое представление для строк оказывается уже мало пригодным.

Для представления строк , как и некоторых других структур данных , применяют другой принцип , состоящий в том , что каждый элемент структуры отображается на несколько соседних элементов массива , часть которых содержит информацию о данном элементе структуры , а другая часть - ссылки на соседние элементы структуры.

По этому принципу каждый элемент строки может быть отображен на два соседних элемента массива - в одном из этих элементов записывается данный символ строки ( точнее- код символа) , а в другом элементе - ссылка на следующий символ строки , то есть индекс первого из двух элементов массива , на которые отображен следующий символ строки. Каждые два элемента массива , на которые отображается один символ строки , называют звеном. С помощью ссылок звенья сцепляются в цепочку , причем соседние звенья теперь совсем не обязательно должны занимать соседние элементы массива - звенья могут следовать в произвольном порядке по элементам массива , а их истинный порядок в цепочке определяется с помощью ссылок.

Например , строка из трех символов: 'п2с' может быть размещена в восьми элементах массива , как это показано на рис 2.3 . Первые два элемента массива занимает служебная информация: первый элемент массива содержит ссылку на первое звено цепочки , а второй элемент массива - ссылку на первый свободный элемент массива , куда может быть записано новое звено цепочки. В каждом звене первый элемент содержит ссылку на следующее звено , а второй элемент содержит символ строки. Последнее звено цепочки имеет нулевую ссылку , что и является признаком конца цепочки.

При таком представлении строки включение нового символа сводится к образованию нового звена в свободном месте массива и изменению некоторых ссылок. Например , пусть в строку 'п2с' нужно вставить символ 'е' после символа 'п' , то есть получить новую строку 'пе2с' . В этом случае новое звено размещается в девятом и десятом элементах массива. Меняется ссылка в звене до разрыва цепочки ( в первом звене) , в новом звене указывается ссылка на звено после разрыва цепочки ( на третье звено) и , наконец , меняется ссылка на первый свободный элемент массива ( значение ссылки увеличивается на 2). В результате получается цепочка , изображенная на рис.2.4 .

Алгоритм включения символа в строку оформим как подчиненный алгоритм с параметрами х и к , где х - символ , включаемый в строку , а к - индекс звена цепочки , после которого включается новое звено , содержащее этот символ:

```
алгоритм включениесимвола(х,к);
начало
    новый:= a[2];     {индекс первого свободного элемента
                       массива}
    a[новый+1]:= x;   {запись нового символа}
    a[новый]:=a[k];   {ссылка на звено после разрыва цепочки}
    a[k]:= новый      {ссылка на звено до разрыва цепочки}
    a[2]:= a[2]+2     {ссылка на первый свободный элемент
                       массива}
конец
```

В этом алгоритме мы впервые использовали комментарии , которые записываются в фигурных скобках и служат для пояснений. Если применить этот алгоритм к строке 'п2с' , представленной цепочкой , изображенной на рис.2.3 , при х='е' и к= 3 , то в



Рис. 2.3

Организация строки 'п2с' в виде цепочки

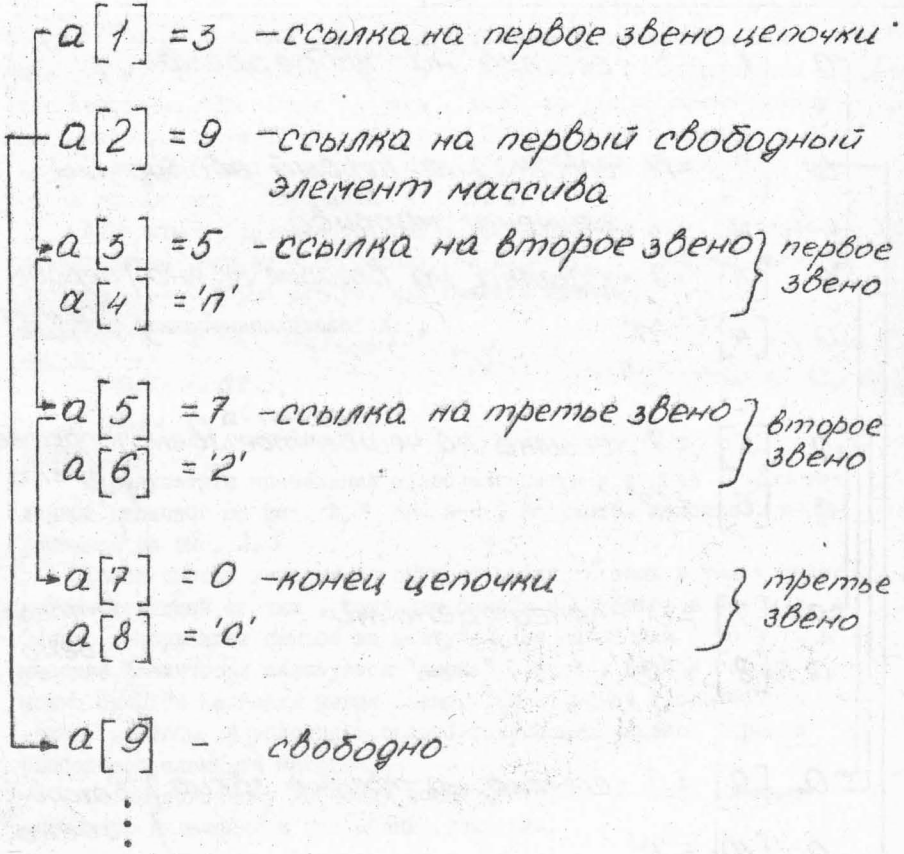
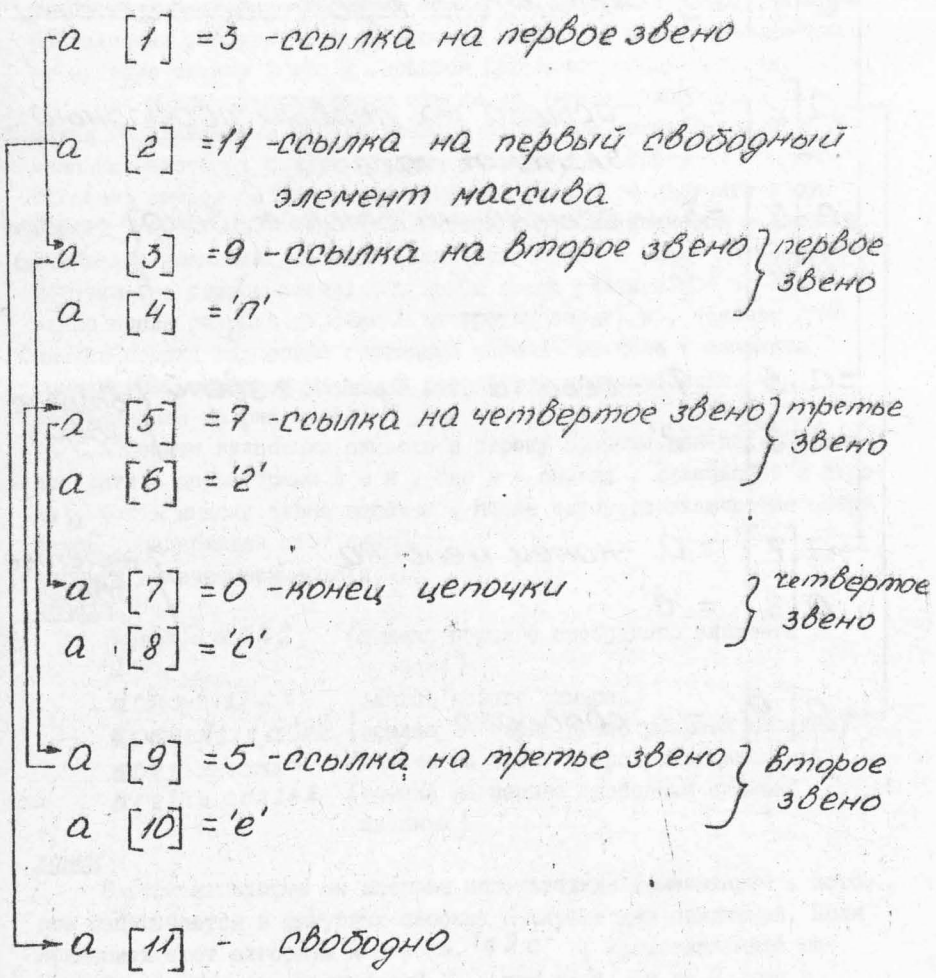


Рис. 2.4.

Организация строки 'пе2с' после включения символа 'е'






результате его исполнения получится цепочка , показанная на рис.2.4 .

Исключение символа из строки сводится к уничтожению в цепочке ссылки на исключаемый символ. Предположим , что из строки 'пе2с' необходимо исключить третий символ , то есть перейти к строке 'пес' . Цепочка , используемая для представления строки 'пе2с' и изображенная на рис.2.4 должна быть изменена так , чтобы в ней отсутствовала ссылка на символ '2' , содержащийся в третьем звене. Это можно сделать , задав во втором звене ссылку на символ , следующий за символом '2' , то есть - на символ 'с', иными словами нужно выполнить команду: a[9] :=7. После этого цепочка примет вид , показанный на рис. 2.5 .

Алгоритм исключения символа , оформленный в виде подчиненного алгоритма , имеет один формальный параметр к - индекс звена цепочки , за которым следует исключаемое звено:

```
алгоритм исключениесимвола (к) ;
начало
      ссылка:= a[к];
      a[к] := a[ссылка] .
конец
```

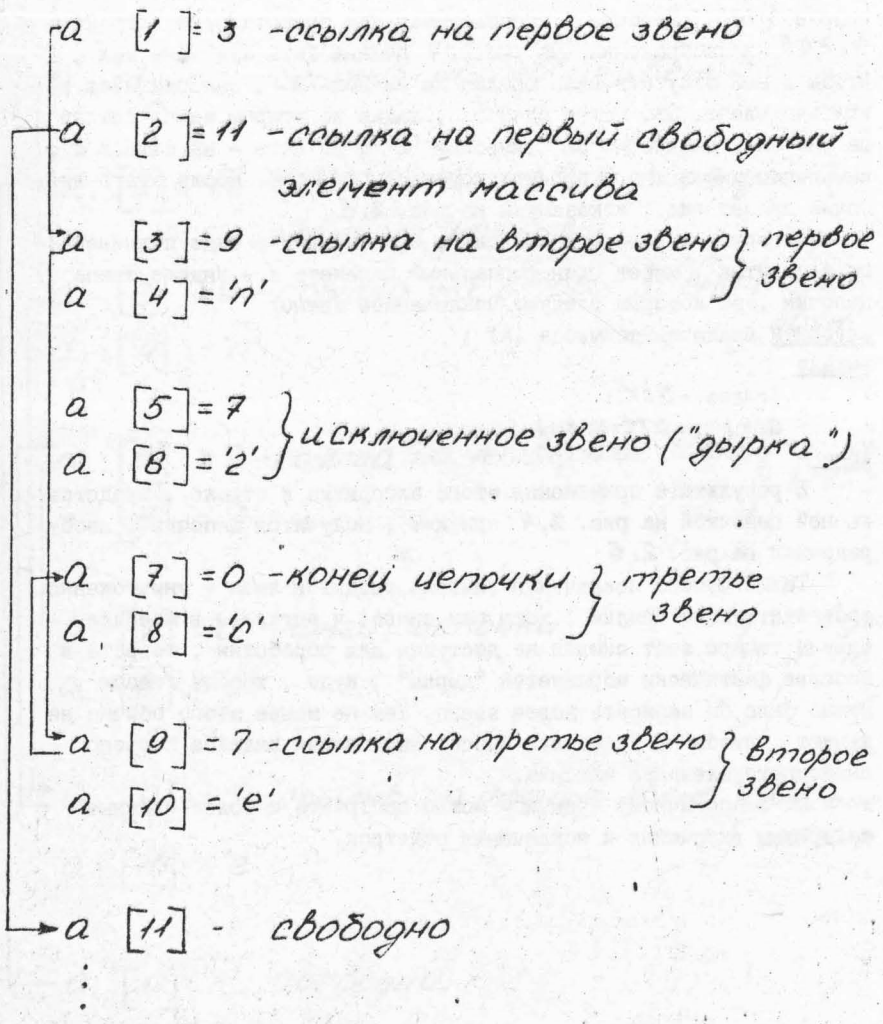
В результате применения этого алгоритма к строке , представленной цепочкой на рис. 2.4 при к=9 , получится цепочка , изображенная на рис. 2.5

Такой способ исключения символа сводится лишь к уничтожению соответствующей ссылки , хотя сам символ и остается в массиве. Однако теперь этот символ не доступен для обработки , то есть в массиве фактически образуется "дырка" , куда , вообще говоря , можно было бы записать новое звено. Тем не менее этого обычно не делают , чтобы не усложнять способ вычисления индекса первого свободного элемента массива.

По аналогичному принципу можно построить и более сложные алгоритмы включения и исключения подстрок.

Рис. 2.5.

Организация строки 'лес' после исключения символа '2'






## 2.6. Списки

Список - это упорядоченный набор записей. Зписи могут содержать различную информацию в зависимости от того , какие реальные объекты они описывают. Например , можно говорить о списке научных учреждений Академии наук , списке факультетов данного вуза , списке студентов данной группы и т.п.

Записи могут иметь разную длину , а их упорядоченность в списке задается в зависимости от назначения списка.

Отображение списка на одномерный массив производится по тому же принципу , что и отображение строк , то есть каждая запись отображается на несколько соседних элементов массива , образуя звено , а звенья сцепляются в цепочку с помощью ссылок.

Особенность отображения списка на массив состоит в том , что разные звенья могут занимать разное количество элементов массива, так как записи имеют разную длину. Поэтому в каждом звене различают информационную часть , на которую отображается содержательная часть записи , и справочную часть , в которой содержатся сведения о длине записи , ссылки на соседние звенья и другая справочная информация о записи.

Начальное звено содержит справочную информацию о расположении первого звена списка и о первом свободном элементе массива , куда можно заносить новые записи списка.

Если каждое звено содержит только одну ссылку на следующее звено списка , то такой список называют однонаправленным. Записи такого списка можно перебирать последовательно только в одном направлении так же , как и символы строки.

Однако часто бывает необходимо обеспечить возможность просмотра записей и в прямом и в обратном направлении. В этом случае каждое звено содержит две ссылки - на предыдущее и на следующее звено цепочки. Такие списки называют двунаправленными.

Если последнее звено цепочки содержит ссылку на первое звено , как на следующее , а первое звено ссылается на последнее, как на предыдущее , то такой список оказывается замкнутым по кольцу и его называют кольцевым.

Часто списки строят по иерархическому принципу. В этом случае записи списка могут иметь внутреннюю структуру , организованную также по принципу списка , то есть основной список оказы-

вается состоящим из записей - подсписков.

Таким образом сцепление звеньев списка с помощью ссылок и наличие справочной части в каждой записи позволяет создавать достаточно сложные структуры списков , пригодные для разных приложений.

Задачи включения и исключения записей списка благодаря наличию ссылок могут решаться по тому же принципу , что и задачи включения и исключения символов строки. При этом новая запись заносится в свободное место массива и производится изменение соответствующих ссылок , а исключаемая запись , хотя и остается в массиве , но все ссылки на нее уничтожаются , так что она становится недоступной для обработки.

Появление "дырок" в массиве в результате исключения записей списка порождает проблему включения освободившихся звеньев в число свободных элементов массива , которые можно использовать для размещения новых записей. Однако , как уже отмечалось , это усложняет задачу поиска первого свободного элемента массива.

Для включения "дырок" в число свободных элементов массива используют разные способы , известные под названием "сборка мусора". Суть этих методов сводится к периодической реорганизации списка и свободных звеньев , позволяющей создать единую цепочку свободных элементов массива , пригодную для размещения новых записей.

## 2.7. Таблицы

Список , как структура данных , мало пригоден для выдачи по запросу информации , хранящейся в произвольной записи , так как поиск записи в списке может производится лишь путем последовательного перебора звеньев списка в соответствии со ссылками. Поэтому для организации информационно-справочной службы используют структуры , называемые таблицами.

Таблица представляет собой набор записей , с каждой из которых связано имя , называемое ключом записи. Поиск нужной записи в таблице производится по ее ключу , в качестве которого может использоваться целое число без знака , а специальные способы организации таблицы позволяют осуществлять такой поиск за приемлемое время.



Над таблицами выполняются три основные операции: найти запись , включить новую запись и исключить запись из таблицы. Разные способы организации таблиц применяют с целью минимизации времени выполнения этих операций в зависимости от конкретных приложений.

При отображении таблицы на одномерный массив удобно использовать раздельный способ хранения ключей и информационных частей записей , что позволяет реализовать достаточно быстрые способы поиска записи. В этом случае таблица отображается на два одномерных массива: один из массивов $a[1], a[2], \ldots, a[n]$ используется для хранения ключевых записей , а другой массив $b[1], b[2], \ldots b[m]$ - для хранения информационных частей записей ( текстов).

Массив $a[1], a[2], \ldots, a[n]$ организован следующим образом: элемент $a[1]$ содержит индекс $i$ первого свободного элемента массива $a[i]$ , а , начиная с элемента $a[2]$, располагаются ключевые записи. Каждая ключевая запись занимает два соседних элемента массива - в первом элементе хранится ключ записи , а во втором элементе - ссылка $j$ на элемент массива $b[j]$, с которого начинается данная текстовая запись. Каждая текстовая запись содержит информацию о длине записи и текст самой записи.

Например , таблица может быть организована так:

$a[1]=78$ - ссылка на первый свободный элемент массива
$a[2]=132$ - ключ
$a[3]=3$ - ссылка на текстовую запись } -I-я ключевая запись
$a[4]=57$ - ключ
$a[5]=12$ - ссылка на текстовую запись } - 2-я ключевая запись

.
.
.

$a[78]$ -свободно

.
.
.

$b[1]=194$ - ссылка на первый свободный элемент массива


Раздельное хранение ключевых и текстовых записей позволяет располагать текстовые записи в массиве в произвольном порядке, так как их истинный порядок определяется порядком расположения ключевых записей с соответствующими ссылками. Это позволяет работать только с ключевыми записями, и производить перемещение записей при необходимости только в массиве $a$, что экономит время, так как ключевые записи, как правило, намного короче текстовых и их перемещение по элементам массива обходится дешевле. Изменения в массив $b$ приходится вносить при такой организации только в том случае, когда в таблицу добавляется новая запись, или производится " сборка мусора".

Поиск записи с заданным ключом в массиве ключевых записей, если эти записи никак не упорядочены, сводится к последовательному просмотру записей. Поиск может быть ускорен, если упорядочить записи в соответствии со значениями ключей - например, расположить записи в массиве в порядке увеличения ключей. В этом случае можно применить так называемый дихотомический поиск, идея которого сводится к следующему.

Предположим, что ключи расположены в последовательных элементах массива $c[1], c[2], \ldots, c[n]$ и упорядочены в порядке возрастания. Это не ограничивает общности, так как метод дихотомии легко обобщить и на случай, когда ключи в массиве следуют



через один элемент, как это было сделано выше.

Пусть требуется найти ключ K. Будем поступать следующим образом. Сравним этот ключ с ключом, хранящимся в среднем элементе массива. Для вычисления индекса этого элемента будем использовать операцию деления нацело (÷), то есть в качестве индекса среднего элемента массива будем брать целую часть $n/2$. Итак, пусть $i = n \div 2$ и $K_i = c[i]$ - значение ключа, хранящееся в этом элементе массива.

Сравним K с $K_i$. Если $K = K_i$, то поиск закончен, и нужный ключ найден. Если $K > K_i$, то ключ нужно искать во второй половине массива $c[i+1], c[i+2], \ldots, c[n]$, в противном случае его нужно искать среди элементов $c[1], c[2], \ldots, c[i-1]$. В выбранной половине массива снова берем средний элемент и т.д.

Каждое сравнение дает уменьшение длины массива, в котором производится поиск, в два раза, поэтому через $1 + \log_2 n$ сравнений нам останется сделать сравнение с единственным оставшимся элементом массива (если, конечно, мы не найдем ключ раньше). Это последнее сравнение и даст окончательный ответ на вопрос, есть ли данный ключ K в таблице.

В алгоритме дихотомии мы будем использовать переменные:
$i$ - для хранения индекса элемента массива, в котором содержится ключ K, $min$ и $max$ - для хранения индексов первого и последнего элементов части массива $c$, в которой ведется поиск и переменную $d$ - для хранения сообщения о том, найден ли ключ K.

```
алгоритм дихотомия ;
начало
    i := 0 ;
  min := 1 ;
  max := n ;
    d := 'ключ не найден' ;
  пока min ≤ max и i = 0
    повторять
      начало
          j := (min + max) ÷ 2 ;
        если K = c[j]
          то начало
                    d := 'ключ найден';
                    i := j
          конец
```

```
        иначе если  K > a[j]
                то  min := j+1
                иначе max := j-1
                все
        все
    конец;
    вывод (i, a)
конец
```

Отметим , что цикл обязательно закончится , так как либо будет найден ключ K ( в этом случае $i=j$ , то есть будет нарушено условие $i \neq 0$) , либо нарушится условие $min \leq max$ , так как после каждого сравнения либо значение $min$ увеличивается на 1 , либо значение $max$ уменьшается на 1.

При обнаружении ключа будет выдано сообщение 'ключ найден' и значение $i$ индекса элемента массива , в котором он содержится. Если же ключ не будет обнаружен , то будут выданы начальные значения $i$ и $a$ , то есть значение индекса $i=0$ и сообщение 'ключ не найден'.

Организация таблицы , описанная выше , удобна в том случае , когда таблица мало меняется. Если же включать и исключать записи приходится так же часто , как и искать нужную запись , то сразу становятся заметными недостатки такой организации. Действительно, при включении новой записи таблицу ключевых записей приходится раздвигать , а при исключении записи - сжимать , перемещая все записи с места разрыва таблицы по элементам массива , чтобы сохранить упорядоченность ключевых записей.

Если таблица подвержена частым изменениям , ее организовывают по принципу двоичного дерева. Схема одного из таких двоичных деревьев приведена на рис.2.6 . Двоичное дерево состоит из вершин , соединенных стрелками , указывающими направление обхода дерева. Вершина дерева , в которую не входит ни одна стрелка, называется его корнем , с него и начинается обход дерева. Каждая вершина содержит ключ. Упорядоченность ключей на дереве задается следующим образом. Левая стрелка , выходящая из вершины , указывает на вершину с меньшим ключом , правая стрелка - на вершину с большим ключом.



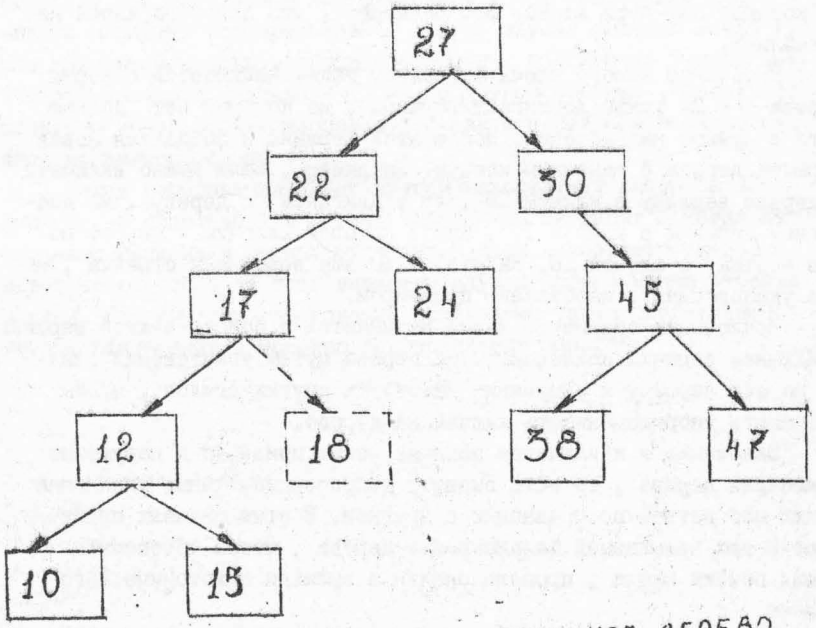



Рис. 2.6. ДВОИЧНОЕ ДЕРЕВО.

При организации таблицы ключевых записей в виде двоичного дерева время поиска , занесения и исключения записи соизмеримо со временем дихотомического поиска.

Поиск ключа начинается с корня дерева. Например , если производится поиск ключа К=24 , то из корня дерева , изображенного на рис. 2. 6 , придется двигаться по левой ветви ( 24< 27) , а из следующей вершины - по правой ветви (24 > 22) , после чего ключ будет найден. Если ключа нет в таблице , то мы дойдем до вершины , из которой нет дальше пути в нужном направлении. Например, при поиске ключа К=28 мы дойдем до вершины , содержащий ключ 30 , из которой нет пути влево. Это означает , что искомого ключа нет в таблице.

Включение нового ключа в таблицу также начинается с корня дерева , и мы также достигаем вершины , из которой нет дальше пути в нужном направлении. После этой вершины и создается новая вершина дерева с заданным ключом. Например , если нужно включить в дерево вершину с ключом 18 , то , двигаясь по дереву , мы достигнем вершины с ключом 17 , после которой включим в дерево новую вершину с ключом 18. На рис. 2.6 эта вершина и стрелка , на нее указывающая , изображены пунктиром.

Исключение заданного ключа начинается с поиска нужной вершины Найденная вершина исключается из дерева путем уничтожения ссылки на эту вершину и изменения некоторых других ссылок , чтобы сохранить упорядоченность ключей на дереве.

Включение и исключение записей может привести к нарушению симметрии дерева , то есть вызвать непропорциональные удлинение одних его ветвей по сравнению с другими. В этих случаях прибегают к так называемой балансировке дерева , чтобы обеспечить время поиска ключа , пропорциональное времени дихотомического поиска.

При отображении двоичного дерева на одномерный массив к двум элементам массива , соответствующим ключевой записи и содержащим ключ и ссылку на текстовую запись , добавляются еще два элемента , содержащие ссылки на следующие вершины дерева- левую и правую.

В описанных выше способах организации таблиц ключ записи никак не был связан с индексом элемента массива , на который о отображался. Существуют способы организации таблиц , использующие такую связь. Простейший способ состоит в том , чтобы значение ключа $K_i$ считать индексом $i$ соответствующего ему элемента массива. Однако в этом случае количество элементов массива , предназначенного для хранения ключей , равно значению наибольшего ключа. Пусть это значение $N$ . Если же фактически число различных ключей , хранимых в таблице - $n$ , причем $n << N$ , то в массиве будет много пустых мест , поэтому такой способ не рационален с точки зрения использования машинной памяти.

На практике для вычисления индекса элемента по ключу используют так называемые Хэш-функции , а преобразование ключа в индекс называют "хэшированием". В этом случае индекс $i = H(K_i)$, что дает возможность , с одной стороны , достаточно быстро вычислить значение индекса $i$ , а , с другой стороны , позволяет экономно использовать массив , так как его размеры выбираются лишь ненамного больше $n$ .

Метод хэширования имеет другой недостаток: какую бы Хэш - функцию ни использовать , может возникнуть ситуация , когда нескольким различным ключам будет поставлен в соответствие один и тот же индекс , то есть возможны случаи переполнения массива ключей. Поэтому при хэшировании используют также специальные методы размещения в массиве переполняющих записей.

РАЗДЕЛ 2. Основы вычислительной техники

ТЕМА 3. ОБЩИЕ СВЕДЕНИЯ ОБ ЭВМ

## 3.1. ЭВМ как исполнитель алгоритма

При выяснении понятия алгоритма мы отмечали, что формулировка и запись алгоритма должны учитывать особенности конкретного исполнителя, на которого рассчитан данный алгоритм. В этом смысле нас будут интересовать особенности электронных вычислительных машин как исполнителей алгоритма.

Решение сложных научно-технических задач требует от исполнителя достаточно большого быстродействия, чтобы исполнить алгоритм решения задачи за практически приемлемое время. Необходимость создания таких исполнителей алгоритма и привела к появлению в середине сороковых годов нашего столетия электронных вычислительных машин - автоматических исполнителей алгоритмов.

Автоматическое исполнение алгоритма обеспечивается, с одной стороны, принципиальной возможностью сформулировать алгоритм в виде, пригодном для исполнения автоматом, а, с другой стороны, принципиальной возможностью реализации абстрактной алгоритмической схемы в виде реальной физической машины.

Прежде всего заметим, что сам термин "электронные вычислительные машины" не совсем точно отражает существо дела.

Термин "электронные" отражает применяемую для построения ЭВМ техническую базу, а не принципы устройства таких машин. Действительно, современные ЭВМ построены на базе электронных элементов, но они вполне могли бы быть реализованы и на другой технической основе.

Так, в наше время существуют проекты вычислительных машин на электро-оптической основе, а проект первой в мире автоматической вычислительной машины Ч.Беббиджа (1883 г.), предвосхитивший многие идеи, лежащие в основе современных ЭВМ, предполагалось реализовать на чисто механических принципах.

Термин "вычислительная" машина появился в связи с тем, что первые ЭВМ действительно использовались для решения задач вычислительного характера. Однако очень скоро выяснилось, что с помощью ЭВМ можно решать самые разнообразные задачи и невычислительного характера, например, управлять производством, переводить с одного



языка на другой, организовывать информационно-справочную службу и т.п.

В этой связи термин "автоматическая машина для обработки информации" больше соответствовал бы принципам построения и использования подобных машин. Действительно, в самом широком смысле ЭВМ - это машины, осуществляющие автоматическую обработку информации. Решение любой задачи с помощью ЭВМ сводится к обработке информации - исходные данные задачи преобразуются в результаты.

Несмотря на это мы не будем "изобретать" новый термин, а в дальнейшем будем пользоваться установившимся и общеупотребительным термином "ЭВМ". Заметим, что наряду с этим термином часто используют и термин "компьютер" от английского слова computer (вычислитель).

ЭВМ - это автоматические исполнители алгоритмов. Каждая ЭВМ выполняет заданную ей программу автоматически без вмешательства человека.

ЭВМ является универсальным средством обработки информации. Это означает, что на одной и той же ЭВМ можно решать задачи разных классов. Для этого необходимо сформулировать соответствующий алгоритм и представить его в терминах системы команд данной ЭВМ, то есть задать программу работы ЭВМ. Изменение программы влечет и изменение решаемой задачи без какого-либо изменения состава или конфигурации физической аппаратуры ЭВМ.

ЭВМ значительно расширили класс решаемых задач - они сделали возможным решение таких задач, о которых нельзя было и помышлять до их появления. Например, решение многих задач атомной энергетики, исследования космического пространства, охраны окружающей среды и разумного использования природных ресурсов Земли немыслимо без использования ЭВМ. В этом смысле ЭВМ являются важным фактором научно-технического прогресса в современном обществе.

## 3.2. Структура ЭВМ и принципы работы

Мир современных ЭВМ необычайно разнообразен, что отражает факт внедрения вычислительной техники в самые различные сферы деятельности людей. Сейчас принято делить электронные вычислительные машины на микро-ЭВМ, мини-ЭВМ, большие и сверхбольшие (супер) ЭВМ, причем каждый класс продолжает пополняться все новыми представителями. Развитие вычислительной техники происходит столь стремительно, что трудно провести четкую грань между разными классами ЭВМ. Можно сказать, что в целом в результате совершенствования технологии изго-

товления ЭВМ происходит увеличение их мощности и уменьшение габаритов. Так, современные микро-ЭВМ обладают практически теми же возможностями, которыми обладали большие ЭВМ шестидесятых годов или мини-ЭВМ семидесятых годов нашего столетия.

Несмотря на разнообразие типов ЭВМ все они построены на основе некоторых общих принципов, поэтому в их логической структуре можно выделить главные устройства, входящие в состав любой ЭВМ. Знакомство с этими устройствами, их функциями и взаимосвязями поможет понять общие принципы построения и функционирования ЭВМ.

### 3.2.1. Память, процессор, ввод-вывод

Принципиально любая ЭВМ имеет два основных устройства, которые называют центральными устройствами машины. Это - основная память и процессор. Основная память хранит обрабатываемую информацию и программу обработки, а процессор производит обработку информации в соответствии с программой и управляет работой других устройств машины. Схема центральной части ЭВМ и взаимосвязь устройств показана на рис. 3.1.

Процесс решения любой задачи на ЭВМ разрабатывается на последовательность элементарных действий, которые может выполнить процессор. Это, например, операции выборки из памяти или записи информации в память, арифметические операции или операции сравнения и т.п. Каждое действие процессора определяется командой программы, хранящейся в памяти. На каждом шаге обработки информации процессор выбирает из памяти команду и данные, необходимые для ее выполнения, выполняет предписанное командой действие и записывает результат в память. Затем процессор переходит к выполнению следующей команды. Так продолжается до тех пор, пока из памяти не будет выбрана специальная команда, предписывающая процессору прекратить выполнение программы. Таким образом программа выполняется автоматически под управлением процессора.

Помимо центральных устройств современные ЭВМ характеризуются достаточно богатым набором периферийных устройств, или устройств ввода-вывода. Эти устройства обеспечивают связь человека с ЭВМ, служат внешним источником информации, попадающей в основную память, хранят большие объемы информации, значительно, превышающие объем основной памяти, позволяют представлять результаты обработки информации в удобной для дальнейшего использования форме в виде



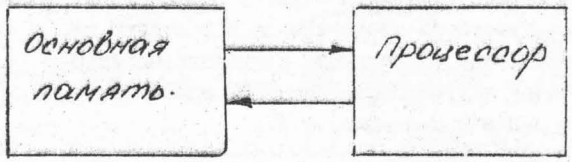



Рис. 3.1. Схема центральной части ЭВМ

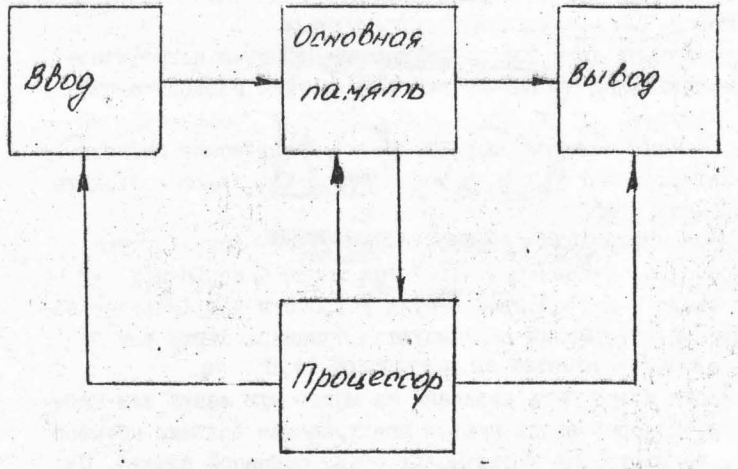



Рис. 3.2. Общая схема ЭВМ.

текстов, таблиц, графиков, изображений и т.п.

Данные передаются с устройств ввода-вывода в основную память и обратно под воздействием управляющих сигналов, поступающих из процессора. С учетом этого общая схема ЭВМ может быть изображена в виде, представленном на рис. 3.2.

Ниже перечисляются некоторые наиболее употребимые периферийные устройства современных ЭВМ и указывается их назначение.

Человек может ввести информацию в ЭВМ или вывести из ЭВМ результаты обработки с помощью терминала. В качестве терминала обычно используется дисплей, имеющий клавиатуру, подобную клавиатуре пишущей машинки, и экран электронно-лучевой трубки, подобной телевизионной трубке. Ввод информации производится с клавиатуры, при этом вводимая информация высвечивается на экране для визуального контроля. Информация, выводимая из ЭВМ также высвечивается на экране дисплея. Типичный дисплей показан ра рис. 3.3.

Дисплеи бывают двух типов: алфавитно-цифровые и графические. Первые позволяют вводить и выводить только алфавитно-цифровую информацию, а вторые кроме этого также и графическую информацию - графики, схемы, рисунки и т.п. Графические дисплеи кроме клавиатуры имеют также световое перо, позволяющее вводить и корректировать графическую информацию.

Для ввода информации в основную память ЭВМ используются и другие устройства, например, устройство ввода с перфокарт или устройство ввода с перфоленты. В этих устройствах в качестве носителей внешней информации используются бумажные карты или бумажная лента.

Информация может быть записана на магнитную ленту или магнитные диски, которые используются для хранения больших объемов информации, во много раз превышающих объем основной памяти. Однако эта информация не доступна для непосредственной обработки процессором. Что бы ее можно было обрабатывать, необходимо переписывать ее отдельными порциями в основную память. По этой причине память на магнитной ленте и магнитных дисках называют внешней памятью ЭВМ. Для чтения или записи информации в этом случае используются магнитофоны и дисководы. На рис. 3.4. и 3.5 показаны типичные магнитофон и дисковод ЭВМ.

Следует отметить, что в современных микро-ЭВМ в качестве магнитофона может использоваться также обычный бытовой кассетный



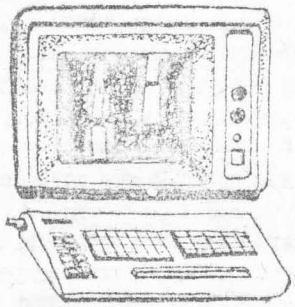


рис. 3.3. Дисплей

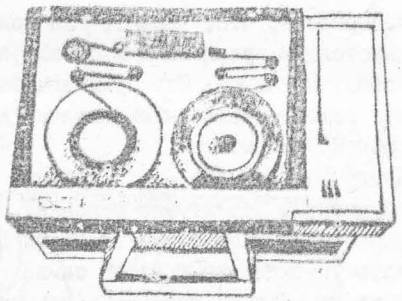


рис. 3.4. Накопитель на магнитной ленте

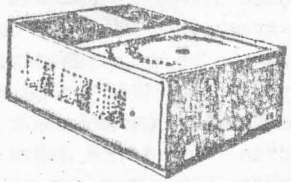


рис. 3.5. Накопитель на магнитных дисках

магнитофон, а для отображения информации - бытовой телевизор.

Для вывода информации используются печатающие устройства, с помощью которых алфавитно-цифровая информация отображается на бумажной ленте в виде "твердой" копии в отличии от "мягкой" копии, получаемой на экране дисплея. Для получения "твердой" копии на бумаге графической информации используются различные графопостроителя. Общие виды печатающего устройства и графопостроителя показаны на рис. 3.6. и 3.7.

Богатый набор периферийных устройств представляет человеку удобные средства общения с ЭВМ, причем с развитием техники этот набор быстро расширяется, например, сейчас появляются устройства для ввода в ЭВМ печатной информации, устройства ввода и вывода речевой информации и т.п.

### 3.2.2. Принципы фон Неймана

Выше уже говорилось, что, несмотря на большое разнообразие существующих в настоящее время ЭВМ, в основу их построения заложены некоторые общие принципы. Эти принципы были сформулированы в сороковых годах нашего столетия выдающимся американским математиком Джоном фон Нейманом.

Первый принцип фон Неймана - это принцип произвольной организации основной памяти. Структурно основная память ЭВМ состоит из дискретных элементов, называемых ячейками памяти. Каждая ячейка может содержать упорядоченный набор символов, который называют словом. Принцип произвольного доступа к памяти состоит в том, что процессору в любой момент времени доступна любая ячейка памяти.

Что-бы обеспечить такой доступ к ячейкам памяти, с каждой из них связывают персональное имя, и обращение к ячейке производится с помощью указания ее имени. Для этого все ячейки основной памяти перенумеровывают от 0 до $N - 1$, и в качестве имени ячейки используют ее порядковый номер, называемый также адресом ячейки. При этом общее число ячеек ($N$) называют объемом основной памяти.

Принцип произвольного доступа выделяет основную память среди других запоминающих устройств ЭВМ. Что-бы лучше понять этот принцип, сравним организацию основной памяти с организацией другого запоминающего устройства - магнитной ленты. Информацию, записанную на магнитной ленте, также можно разбить на элементарные единицы - машинные слова. При произвольном положении магнитной ленты доступно лишь то слово, которое записано на участке магнитной ленты,



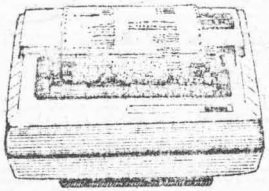


рис. 3.6. Печатающее устройство /принтер/

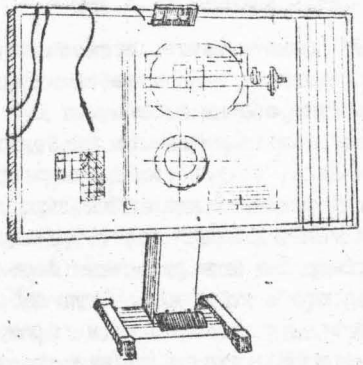


рис. 3.7. Графопостроитель

находящимся непосредственно под головками чтения-записи. Для чтения слова, записанного на другом участке магнитной ленты, необходимо предварительно переместить этот участок под блок головок, то есть доступ к этому слову можно получить лишь после просмотра предыдущих участков магнитной ленты. По этой причине магнитную ленту принято называть памятью с последовательным доступом.

Второй фундаментальный принцип фон Неймана - это принцип хранимой программы. Программа решения задачи хранится в основной памяти наряду с обрабатываемыми данными. Именно это делает ЭВМ универсальным средством обработки информации - для решения другой задачи требуется смена в основной памяти программы и обрабатываемых данных.

Информация, хранимая в основной памяти, не имеет признаков принадлежности к определенному типу. Поэтому ЭВМ, вообще говоря, не различает, что именно хранится в данной ячейке памяти - число, текст или команда. Это означает, в частности, что над командами программы могут выполняться такие же действия, как над данными. Так, команды одной программы могут быть получены как результаты выполнения другой программы. На этом принципе основаны методы трансляции программы с одного языка программирования на другой.

### 3.2.3. Представление информации в ЭВМ

Информация в ЭВМ представляется в двоичном виде, то есть для представления команд и данных используется алфавит, состоящий всего из двух символов, которые обычно обозначают 0 и I. Такое представление информации объясняется техническими особенностями реализации конкретных ЭВМ. В самом деле, технически гораздо проще реализовать, например, запоминающий элемент, имеющий только два устойчивых состояния, чем такой элемент, имеющий десять различных состояний. Аналогичное замечание относится и к различным переключательным схемам - элементы таких схем достаточно просто могут быть реализованы в виде переключателей, либо пропускающих, либо не пропускающих сигнал.

Любая информация в ЭВМ может быть представлена в виде последовательности двоичных символов. Такую информацию называют двоично-кодированной. Способ кодирования может быть разный, существо же кодирования состоит в том, что каждому символу алфавита, на котором представляется информация в ЭВМ, ставится в соответствие некоторый код, состоящий из определенного числа двоичных символов.

Например, можно использовать такой простейший способ кодирования: расположить все символы внешнего алфавита в определенном порядке и перенумеровать их, тогда каждому символу этого алфавита можно представить в соответствие двоичный код, начинающийся и кончающийся единицей и содержащий столько нулей между единицами, каков порядковый номер символа во внешнем алфавите. Очевидно, что это не оптимальный способ кодирования, так как с увеличением номера символа длина его двоичного кода будет увеличиваться, но он иллюстрирует принципиальную возможность представления произвольной информации в двоично-кодированном виде. На практике используются другие, более эффективные способы двоичного кодирования информации.



Машинное слово представляет упорядоченную последовательность двоичных символов. Каждый такой символ может иметь значение 0 и I. Такие единицы информации называют битами и говорят, что машинное слово содержит n битов. Длина слова является важной характеристикой ЭВМ.

В большинстве современных ЭВМ принято выделять наряду со словом другую стандартную единицу информации, содержащую 8 битов и называемую байтом. Слово машины обычно содержит кратное число байтов, например, 16-разрядные слова некоторых мини-ЭВМ содержат два байта, 32-разрядные слова больших ЭВМ содержат четыре байта и т.п.

Рассмотрим теперь общие принципы представления в ЭВМ команд и данных.

Текстовые данные представляются в виде последовательности двоично-кодированных символов и могут занимать несколько машинных слов. Двоичный код каждого символа обычно занимает один байт. Следовательно, текст, содержащий m символов, займет в памяти машины m последовательных байтов.

Число можно кодировать по аналогичному принципу. Действительно, каждую десятичную цифру можно представить в виде двоичного кода, при этом для кодирования каждой из десяти цифр достаточно четырех битов. Это значит, что в байте могут разместится двоичные коды двух десятичных цифр. Числа, представленные в таком виде, называют двоично-кодированными десятичными числами. Такое представление применяется в ряде ЭВМ, например, при обработке экономической информации.

Двоично-кодированное представление десятичных чисел не эффективно с точки зрения использования памяти ЭВМ. Поэтому в большинстве случаев для представления чисел в ЭВМ используется двоичная система счисления, и действия над числами выполняются именно в двоичной системе, то есть в ЭВМ реализуется двоичная арифметика.

При определенных плюсах в использовании памяти и скорости выполнения операций использование двоичной системы счисления порождает проблемы перевода чисел из десятичной системы в двоичную при вводе чисел в ЭВМ и обратного перевода при выводе чисел из ЭВМ. Это может оказаться довольно накладным, если приходится вводить и выводить большие массивы чисел при сравнительно небольшом времени их обработки. Именно поэтому часто при обработке экономической информации предпочитают использовать двоично-кодированные десятичные числа.

При использовании двоичной системы счисления для представления чисел внутри ЭВМ используются те же формы, что и в большинстве языков программирования для представления десятичных чисел. Это представление чисел с фиксированной и плавающей точкой. В первом случае число представляется в виде целой и дробной части, причем положение двоичной точки в числе фиксировано раз и навсегда. Кроме того, один двоичный разряд используется для представления знака числа (0 - для положительных чисел и 1 - для отрицательных). Во втором случае число представляется в виде $\pm m \cdot 2^{\pm p}$, причем в машинном слове, используемом для представления числа с плавающей точкой, в определенных разрядах записывается мантисса ($m$), а в других разрядах - порядок ($p$). Два разряда используются соответственно для представления знака числа и знака порядка.

Заметим, что из-за ограниченности разрядной сетки вещественные числа представляются в машине, вообще говоря, приближенно и операции над ними также выполняются приближенно.

Для представления целых чисел можно использовать форму с фиксированной десятичной точкой, когда эта точка фиксирована после младшего разряда числа, то есть дробная часть отсутствует. Целые числа представляются в ЭВМ точно и действия над ними также выполняются точно в пределах диапазона представимых в ЭВМ целых чисел. С этими вопросами мы еще встретимся при обсуждении основных типов данных, используемых в языках программирования.

Машинные команды представляются в двоично-кодированном виде и, как правило, занимает одно машинное слово, но некоторые современные ЭВМ имеют в своей системе команд команды разной длины. В представлении команды различают два основных поля - поле операции и поле адреса (адресов). В поле операции указывается двоичный код операции, которую предписывает выполнить данная команда, а в поле адреса - адреса данных, над которыми должна быть выполнена эта операция.



Каждая операция, которую может выполнить данная ЭВМ, имеет свой код, причем разные ЭВМ могут иногда значительно различаться наборами операций, которые они могут выполнять.

Адресная часть команды должна содержать адреса данных, над которыми выполняется операция и адрес памяти, куда должен быть отправлен полученный результат. Однако в реальных программах далеко не все команды используют три адреса. Поэтому стремление разработчиков ЭВМ эффектно использовать адресное поле команды привело к созданию ЭВМ с очень разнообразными системами команд - одноадресными, двухадресными, трехадресными и даже четырехадресными. В ряде современных ЭВМ в систему команд входят команды с разным числом адресов (переменно-адресные команды).

Все сказанное о машинных командах показывает насколько разнообразны системы команд реальных ЭВМ, то есть их машинные языки. Однако эти различия не являются принципиальными - основные правила и методы создания программ на машинном языке годятся для любой машины.

В настоящее время за машинным языком сохранилась роль внутреннего языка, на котором представляется в конечном счете выполняемая программа. Человек же ведет разработку программы на языке программирования более высокого уровня. Поэтому эти общие правила и методы создания машинных программ находят теперь свое отражение в принципах построения трансляторов - специальных программ, предназначенных для перевода программы с языка высокого уровня на машинный язык.

### 3.2.4. Принцип работы ЭВМ

Согласно второму принципу фон Неймана, информация, находящаяся в памяти ЭВМ, не имеет признака типовой принадлежности. Поэтому одно и то же машинное слово может быть интерпретировано и как число, и как команда. Действительное различие между командами программы и данными на стадии обработки информации достигается за счет информации о месте расположения программы в основной памяти.

Процессор ЭВМ состоит из двух основных устройств - устройства обработки команд, которое также называют устройством управления, и устройства обработки данных, которое также называют арифметико-логическим устройством. Общая схема ЭВМ, на которой показаны эти устройства и их связи с другими устройствами, приведена на рис.3.8.

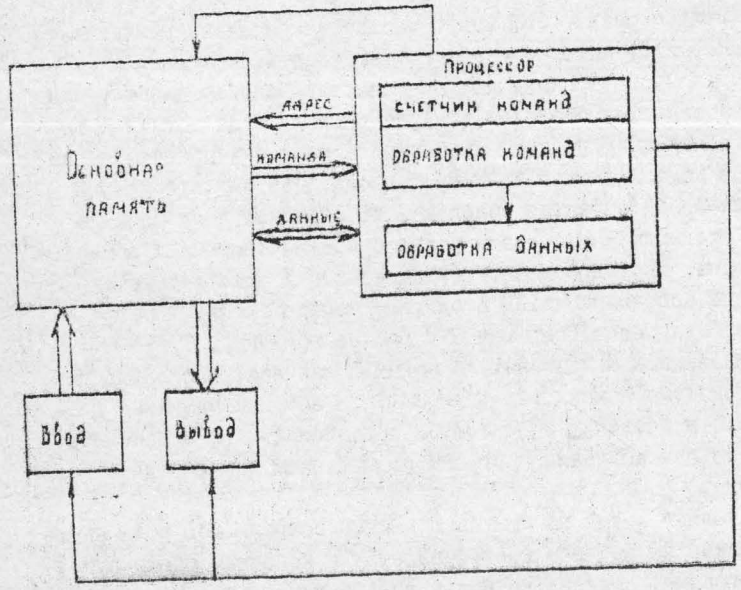



Рис. 3.8. Схема, поясняющая принципы работы ЭВМ.

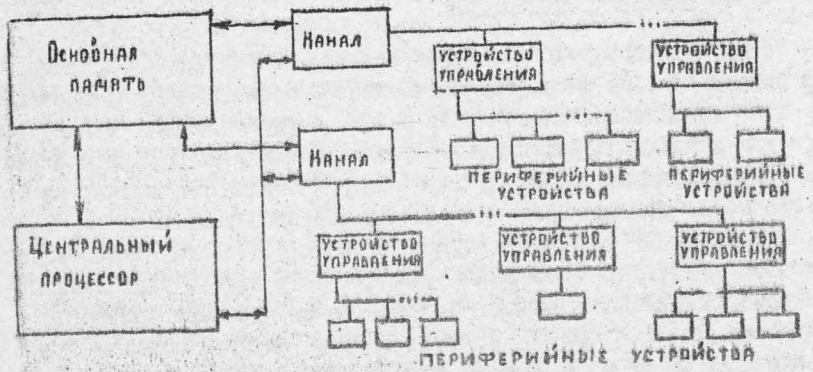



Рис. 3.9. Структура ЭВМ третьего поколения.



На этом рисунке двойными стрелками показана передача команд и данных, а обычными стрелками – передача управляющих сигналов.

Устройство обработки команд выбирает из основной памяти команды программы, обрабатывает эти команды и выдает управляющие сигналы в другие устройства.

Для выборки команд из основной памяти используется счетчик команд, содержащий адрес очередной команды. При первоначальной загрузке программы в основную память в этот счетчик записывается адрес той ячейки памяти, в которую загружена первая команда программы. Например, можно договориться, что программа всегда загружается в основную память, начиная с одной и той же ячейки. Тогда при загрузке программы в счетчик команд будет записываться адрес фиксированной ячейки.

Таким образом, первый шаг выполнения программы состоит в выборке первой команды программы из основной памяти по адресу, хранящемуся в счетчике команд, и передаче ее в устройство обработки команд.

Обработка команды производится в несколько этапов. Устройство обработки команд расшифровывает адресную часть команды, и по этому адресу (или адресам, если их несколько в команде) из памяти выбираются данные, которые пересылаются в устройство обработки данных.

После этого устройство обработки команд интерпретирует код операции и выдает в устройство обработки данных сигнал о том, какую операцию необходимо выполнить.

Устройство обработки данных, получив данные из памяти и сигнал из устройства обработки команд, выполняет предписываемую командой операцию. В зависимости от типа операции данные могут интерпретироваться как числа с фиксированной или плавающей точкой или как целые числа и т.п.

Результат операции либо остается в устройстве обработки данных, которое имеет собственную небольшую память, либо отправляется в основную память, если в команде был явно указан адрес результата.

Автоматическое выполнение программы обеспечивается тем, что после завершения выполнения очередной команды счетчик команд, как правило, увеличивается на единицу, чем обеспечивается выборка из основной памяти следующей команды программы.

Таким образом команды программы выполняются в естественном порядке. Однако при рассмотрении алгоритмов мы отмечали, что иногда приходится нарушать такой порядок выполнения команд. Для изме-

нения естественного порядка выполнения команд программы в системе команд любой ЭВМ предусмотрены специальные команды перехода. При выполнении такой команды в счетчик команд засылается адресная часть команды перехода, что обеспечивает выборку из памяти той команды, на которую указывает команда перехода.

Команда может предписывать ввести или вывести данные. В этом случае управляющий сигнал из устройства обработки команд поступает в устройство ввода или вывода, а данные в соответствии с предписанием команды либо передаются из устройства ввода в основную память, либо, наоборот, поступают из основной памяти в устройство вывода.

Выборка команд из основной памяти прекращается после выполнения специальной команды "стоп", предписывающей прекратить выполнение данной программы.

Выполнение программы производится автоматически без вмешательства человека. Человек принимает участие лишь на этапе загрузки программы. Здесь его роль сводится к установке носителя, содержащего программу, на соответствующее устройство ввода и запуске этого устройства с целью загрузки программы в основную память.

Ниже мы увидим, что развитие структуры ЭВМ привело и к развитию другого режима работы ЭВМ, когда человек работает в режиме диалога, непосредственно общаясь с ЭВМ в процессе выполнения программы.

## 3.3. Развитие структуры ЭВМ

Развитие электронных вычислительных машин за сравнительно короткий исторический период (сорок лет) происходило такими бурными темпами, что сейчас принято говорить уже о четырех поколениях ЭВМ и о проектах перспективных машин нового, пятого поколения. Таким образом, в среднем каждое десятилетие происходила смена одного поколения другим. Каждое новое поколение отличалось от предыдущего не только новой технической базой, но и новыми решениями, применяемыми при проектировании структуры или, как принято говорить, архитектуры ЭВМ.

Большое разнообразие существующих типов ЭВМ не позволяет коротко рассмотреть все особенности их архитектурных решений, поэтому ниже мы остановимся лишь на некоторых общих особенностях, присущих архитектуре ЭВМ разных поколений, и проследим основные тенденции ее развития.



### 3.3.1. Повышение эффективности работы аппаратуры ЭВМ

Развитие ЭВМ сопровождалось увеличением их производительности. Под производительностью ЭВМ обычно понимают число задач определенного класса, которое можно на ней решить за данное фиксированное время. Производительность связана с другой важной характеристикой ЭВМ - быстродействием, то есть средним числом операций, выполняемых машиной за единицу времени (например, число операций в секунду). Очевидно, чем больше быстродействие, тем больше и производительность, но эта связь не является простой пропорциональной зависимостью. Это объясняется тем, что быстродействие определяется, исходя из скорости работы центральных устройств ЭВМ, то есть ее процессора и основной памяти, а при решении реальных задач используются также периферийные устройства машины.

Уже в машинах первого поколения было замечено, что использование периферийных устройств в процессе выполнения программы приводит к простоям процессора, то есть к снижению эффективности его работы. Дело здесь в том, что существует определенный дисбаланс в скоростях работы различных устройств ЭВМ. Если центральные устройства построены на базе электронных (безынерционных) элементов, что и обеспечивает их высокое быстродействие, то периферийные устройства имеют в своем составе механические (инерционные) узлы, значительно ограничивающие скорость их работы. Например, скорость чтения информации с магнитной ленты ограничена скоростью механического перемещения ленты под блоком магнитных головок, скорость работы устройства ввода с перфокарт ограничена скоростью механического перемещения перфокарт и т.д.

Поэтому, если при выполнении программы требуется произвести ввод или вывод информации, то есть осуществить обмен данными между основной памятью и периферийным устройством, то процессор должен ожидать окончания обмена, то есть простаивать. А так как скорость обмена фактически определяется скоростью работы механических узлов периферийного оборудования, то время простоя оказывается непропорционально большим, если учесть какую работу мог бы проделать процессор за это время.

Простои процессора вызывались и самим режимом использования ЭВМ. На машинах первого поколения автор программы - программист мог работать непосредственно за пультом управления ЭВМ и имел возможность вмешиваться в ход выполнения программы: останавливать процессор, анализировать промежуточные результаты, вносить исправ-

ления в программу и т.п. Такая работа в режиме человек - машина, с одной стороны, создавала определенные удобства программисту, позволяя ему оперативно анализировать результаты и вносить необходимые изменения в программу, а с другой стороны, вызывала еще большие простои процессора и снижала эффективность его использования.

Стремление повысить эффективность работы процессора привело к двум важным последствиям.

Прежде всего программист был отстранен от ЭВМ, и программы стали выполняться в так называемом режиме операторского счета. В этом случае доступ к пульту управления ЭВМ имеет только один человек - оператор ЭВМ, роль которого сводится к запуску очередной программы и получению результатов с устройства вывода, которые затем передаются программисту. Это в определенной степени повысило эффективность работы центральных устройств ЭВМ, но программист лишался возможности оперативной работы с программой.

Однако полностью устранить влияние человеческого фактора на эффективность работы ЭВМ не удалось - оператор должен был выполнять целый ряд работ вручную, например, устанавливать на магнитофон нежную ленту, устанавливать на устройство чтения с перфокарт нужную колоду перфокарт, запускать очередную программу и т.д. При этом оператор мог и ошибиться. Все это вызвало непроизводительные простои процессора.

Следующий шаг в повышении эффективности работы процессора состоял в автономизации работы периферийных устройств. Эти устройства были снабжены собственными устройствами управления, которые, получив команду из процессора, выполняли дальнейшую работу по обмену данных с основной памятью самостоятельно, освободив тем самым процессор для выполнения другой работы.

Эта идея нашла свое логическое завершение в машинах третьего поколения, в которых вся работа по обмену данными между основной памятью и периферийными устройствами выполняется автономно с помощью каналов -специализированных процессоров ввода-вывода. В связи с появлением в архитектуре ЭВМ нескольких периферийных процессоров основной процессор стали называть центральным процессором ЭВМ.

Теперь схема выполнения ввода-вывода выглядит так:
центральный процессор выдает команду - заказ на обмен информацией с определенным периферийным устройством в соответствующий канал, который, получив этот заказ, начинает работать независимо от центрального процессора, выполняя собственную программу. Его работа состоит в запуске соответствующего периферийного устройства и выполнении обмена информацией между этим устройством и основной памятью.



Таким образом, периферийные устройства ЭВМ подключены к основной памяти через соответствующий канал и работают под его управлением. С учетом этого структура ЭВМ может быть изображена так, как это показано на рис.3.9. Каждая ЭВМ может иметь несколько каналов, а к каналам может быть подключено несколько периферийных устройств.

Такая структура ЭВМ позволяет совместить работу центрального процессора и канала - с момента выдачи заказа в канал центральный процессор и канал вместе с соответствующим периферийным устройством работают одновременно, выполняя каждый свою программу, что повышает эффективность использования аппаратуры ЭВМ.

Однако эффективное совмещение работы различных устройств при выполнении только одной программы удается получить далеко не всегда. Для этого необходимо так организовать выполнение программы, чтобы после команды ввода-вывода можно было продолжить выполнение следующих команд, т.е. распараллелить программу на несколько процессов, которые могут выполняться одновременно. Это невозможно, если, например, программа построена последовательно так, что команды, следующие за командой ввода, используют результаты ее выполнения. В этом случае центральный процессор вынужден ждать окончания ввода данных.

Следовательно, чтобы обеспечить эффективное совмещение работы различных устройств ЭВМ, необходимо выполнять не одну, а одновременно несколько программ. Таким образом, необходимость повышения эффективности работы аппаратуры ЭВМ логически приводит к идее мультипрограммной работы ЭВМ, то е ь одновременному выполнению нескольких программ.

### 3.3.2. Мультипрограммный режим работы ЭВМ

В мультипрограммном режиме работа ЭВМ организуется следующим образом. В основной памяти находится несколько программ, готовых к выполнению. Пусть таких программ будет три. Центральный процессор начинает выполнение одной из этих программ, например, первой. Ее выполнение продолжается до тех пор, пока не встретится команда ввода-вывода. В этом случае центральный процессор выдает заказ на обмен в соответствующий канал и освобождается для выполнения другой программы, например, второй. С этого момента центральный процессор выполняет вторую программу, а канал производит обмен для первой программы, в этом смысле две программы выполняются одновременно.

При выполнении второй программы также может встретиться команда ввода-вывода. В этом случае также выдается запрос в соответствующий канал, и вторая программа прерывается. Теперь центральный процессор должен решить, какую программу выполнять. Есть две возможности: вернуться к продолжению прерванной первой программы или перейти к выполнению третьей программы. Допустим, что к этому моменту обмен для первой программы еще не закончился, тогда центральный процессор переходит к выполнению третьей программы.

Временная диаграмма выполнения трех программ в мультипрограммном режиме по описанному выше принципу приведена на рис.3.10. На этой диаграмме на оси абсцисс отмечаются моменты времени $t_1, t_2, t_3, \ldots$, в которые центральный процессор переключается с выполнения одной программы на другую, а на оси ординат отмечены точки 1, 2 и 3, соответствующие этим программам. Работа центрального процессора изображается сплошной жирной линией, а работа канала - сплошной тонкой линией. Время, когда программа находится в состоянии ожидания, изображено на диаграмме пунктиром.

Из этой временной диаграммы видно, что при одновременном выполнении трех программ удается достичь достаточно эффективного совмещения работы центрального процессора и периферийного оборудования. Однако и здесь возможны случаи простоя центрального процессора. Действительно, в периоды от $t_1$ до $t_5$ и от $t_6$ до $t_7$ центральный процессор простаивает, так как в это время производится обмен для всех трех программ.

Простои центрального процессора при мультипрограммном режиме работы ЭВМ определяются как выбранной стратегией переключения центрального процессора с одной программы на другую, так и характером одновременно выполняемых программ. В самом деле, если каждая из трех одновременно выполняемых программ такова, что требует большого обмена и малого времени работы центрального процессора, то ясно, что при таком выборе программ возможны простои центрального процессора, что демонстрирует временная диаграмма, приведенная на рис.3.11.

Если же выбрать для одновременного выполнения программы таким образом, чтобы среди них была программа, требующая значительного времени работы центрального процессора и малого времени обмена, то на фоне этой программы можно выполнять другие программы, требующие большого обмена. При этом простои центрального процессора практически могут быть сведены на нет. Подобную ситуацию демонстрирует временная диаграмма, приведенная на рис.3.12.

Уже из приведенного выше весьма упрощенного описания идеи мульти-



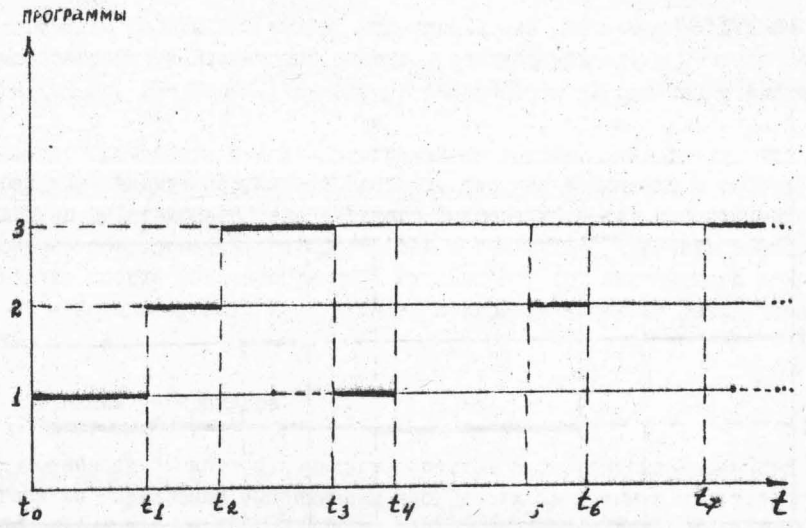



Рис. 3.10. Временная диаграмма одновременного выполнения трех программ.

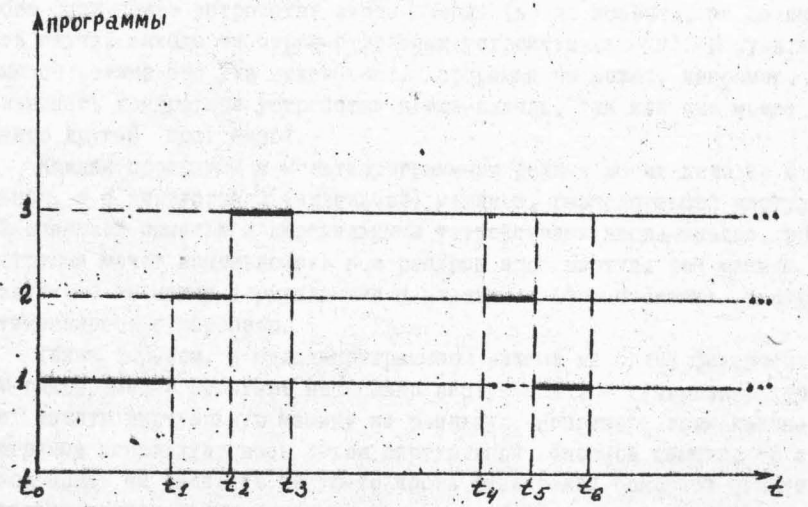



Рис. 3.11. Простои центрального процессора при неудачном выборе одновременно выполняемых программ.

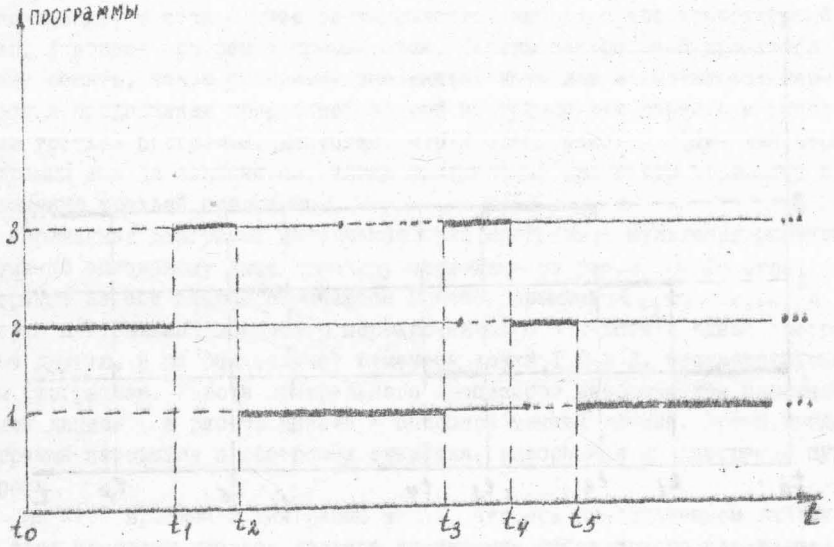



Рис. 3.12. Мультипрограммная работа при наличии фоновой программы 1.



программной работы ЭВМ видно, что управление одновременно выполняемыми программами достаточно сложно и реализовать его с помощью аппаратуры трудно, тем более, что для разных классов одновременно выполняемых программ может быть необходима разная стратегия переключения центрального процессора с одной программы на другую. Ясно также, что такое управление нельзя поручить оператору, так как включение в работу ЭВМ медленно действующего человеческого фактора приведет к значительным простоям оборудования. Выход состоит в том, чтобы поручить такое управление специальной программе – супервизору (от английского supervisor – надсмотрщик). Иногда такую программу называют также диспетчером.

## Функции супервизора

Основная функция супервизора состоит в реализации заданного алгоритма управления выполнением нескольких программ в мультипрограммном режиме работы ЭВМ. При этом супервизор может быть построен достаточно гибко, чтобы осуществлять различные стратегии управления при выполнении программ разных классов.

Действительно, в однопрограммном режиме все физические ресурсы ЭВМ находятся в полном распоряжении единственной выполняемой программы. Поэтому такая программа может использовать всю основную память и любое физическое устройство ввода-вывода (если, конечно, не рассматривать случаи выхода из строя отдельных устройств машины). В мультипрограммном режиме это уже невозможно. Программе не может, например, использовать конкретное устройство ввода-вывода, так как оно может быть занято другой программой.

Каждая программа в мультипрограммном режиме имеет дело не с реальной, а с виртуальной (идеальной) машиной, располагающей виртуальной основной памятью и виртуальными устройствами ввода-вывода, причем программе может использовать все ресурсы этой виртуальной машины. Соответствие же между виртуальными и реальными (физическими) ресурсами устанавливает супервизор.

Таким образом, в мультипрограммном режиме на одной физической ЭВМ одновременно работают несколько виртуальных, а супервизор отображает каждую виртуальную машину на реальную. Например, если какая-то программа использует весь объем виртуальной основной памяти, то супервизор может ей выделить какую-то часть физической основной памяти, а остальную виртуальную основную память отобразить на физическую внешнюю память ЭВМ, например, на магнитные диски. Управление выполнением такой

программы супервизор должен реализовать так, чтобы программа "не знала", где физически записаны данные. Для этого по мере необходимости супервизор (вызывает) в физическую основную память те данные, которые записаны на магнитных дисках.

Чтобы супервизор мог реализовать алгоритм управления программами в мультипрограммном режиме, к аппаратуре ЭВМ предъявляется целый ряд дополнительных требований по сравнению с требованиями к аппаратуре однопрограммной ЭВМ. Отметим основные из них.

Прерывания

При одновременном выполнении нескольких программ от центрального процессора требуется реакция на определенные события. Например, при появлении команды ввода-вывода требуется прервать выполняемую программу, окончание обмена также требует внимания центрального процессора, так как дает возможность продолжить прерванную программу и т.д. Таким образом, ЭВМ из детерминированного устройства, производящего обработку в соответствии с заранее составленной программой, все более превращается в автомат, реагирующий на внешние события.

Такая реакция обеспечивается включением в аппаратуру ЭВМ так называемого механизма прерываний, который обеспечивает (запоминание) причины прерывания, прерывает выполняемую программу и передает управление на выполнение программы - супервизора.

Управление прерываниями осуществляется с помощью регистра, имеющего столько разрядов, сколько возможно различных причин прерываний. Если возникает какое-то событие, требующее прерывания, то в соответствующий разряд регистра прерываний записывается I (если причины прерываний отсутствуют, то все разряды регистра прерываний содержат 0). Ясно, что одновременно может возникнуть несколько прерываний, поэтому все причины прерываний упорядочены в соответствии с их приоритетом, как и разряды регистра, и центральный процессор реагирует на прерывание с самым высоким приоритетом.

Реакция центрального процессора на появление единицы в регистре прерываний состоит в том, что выполняемая программа по завершении выполнения очередной команды прерывается, и производится обработка прерывания.

Обработка прерывания начинается с запоминания информации о прерванной программе, которая необходима для продолжения этой



программы, когда это будет необходимо. В мультипрограммном режиме с каждой программой связано слово состояния программы, содержащее необходимую информацию для ее выполнения. Это слово и запоминается в памяти, в так называемом информационном поле супервизора. Вместо этого слова выбирается из памяти новое слово состояния программы, соответствующее причине прерывания. Можно сказать, что каждой причине прерывания соответствует свое слово состояния программы. Новое слово состояния программы становится текущим, и это обеспечивает переход к выполнению супервизора.

Супервизор в зависимости от причины прерывания принимает решение о дальнейшей работе, например, в соответствии с принятым алгоритмом переходит к выполнению другой программы, что также обеспечивается сменой текущего слова состояния программы.

Помимо внешних прерываний возможны также прерывания, вызываемые внутренним состоянием центрального процессора, например, ошибкой при выполнении какой-то команды (деление на нуль, слишком большой результат, выходящий за пределы диапазона представимых чисел и т.п.). Такие прерывания называют внутренними. Обычно при возникновении такого рода прерываний супервизор прекращает выполнение программы, вызвавшей прерывание, и выдает об этом соответствующее сообщение.

Таким образом, обработка прерываний происходит в режиме супервизора, то есть при выполнении программы-супервизора. В этом режиме вновь появляющиеся прерывания хотя и запоминаются в регистре прерываний, но не вызывают прерывания супервизора. Говорят, что прерывания заблокированы. Это сделано для того, чтобы дать возможность супервизору закончить обработку текущего прерывания. Разблокировка прерываний происходит только после завершения обработки очередного прерывания, и тогда супервизор вновь получает возможность заняться обработкой других прерываний.

Привилегированный режим

Режим супервизора называют также привилегированным режимом. Причина такого названия состоит в следующем. В режиме супервизора могут выполняться некоторые команды, выполнение которых в режиме обычной программы запрещено. Такие команды называют привилегированными. Эти команды необходимы для управления работой программ в мультипрограммном режиме, но их использование при выполнении

обычной программы могло бы привести к нарушению выбранной стратегии управления.

Например, к числу привилегированных команд относятся команды записи в регистр прерываний. Ясно, что супервизор, начав обработку прерывания, должен погасить единицу в соответствующем разряде регистра прерываний, чтобы не потерять нового прерывания с той же причиной. Если же обычная программы будет иметь доступ к регистру прерываний, то это может привести к неправильному функционированию супервизора.

По этой причине различают два режима работы машины: обычный режим, когда выполняется обычная программа, и привилегированный режим, когда выполняется супервизор. В привилегированном режиме могут выполняться любые команды, имеющиеся в системе команд ЭВМ, а в обычном режиме - только непривилегированные команды. Попытка выполнить привилегированную команду в обычном режиме приводит к прерыванию программы, пытавшейся использовать такую команду.

Привилегированный режим устанавливается аппаратно, когда в результате прерывания передается управление на супервизор, а обычный режим устанавливает супервизор, когда после обработки прерывания он передает управление на обычную программу.

Защита памяти.

При мультипрограммном режиме работы ЭВМ в основной памяти машины находятся одновременно несколько программ вместе с обрабатываемыми данными. Это порождает проблему защиты памяти каждой такой программы от вмешательства другой программы. Такое вмешательство может случиться по причине ошибки, например, в результате ошибки в программе может появится команда записи данных в "чужую" область памяти, что приведет к нарушению правильности другой программы или ее данных.

Защита памяти осуществляется аппаратно, то есть в машине предусмотрен специальных механизм защиты, обеспечивающий прерывание программы, при попытке использования "чужой" области памяти.

Принцип одного из возможных способов защиты основан на следующем. Выше мы уже говорили, что каждая программа работает в виртуальной памятью, а отображение этой виртуальной памяти на физическую память ЭВМ осуществляет супервизор, который закрепляет за каждой программой определенную область основной памяти машины.

В современных ЭВМ используется так называемая страничная организация памяти, когда вся физическая память делится на страницы, содержащие одинаковое число слов, например, 1024 слова. С каждой страницей связан свой регистр защиты, содержащий ключ защиты. Супервизор выделяет для каждой из одновременно выполняемых программ определенное число страниц памяти, и в регистр защиты каждой выделенной страницы заносит ключ, например, номер программы, которой выделена эта страница.



Теперь при выполнении программы при каждом обращении к странице памяти номер выполняемой программа аппаратно сравнивается с ключом, хранящимся в соответствующем регистре защиты. Если номер программы не совпадает с ключом защиты, то это свидетельствует о попытке программы обратиться к "чужой" области памяти, что и приводит к прерыванию программы.

Часы (таймер)

Что бы супервизор мог прерывать выполнение программы по истечении некоторого фиксированного интервала времени, в машине аппаратно реализуются часы, ведущие отсчет реального времени. Такие часы представляются в виде счетчика, на который с известным интервалом времени поступают импульсы от генератора тактовых импульсов, имеющегося в составе любой ЭВМ. Счетчик суммирует эти импульсы, то есть отсчитывает реальное время.

Супервизор может прочитать значение счетчика или измерить его значение с помощью специальных команд. Кроме того, через определенные интервалы времени со счетчика поступает сигнал прерывания, который также может быть использован супервизором, чтобы вести учет реального времени.

Часы могут быть использованы, например, для определения момента аварийного прерывания программы, если в результате ошибок ее выполнение длится слишком долго, сверх предусмотренного лимита времени.

Кроме того, часы обычно используются для ведения общего временного протокола выполнения программы, в котором указывается время использования центрального процессора и время, затраченное на выполнение ввода-вывода.

### 3.3.3. Развитие операционных систем

Как мы видели, стремление повысить эффективность работы аппаратуры ЭВМ привело к отстранению от машины программиста и использованию ЭВМ в режиме операторского счета. Что бы облегчить работу оператора и еще больше ограничить влияние человеческого

фактора на работу ЭВМ, уже на машинах первого поколения стали использовать специальные управляющие программы - так называемые пакетные мониторы.

Функция пакетного монитора сводилась к автоматическому управлению потоком программ, собранных в пакет. За оператором сохранилась роль формирователя пакета - он собирал пакет перфокарт, в который помещалось несколько программ, и производил запуск пакета. Чтобы можно было различить программы в пакете, к их оформлению предъявлялись теперь определенные требования: каждая программа должна была начинаться с управляющей карты, содержащей сведения о программе и о требуемых для ее выполнения ресурсах ЭВМ, и должна была заканчиваться специальной картой, содержащей признак конца программы.

После запуска пакета управление передавалось монитору, который управлял выполнением программ, помещенных в пакет. Это управление схематически может быть изображено следующим образом. Монитор осуществлял ввод с перфокарт очередной программы и передавал управление на ее начало. Выполнение каждой программы завершалось передачей управления монитору, который вводил следующую программу и т.д. Работа монитора завершалась после выполнения всех программ пакета, о чем монитор выдавал сообщение оператору.

В связи с тем, что машины первого поколения не обладали системой прерываний и защиты памяти, возможности пакетных мониторов были ограничены. Так, пакетный монитор не мог вмешиваться в процесс выполнения программы - прерывать ее или запрещать ей запись данных в какую-то область основной памяти. Поэтому неверно выполняющаяся программа могла испортить сам монитор или "зациклиться", то есть выполняться сверх отпущенного ей лимита времени. В этих случаях для восстановления правильного функционирования машины необходимо было вмешательство оператора.

Несмотря на указанные недостатки пакетные мониторы сыграли определенную положительную роль в увеличении эффективности работы ЭВМ. Именно с них началось развитие операционных систем - систем программ, предназначенных для управления потоком решаемых на ЭВМ задач и служащих для повышения эффективности работы машины.

С появлением прерываний и защиты памяти на машинах второго поколения появилась возможность существенно повысить эффективность использования оборудования машины путем организации мультипрограммного режима работы. Управление потоком программ и ресурсами ЭВМ



теперь производится с помощью супервизоров, которые имеют гораздо более широкие возможности по сравнению с пакетными мониторами.

Постепенно на супервизоры помимо их основных функций, рассмотренных в п.3.3.2., стали возлагать все новые функции: выполнение набора директив, то есть значительно более сложных команд по сравнению с теми, которые входят в систему команд, учет всех решаемых на ЭВМ задач, осуществление диагностики правильности работы машинного оборудования, управление файлами, то есть организованными наборами данных, хранящихся на периферийных устройствах машины и т.д.

Для выполнения всех этих функций потребовалось создание целого комплекса программ, увязанных в единую систему, которую и стали называть операционной системой. Ядром операционной системы остается супервизор.

Операционная система является программным расширением аппаратуры ЭВМ и составляет такую же неотъемлемую часть любой машины, как и ее аппаратура. Функционирование любой современной ЭВМ просто невозможно без операционной системы. Кроме того, операционная система является посредником между человеком и ЭВМ, предоставляя человеку возможность иметь дело с виртуальной машиной.

Наиболее полное развитие операционные системы получили в машинах третьего и четвертого поколений, которые характеризуются наличием развитых операционных систем. Современные операционные системы помимо обеспечения эффективного использования машинного оборудования предоставляют пользователю широкий набор различных услуг, значительно облегчающих процесс подготовки и решения задач на ЭВМ. В связи с этим само понятие операционной системы трансформировалось с появлением машин третьего поколения. Теперь под понятием операционной системы объединяют и программы, управляющие потоком программ и ресурсами ЭВМ, и программы, предназначенные для автоматизации изготовления программ пользователя (так называемые системы программирования).

Изменился и сам принцип изготовления операционных систем. Дело в том, что современные ЭВМ строятся по модульному принципу. Это означает, что одна и та же модель ЭВМ может иметь разный набор модулей - разный объем основной памяти, состав устройств центрального процессора и состав периферийного оборудования в зависимости от конкретного использования этой модели.

По такому же модульному принципу строятся и операционные

системы: имеется некоторая исходная операционная система, содержащая полный набор возможных программных модулей, а каждая конкретная операционная система, предназначенная для конкретной модели ЭВМ, получается из этой исходной системы путем генерации, то есть отбора и настройки определенного подмножества модулей исходной системы в соответствии с составом оборудования конкретной ЭВМ.

### 3.4. Режимы использования ЭВМ

Выше мы познакомились с одним из режимов использования ЭВМ, который возник еще на машинах первого поколения. Этот режим называют режимом пакетной обработки программ. Он получил свое развитие на машинах второго и следующих поколений.

Схема пакетной обработки программ выглядит следующим образом. Оператор получает от пользователей их программы и формирует из них пакет. Каждая программа пакета имеет паспорт, в котором указывается специфика программы: время счета (ориентировочно), требуемые машинные ресурсы, объем и примерная частота ввода-вывода и т.п. Этот паспорт обычно является первой перфокартой программы и используется операционной системой при планировании работы.

Оператор помещает сформированный пакет в устройство чтения перфокарт и сообщает об этом операционной системе, введя специальную директиву с операторского пульта управления. Дальнейшая обработка пакета выполняется операционной системой. Прежде всего пакет программ вводится в машину и размещается во внешней памяти - на магнитных дисках или магнитной ленте.

Так как машина работает в мультипрограммном режиме, то следующая задача, которую решает операционная система - это выбор из пакета тех программ, которые будут выполняться одновременно. Мы уже видели, что от правильного выбора таких программ зависит эффективность использования оборудования машины (см.рис.3.II и 3.I2). Выбор программ для одновременного выполнения осуществляет специальная программа операционной системы - планировщик заданий. При этом планировщик пользуется информацией о характере программы, содержащейся в ее паспорте.

Выбранные для одновременного выполнения программы переписываются из пакета в основную память машины и выполняются под управлением супервизора. Затем планировщик планирует следующие програм-



мы для мультипрограммной обработки, и так происходит до тех пор, пока не будет исчерпан весь пакет. Чтобы при этом не происходило простоев оборудования, операционная система может выдать заблаговременно требование оператору установить новый пакет.

Таким образом, в процессе пакетной обработки между оператором и операционной системой устанавливается связь через посредство пульта оператора. Операционная система может выдавать на этот пульт требования оператору, например, установить на магнитофон нужную ленту, включить определенное периферийное устройство, установить на устройство чтения перфокарт новый пакет и т.п. Кроме того на этот же пульт операционная система выдает протокол обработки пакета, сведения о сбоях и отказах оборудования ЭВМ и другую служебную информацию.

Оператор с этого пульта может вводить сообщения операционной системе о выполнении ее требований и, в свою очередь, выдавать операционной системе директивы произвести определенные действия, например, прекратить выполнение определенной программы, исключить заданную программу из пакета, срочно выполнить одну из программ пакета и т.п.

В режиме пакетной обработки возможно установление приоритетов для выполнения тех или иных программ. Приоритет может быть задан в паспорте задачи. Кроме того, оператор может изменять приоритет программы, задавая со своего пульта новый приоритет. Наличие приоритета учитывает планировщик при планировании порядка выполнения программ пакета. Это позволяет в первую очередь обрабатывать программы с более высоким приоритетом.

Режим пакетной обработки программ позволяет достаточно эффективно использовать оборудование ЭВМ при одновременном выполнении нескольких программ. Однако, он создает определенные неудобства пользователям, чьи программы обрабатываются в этом режиме. Общение пользователя с ЭВМ происходит в таком режиме через посредство оператора - пользователь отдает оператору свою программу, а по окончании обработки пакета получает от оператора готовые результаты. Он лишен возможности оперативно вносить исправления в свою программу, так как не имеет непосредственного контакта с ЭВМ, как это было на самых первых машинах.

В связи с этим в машинах второго поколения наметился возврат к такому режиму работы, когда пользователь работает в режиме непосредственного диалога с ЭВМ. Но возврат к режиму диалога произо-

шел на качественно новом уровне – появился режим разделения времени. Этот режим использования ЭВМ получил свое развитие в машинах третьего и четвертого поколений, где он больше известен под названием режима мультидоступа или режима коллективного пользования.

В режиме коллективного пользования на ЭВМ также одновременно обрабатывается несколько программ, но теперь с каждой такой программой работает в режиме диалога пользователь. Это достигается за счет того, что к ЭВМ подключается несколько терминалов, в качестве которых обычно используются дисплеи, и каждый терминал отдается в распоряжение пользователя. Операционная система организует одновременное обслуживание всех пользователей, выделяя каждому терминалу определенный квант времени работы центрального процессора. На самом деле терминалы обслуживаются последовательно, но большое быстродействие современных ЭВМ позволяет обеспечить минимальную задержку в выполнении заказа, поступившего с каждого терминала. Поэтому у пользователя создается полная иллюзия того, что он один располагает всеми ресурсами машины.

Таким образом, в режиме коллективного пользования на одной физической ЭВМ также работает несколько виртуальных, каждая из которых управляется своим терминалом. Это позволяет достаточно эффективно использовать оборудование машины, так как возможные простои, вызванные медленной реакцией пользователя, теперь связаны не с физической, а лишь с виртуальной машиной, причем простои одной виртуальной ЭВМ восполняются работой других виртуальных машин.

Все-таки, в режиме коллективного пользования также возможны простои физического оборудования, когда одновременно простаивает несколько виртуальных машин. Чтобы восполнить эти простои, режим коллективного пользования иногда совмещают с режимом пакетной обработки, причем обработка пакета производится в те интервалы времени, когда простаивают виртуальные машины пользователей.

В настоящее время получил развитие еще один режим использования ЭВМ, называемый режимом реального времени. Он используется при организации справочно-информационной службы, в системах бронирования и продажи железнодорожных и авиационных билетов, в системах управления производственными процессами и т.д. В этом режиме одна из ЭВМ также работает со многими пользователями, обслуживая каждого из них за определенный интервал времени, не превышающий времени реакции человека или автомата (робота), если речь идет



об управлении реальным производственным процессом.

На первый взгляд может показаться, что режим реального времени ничем не отличается от режима коллективного пользования. Однако здесь есть четкое различие. В режиме коллективного пользования каждый пользователь работает со своей программой, выполняемой его виртуальной машиной, а в режиме реального времени все пользователи работают с одной и той же программой или одним и тем же набором программ. В режиме коллективного пользования работа ЭВМ должна быть построена так, чтобы создать наибольшие удобства каждому пользователю для работы с его программой при приемлемом времени ответа на его реакцию, а в режиме реального времени работа машины организуется так, чтобы обслужить наибольшее число пользователей. Поэтому в режиме коллективного пользования число терминалов не превышает одного-двух десятков (в наиболее мощных системах оно может достичь сотни), а в системах реального времени это число может достичь тысячи или даже нескольких тысяч.

Примером использования ЭВМ в режиме реального времени является созданная в нашей стране система "Сирена" по продаже авиационных билетов или аналогичная система "Экспресс" по продаже железнодорожных билетов.

Развитие систем коллективного пользования и реального времени связано, прежде всего, с созданием больших ЭВМ с очень высокой производительностью, эффективное использование которых можно обеспечить лишь при организации коллективного доступа. Такие ЭВМ через линии связи объединяются в мощные многомашинные комплексы, образуя сети ЭВМ. Создание сетей ЭВМ является важной тенденцией развития вычислительной техники и режимов ее использования.

Пользователь получает доступ к сети через терминал или терминальную станцию, имеющую выход в линию связи данной сети ЭВМ. Терминальная станция пользователя должна иметь в своем составе устройства, обеспечивающие связь с сетью, и устройства, необходимые для получения сообщений и результатов из сети и отображения их в виде, удобном для дальнейшего использования.

В качестве терминальной станции может использоваться мини- или микро-ЭВМ. В этом случае полученные из сети данные могут получить окончательную обработку на этой машине и отображение на ее периферийных устройствах.

Создание все более мощных микро-ЭВМ отражает вторую важную тенденцию развития вычислительной техники – персонализацию вычис-

лений. Развитие персональных компьютеров связано с массовым распространением микропроцессоров, начавшимся в семидесятых годах. Микропроцессор, который физически реализуется в виде интегральной схемы большого уровня интеграции /БИС/ на кристалле кремния размером обычно 6 X 6 мм, является сердцем любой персональной ЭВМ.

Сейчас считается, что распространение персональных компьютеров может служить толчком к действительно революционному перевороту в методах хозяйственной деятельности, в развитии образования, в подходе человека к организации своих личных дел, а, возможно, даже и в методах человеческого мышления.

## 3.5. Персональные ЭВМ

Персональные ЭВМ находят применение в различных сферах человеческой деятельности. Они получают широкое распространение в образовании - многие высшие учебные заведения используют их для обеспечения учебного процесса, а в связи с реформой общеобразовательной и профессиональной школы и необходимостью обеспечения всеобщей компьютерной грамотности происходит их внедрение и в общеобразовательную школу.

Персональная ЭВМ - это небольшая ЭВМ, основой которой является микропроцессор, другими словами - это микро-ЭВМ. Обычно к числу основных характеристик таких машин относят следующее:

1. Невысокая стоимость, делающая компьютер доступным для индивидуального пользования.
2. Машина рассчитана на разные категории пользователей, в том числе на таких пользователей, которые не имели никакого предшествующего опыта работы с вычислительной техникой.
3. Операционная система персональной ЭВМ обеспечивает удобные средства для диалога человек - ЭВМ.
4. Машина универсальна и обладает достаточной гибкостью, позволяющей расширять библиотеку программ для разнообразных применений.
5. Машина способна работать по крайней мере с одним языком программирования высокого уровня типа Бейсик, Фортран, Си, Паскаль.

Общий вид персональной ЭВМ показан на рис. 3.13.



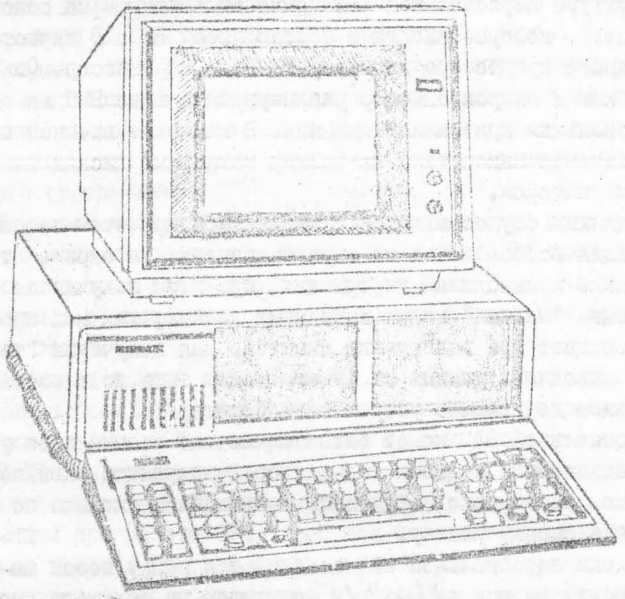


рис. 3.13. Общий вид персональной ЭВМ

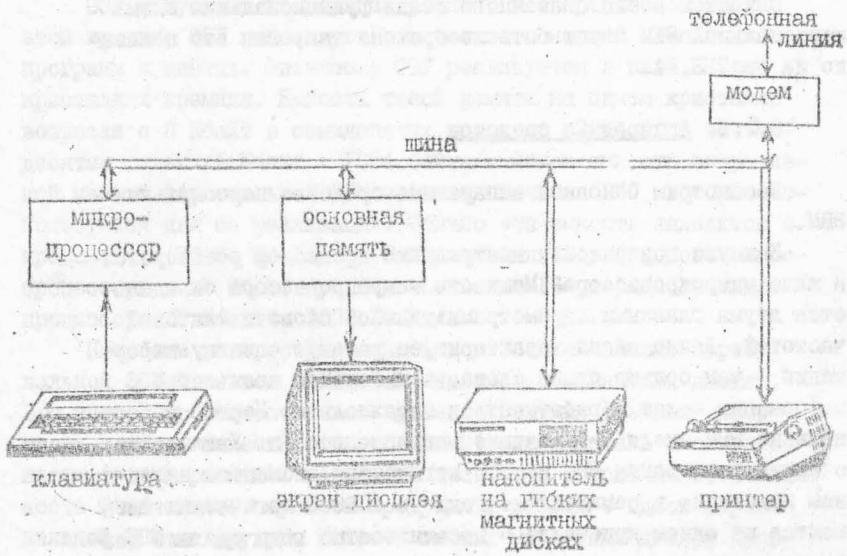



рис. 3.14. Общая функциональная схема персональной ЭВМ

В структуре персональной ЭВМ можно выделить те же основные устройства, которые входят в состав любой ЭВМ. В качестве центрального процессора используется микропроцессор. Основная память как и микропроцессор реализуется в виде БИС на одном или нескольких кристаллах кремния. В качестве внешней памяти используется накопитель на гибких магнитных дисках или кассетный магнитофон.

Ввод данных осуществляется с клавиатуры, при этом вводимая информация отображается на экране дисплея. На экране отображается также и выводимая информация, при этом получается так называемая "мягкая" копия выводимых данных, так как изображение пропадает при выключении дисплея. Для получения "твердой" копии выводимых данных на бумаге может быть использовано портативное печатающее устройство /принтер/.

К персональной ЭВМ может быть подключено специальное устройство, называемое модемом /модулятор-демодулятор сигналов/, с помощью которого можно передавать и принимать сигналы по обычному телефонному каналу.

Все блоки персональной ЭВМ соединяются между собой шиной /английский термин - "bus" /, состоящей из большого числа сигнальных линий.

С учетом всего сказанного общая функциональная схема персональной ЭВМ может быть изображена так, как это показано на рис. 3.14.

### 3.5.1. Аппаратные средства

Рассмотрим основные аппаратные средства персональных ЭВМ.

Как уже говорилось, центральный процессор реализуется в виде микропроцессора. Мощность микропроцессора определяется двумя главными параметрами: длиной слова и тактовой частотой. Длина слова характеризует рабочую единицу информации - чем больше длина слова, тем большее количество информации может обрабатываться параллельно. Первые персональные ЭВМ имели 8-разрядные микропроцессоры. Увеличение степени интеграции БИС с 5000 логических элементов на одном кристалле в семидесятые годы до 450000 логических элементов на одном кристалле в восьмидесятые годы сделало возможным увеличить длину слова. В начале восьмидесятых годов



появились 16-разрядные микропроцессоры, а в наше время начат серийный выпуск 32-разрядных микропроцессоров.

Создание более сложных БИС /которые теперь называют - СБИС - сверхбольшие интегральные схемы/ позволило не только увеличить длину слова, но и реализовать расширенную систему команд, что в свою очередь, облегчило создание программного обеспечения.

Тактовая частота определяет быстродействие микропроцессора, то есть скорость обработки информации. За прошедшее десятилетие тактовая частота возросла более, чем на порядок: с 0.5 мгц в семидесятых годах до 10 мгц в середине восьмидесятых годов.

Основная память персональной ЭВМ бывает двух видов - постоянная /или ПЗУ - постоянное запоминающее устройство/ и оперативная /или ОЗУ - оперативное запоминающее устройство/.

В ПЗУ информация заносится на заводе - изготовителе и в дальнейшем остается неизменной. В ПЗУ невозможна запись информации при работе ЭВМ, этот вид памяти допускает только операцию чтения. По этой причине в ПЗУ удобно хранить программы, которые разработаны раз и навсегда и остаются неизменными, например, составные части операционной системы.

ОЗУ допускает и операцию чтения, и операцию записи. По этой причине ОЗУ используют в качестве основной памяти для программ и данных. Физически ОЗУ реализуется в виде БИС на кристаллах кремния. Емкость такой памяти на одном кристалле возросла с 8 Кбайт в семидесятых годах до 32 Кбайт в восьмидесятых годах /1 Кбайт = 1024 байта/. Общая емкость оперативной памяти персональной ЭВМ зависит от числа БИС, которые используются для ее реализации. Обычно эта емкость находится в пределах от 32-64 Кбайт при использовании 8-разрядного микропроцессора до 1 Мбайт при использовании 16-разрядного микропроцессора /1 Мбайт = 1024 Кбайт/.

Наиболее распространенным видом внешней памяти персональной ЭВМ является накопитель на гибких магнитных дисках. Это устройство обеспечивает хранение больших объемов информации и имеет механизм, позволяющий устанавливать и снимать носитель информации - магнитный диск, что обеспечивает возможность физического переноса программ и данных с одной персональной ЭВМ на другую. Емкость гибких магнитных дисков находится в пределах 125 - 500 Кбайт, однако сейчас уже появля-

ются гибкие магнитные диски повышенной емкости – до 3 Мбайт.

Помимо гибких дисков в персональных ЭВМ используются также жесткие /винчестерские/ магнитные диски, имеющие емкость от 30 до 50 Мбайт. Однако жесткие диски не являются съемными – их невозможно переносить с одной ЭВМ на другую. Кроме того жесткие магнитные диски очень чувствительны к пыли, поэтому их обычно размещают в герметизированном корпусе.

Перспективным устройством внешней памяти персональных ЭВМ обещают стать оптические диски, на которых запись и чтение информации обеспечивается с помощью лазера. Эти диски при тех же размерах, что и гибкие магнитные диски, могут иметь емкость до 200 Мбайт.

Стандартным устройством ввода персональной ЭВМ является клавиатура. Обычно клавиатура помимо символьных клавиш содержит также так называемые функциональные клавиши, служащие для ввода некоторых часто употребляемых команд /например, PRINT или READ / и, тем самым, для повышения скорости ввода информации. Клавиатура персональных ЭВМ может быть неотъемлемой частью конструкции или съемным блоком, подключаемым к ЭВМ с помощью кабеля.

В качестве основного устройства отображения данных во всех персональных ЭВМ служит обычный телевизионный экран или специальный монитор. Для хранения информации, отображаемой на экране дисплея, используется либо часть основной памяти, либо специальная память дисплея.

Для получения "твердой" копии выводимых данных используется точечно – матричное печатающее устройство, обеспечивающее высокое /почти типографское/ качество печатаемого текста. Специальный вид точечно – матричных печатающих устройств обеспечивает не только печать буквенно – цифровых символов, но и построение графических изображений с достаточно высокой разрешающей способностью /4 точки на мм/. В перспективе в качестве печатающих устройств персональных ЭВМ очевидно будут использоваться лазерные печатающие устройства, обеспечивающие типографское качество выводимого текста и почти фотографическое качество графических изображений.

Модем делает возможным подключение персональной ЭВМ через телефонный канал к сети ЭВМ. В этом случае персональная ЭВМ получает доступ к информации, хранящейся в мощных ЭВМ, а сама персональная ЭВМ превращается в интеллектуальный терминал, на котором осуществляется обработка данных, полученных из сети, и отображение результатов обработки.



### 3.5.2. Программные средства

Программные средства составляют вторую составную компоненту персональной ЭВМ, без которой невозможно функционирование машины. Основу этих программных средств составляет операционная система, выполняющая следующие функции:

- управление аппаратными средствами: микропроцессором, памятью, периферийными устройствами;
- управление файлами;
- планирование заданий и управление вычислительными процессами;
- обеспечение удобств работы пользователя.

В отличие от больших ЭВМ, где одной из главных задач операционной системы является обеспечение эффективного использования ресурсов центрального процессора и режима коллективного пользования, в персональных ЭВМ обеспечение высокого коэффициента использования микропроцессора не является главной целью, так как эти машины ориентированы на индивидуальных пользователей. Главное назначение операционных систем персональных ЭВМ состоит в обеспечении максимальных удобств работы пользователю.

Фактически операционная система представляет собой иерархическую структуру уровней абстракции, устроенную таким образом, чтобы на каждом уровне можно было игнорировать подробности процессов, происходящих на более низких уровнях. Самый верхний уровень – это уровень пользователя, который таким образом может игнорировать все реальные физические процессы, происходящие в машине, а реагировать только на ответную реакцию компьютера на его команды. При этом, конечно, реакция ЭВМ на команды пользователя должна представляться в терминах, адекватных тем, в которых выражаются запросы пользователя на выполнение компьютером определенных заданий.

Ядро операционной системы составляют программы управления аппаратными средствами, планировщик заданий и программа управления файлами. Все остальные программные средства могут

быть разбиты на три основные части: языки программирования и соответствующие интерпретаторы, компиляторы и ассемблеры, служебные программы и прикладные программы. Общая структура программного обеспечения персональных ЭВМ изображена на рис. 3.15.

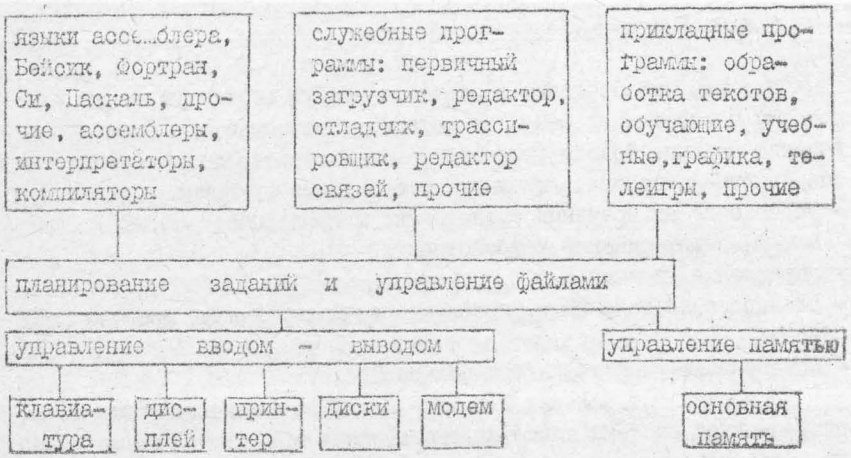



рис. 3.15. Общая структура программного обеспечения персональных ЭВМ

На сегодняшний день основными языками программирования персональных ЭВМ являются Бейсик, Паскаль, Си, Фортран и Ассемблер. Бейсик наиболее прост в освоении и по этой причине остается одним из самых распространенных языков программирования на персональных компьютерах. Паскаль и Си сложнее в освоении, однако они удобны для реализации структурного программирования и документирования программ, при этом Паскаль предпочтительнее для прикладного программирования, а Си - для системного. Фортран остался на персональных ЭВМ в наследство от больших и малых ЭВМ. Ассемблер обеспечивает создание наиболее эффективных программ и непосредственное управление устройствами ввода - вывод.

В состав служебных программ входят программы, обеспечивающие режим первоначальной загрузки, без чего невозможно обеспечить начало работы компьютера по ле его включения, а



также целый ряд программ, обеспечивающих удобство работы пользователя.

Состав прикладных программ в значительной степени определяется областью применения персонального компьютера. Например, в последние годы значительно возрастает популярность обработки с помощью персональных ЭВМ текстовой и графической информации. В этой связи получают быстрое развитие прикладные программы, предназначенные для обработки текстовой и графической информации.

Специфические требования области применения персональных ЭВМ находят отражение не только на составе прикладного программного обеспечения, но и на характеристиках других частей программного обеспечения. Так, использование персональных ЭВМ в сфере обучения предъявляет повышенные требования к обеспечению удобного диалога обучаемого с ЭВМ, к библиотеке учебных программ, к составу и качеству программ для автоматизированного самообучения. Кро. того программное обеспечение персональных ЭВМ, используемых для обучения, должно обеспечивать возможность объединения нескольких компьютеров в локальную сеть в пределах данной аудитории. Наконец, повышенные требования предъявляются и к аппаратуре таких ЭВМ - она должна быть достаточно прочной, чтобы уцелеть в условиях интенсивного использования самыми разными обучаемыми.

## ЛИТЕРАТУРА


1. Батанова В.Е., Жданов С.А., Кузнецов Э.И., Монза А.А., Реутова С.К., Сулейменова Б.Б. Методические разработки по курсу "Основы информатики и вычислительной техники" /машинный вариант/. Раздел: основы программирования. -М.: МГПИ им. В.И.Ленина, 1986.
2. Программа курса основ информатики и вычислительной техники /проект/ IX - X классы /102 ч./. Научный редактор академик А.П.Ершов. -М.: Ротапринт НИИ СиМО АПН СССР, 1986.
3. Основы информатики и вычислительной техники. Пробное учебное пособие для средних учебных заведений: часть I. Под редакцией А.П.Ершова и В.М.Монахова. -М.: Просвещение, 1985.
4. Жданов С.А., Ионов Г.Н., Кузнецов Э.И. Работа с программируемым калькулятором типа МК-54 /МК-56/. Учебно-методические материалы для учителей математики шестого класса общеобразовательной школы. -М.: МГПИ им. В.И.Ленина, 1985.
5. Батанова В.Е., Жданов С.А., Ионов Г.Н., Кузнецов Э.И., Монза А.А., Сулейменова Б.Б. Основы информатики и вычислительной техники /машинный вариант/. Учебно-методические материалы для учителей информатики девятого класса общеобразовательной школы. -М.: МГПИ им. В.И.Ленина, 1985.
6. Любимский Э.З., Мартынюк В.В., Трифонов Н.П. Программирование. -М.: Наука, 1980.


Эдуард Иванович Кузнецов

Методические разработки по курсу "ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ" /машинный вариант/. Разделы: основы алгоритмизации и основы вычислительной техники


Подп. в печ. 24.03.87. Формат 60x90/16 Бум. тип. № 3
Усл. печ. л. 5 Уч.-изд. л. 5 Печать офсетная
Тираж 2000 экз. Заказ 701 Цена 50 к.

Московский государственный педагогический институт имени В.И.Ленина
113882, Москва, Малая Пироговская ул., д.1
Типография МГПИ им. В.И.Ленина
129243, Москва, ул. Кибальчича, д.6