Министерство Высшего и Среднего специального
Образования Республики Узбекистан

М.М. Алиев, Р.И. Ибрагимов

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ ЖЕЛЕЗНОДОРОЖНОЙ АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

Учебное пособие

Ташкент-2007

М.М. Алиев, Р.И. Ибрагимов

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ ЖЕЛЕЗНОДОРОЖНОЙ АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

Учебное пособие

Утверждено МВ и ССО Республики Узбекистан от 24.07.2006 г. приказ № 167 в качестве учебного пособия для студентов высших технических учебных заведений

Ташкент-2007

УДК 681.31.

Алиев М.М., Ибрагимов Р.И. Программное обеспечение систем автоматики и телемеханики на железнодорожном транспорте. 2006. 142 с.

В данном учебном пособии рассмотрены вопросы прикладного программного обеспечения систем железнодорожной автоматики и телемеханики, алгоритмизация задач, выбор языка программирования, методы разработки программ. Приводится пример составления программы для небольшой станции.

Учебное пособие предназначено для студентов, обучающихся по направлению бакалавриата 5521800 – автоматизация и управление, а также для специалистов в области систем железнодорожной автоматики и телемеханики.

Ил. 26. Библиограф. 8 назв.

Рецензенты: А.Р.Азизов - к.т.н., доцент кафедры «Автоматика и телемеханика на ж.д. транспорте»
А.Ш. Мухамадиев – к.ф.-м.н., доцент кафедры «Информатика» ТГИК

ВВЕДЕНИЕ

Программное обеспечение является необходимой составной частью любой ЭВМ. Без соответствующих программ практически невозможно заставить машину сделать что-либо полезное. В состав программного обеспечения персональных компьютеров входят как универсальные средства, так и прикладные программы, ориентированные на отдельные проблемные области.

В настоящее время для разных типов ПК разработано несколько десятков тысяч программ, которые могут быть разделены на следующие основные классы:

- операционные системы и сервисные программы,
- инструментальные языки и системы программирования,
- прикладные системы.

Операционные системы дополняют аппаратные средства любого персонального компьютера, позволяя прикладным программам обращаться к внешним устройствам, а человеку – пользователю ПК – управлять работой машины с помощью соответствующих команд. Ядро ОС обычно дополняется набором сервисных программ, которые служат разным целям; с их помощью производится начальная разметка дисков, установка параметров внешних устройств, тестирование оперативной памяти и других устройств, выдача информации на печать, стыковка с большой машиной или локальной сетью и др.

Инструментальные языки и системы программирования – это особая категория программных средств. С их помощью создаются все другие программы; таким образом, они занимают в информатике особое место. Существует широкая номенклатура языков программирования, каждый из которых характеризуется определенными свойствами. Одни программисты предпочитают пользоваться каким-либо одним языком, другие, в зависимости от типа создаваемой системы выбирают из имеющегося набора наиболее подходящий язык программирования или пользуются сочетанием разных языков.

К категории инструментальных средств относятся не только трансляторы с языков высокого уровня, таких как Basic, Pascal, Turbo Pascal, Borland Pascal Delphi, C++ , но и ассемблеры, загрузчики, отладчики и другие системные программы. С помощью инструментальных средств создается и прикладное программное обеспечение, и новые средства системного программирования,

включая трансляторы с языков высокого уровня. Следовательно, эта категория программных средств совершенно аналогична средствам производства в промышленности, – таким как станки, инструменты, средства переработки сырья в нужную форму. При этом роль сырья играет информация – текстовые и числовые данные, закодированные сообщения, графические изображения. Такая же информация является и результатом работы любой программной системы.

Прикладные системы могут иметь общий характер, например, обеспечивать составление документов и их печать или хранение и выдачу справок. Другие классы прикладных систем ориентируются на автоматизацию конкретных видов деятельности, например, обучение определенным предметам в институте, проектирование электронных изделий, анализ электрокардиограмм, проведение финансовых расчетов, управление технологическими и производственными процессами и многое другое.

ГЛАВА 1. ОСНОВЫ ПРОГРАММИРОВАНИЯ ПРИКЛАДНЫХ ЗАДАЧ

1.1. Критерии выбора языка программирования

Разработка системного и прикладного программного обеспечения осуществляется с помощью инструментальных средств, к которым в первую очередь относятся:

- трансляторы с языков высокого уровня;
- средства редактирования, компоновки и загрузки программ;
- макроассемблеры (машинно-ориентированные языки);
- отладчики машинных программ.

С каждой операционной системой связывается относительно стандартный набор инструментальных средств. Различия имеются лишь в версиях языков высокого уровня и в форматах объектных и загрузочных модулей, получающихся после трансляции и компоновки программ.

Основные инструментальные языки высокого ур овня, используемые на персональных компьютерах – Бейсик, Паскаль, Си, Дельфи.

Когда возникает необходимость создания большой программной системы или составления программы для решения какой-либо

частной задачи, встает вопрос о выборе для этой цели наиболее подходящего языка программирования. Во многих случаях такой выбор допускается очень простыми «земными» факторами – допустимостью того или иного транслятора и умением составлять программы на данном языке. Если, однако, в распоряжении пользователя имеется достаточно большой выбор языков программирования, то следует учитывать следующие обстоятельства:

- назначение разрабатываемой программы - нужна ли она временно или будет использоваться постоянно, планируется ли передавать ее другим организациям, будут ли разрабатываться ее новые ресурсы;
- требуемая скорость работы программы, соотношение ее диалоговых и вычислительных компонентов;
- ожидаемый размер программы – можно ли будет ее создавать как единое целое или придется разбивать на отдельные взаимодействующие модули, требуется ли минимизировать размер памяти, занимаемой программой во время работы;
- необходимость сопряжения разрабатываемой программы с другими пакетами и программами, в том числе составленными на других языках программирования;
- предусматривается ли возможность переноса программы на другие типы ПК;
- основные типы данных, с которыми придется иметь дело, необходимость поддержки работы с действительными числами, строками, списками и другими типами структур;
- характер и уровень использования аппаратных средств – дисплея, клавиатуры и др., необходимость в специальном программировании некоторых функций для работы с внешними устройствами;
- возможность и целесообразность использования имеющихся стандартных библиотек подпрограмм, процедур, функций, модулей.

Так как целью изучения данного материала является использование одного из языков программирования для разработки программ по микропроцессорному управлению и контролю систем железнодорожной автоматики, телемеханики и связи, то к выбору языков программирования добавляются еще ряд требований.

Так как системы автоматики, телемеханики и связи обеспечивают безопасность движения поездов, то к ним предъявляются особые жесткие условия. Вероятность отказа в устройствах таких систем должна быть очень высокой, например, в

системах железнодорожной автоматики и телемеханики для включения на светофоре разрешающего огня проверяются ряд условий, и только при выполнении этих условий может быть включен разрешающий огонь, позволяющий поезду проследовать по участку, а каждое условие это отдельный самостоятельный модуль и при этом все модули взаимозависимы.

Основная программа составляется из процедур и модулей, и перед тем как выдать на исполнительный орган управляющий сигнал она должна быть многократно прокручена, т.е. должна работать циклично и чем выше частота цикла тем надежнее будет выдан управляющий сигнал . Кроме этого программа должна состоять из двух частей: статической и динамической графики для изображения работы систем автоматики и телемеханики, а также поездной ситуации на участке.

Для стыковки компьютера с напольными устройствами часть программы должна быть составлена на ассемблере.

Всем этим требованиям соответствует язык программирования Borland Pascal, но разработчики таких систем могут выбрать для этих целей и другие языки программирования удовлетворяющие выше изложенным требованиям.

1.2. Стандарты на разработку прикладных программных средств

При создании прикладных систем на ПК большая часть программных модулей составляется на языках высокого уровня (ЯВУ). Имеется несколько вариантов организации их взаимодействия, основанных как на свойствах ЯВУ, так и на особенностях операционных систем. Довольно часто возникает необходимость в программировании машинно-зависимых частей на языке ассемблера или макроассемблера, в связи с чем встает задача организации взаимодействия программ на ЯВУ с программами на ассемблере.

При проектировании большой прикладной системы с самого начала необходимо решить несколько принципиальных вопросов, касающихся общей структуры системы и способа взаимодействия отдельных компонентов. В частности, должны быть определены следующие характеристики.

А. Состав исходного текста программ, который может представлять собой:

А1) единый текст на ЯВУ или на ассемблере;

А2) отдельные текстовые модули на ЯВУ или на ассемблере, которые составляются независимо и, возможно, даже разными людьми.

Б. Структура исполняемой программы, которая может представлять собой:

Б1) единый модуль, полностью загружаемый в оперативную память при запуске системы;

Б2) несколько сегментов, загружаемых в оперативную память по мере необходимости с частичным взаимным перекрытием (наложением друг на друга);

Б3) резидентную часть, загружаемую в оперативную память в начале сеанса, и одну или несколько нерезидентных частей, загружаемых в оперативную память по мере необходимости.

В. Способ хранения данных, с которыми работает система. Основные варианты хранения:

В1) все данные располагаются в одном файле;

В2) данные распределены по нескольким файлам.

Различные сочетания указанных характеристик приводят к построению прикладных систем, которые могут отличаться очень сильно. Варианты А влияют на способ и качество разработки. Варианты Б оказывают критическое воздействие на оперативные характеристики системы — объем требуемой памяти и быстродействие. Варианты В, с одной стороны, влияют на быстродействие при доступе к данным, с другой стороны — на характер использования и экономию внешней памяти.

Рассмотрим несколько приемов реализации указанных характеристик при построении прикладных систем.

1.2.1. Структурирование программ на уровне текстовых модулей

Простые программы. Самый простой способ разработки программ не предполагает применения каких-либо приемов деления их на модули или на сегменты. Для составления такой программы обычно используется текстовый редактор общего назначения или редактор, встроенный в систему (как, например, в языке бейсик или в системе программирования Turbo Pascal). .

Простейшая программа на Паскале, печатающая на экране слово «Hello», может выглядеть следующим образом:

```
PROGRAM Test;
BEGIN
```

```
WRITELN ('Hello');
END.
```

Более сложные программы включают описания типов данных, переменных, констант. Важнейшими компонентами программ на ЯВУ являются процедуры и функции, которые обеспечивают структуризацию программ на уровне исходных текстов. Так, например, общая структура программы на Паскале может иметь вид:

```
PROGRAM  имя -программы (параметры-программы);
<описания глобальных объектов программы -
типов данных, переменных, констант>
PROCEDURE P1 (параметры процедуры P1);
<описания локальных объектов процедуры P1>
BEGIN
<тело процедуры P1>
 END;
PROCEDURE P2 (параметры процедуры P2);
<описания локальных объектов процедуры P2>
 BEGIN
<тело процедуры P2>
 END;
……
BEGIN
  <начало тела программы>
  ... P1 (...);... {обращение к P1}
  . . . P2 (...); ...{| обращение к P2}
  <конец тела программы>
  END.
```

В этом тексте можно выделить три основных компонента;

- заголовок программы — название, список параметров, описания типов, глобальных переменных, констант;
- описания процедур — их заголовки с описаниями параметров и тела, состоящие из выражений;
- тело программы — последовательность выражений, среди которых встречаются обращения к определенным выше процедурам.

Таким образом, структуризация программы на уровне исходного текста обеспечивается благодаря оформлению отдельных частей алгоритмов в виде процедур и последующему вызову этих процедур в теле программы. Все необходимые связи между формальными и

фактическими параметрами процедур устанавливаются транслятором данного языка программирования.

1.2.2. Текстовая подстановка программных модулей

Рассмотренный выше подход позволяет создавать программы, тексты которых представляют собой единое целое и хранятся в отдельных массивах на внешних носителях.

Один из приемов деления исходного текста программы на отдельные части состоит в использовании метода макрогенерации. В текст главной программы вводятся специальные выражения, указывающие компилятору на необходимость включения в нее текста других модулей. В системе программирования на основе Паскаля «включение» текстового файла Ml.PAS в программу осуществляется с помощью выражения вида:

```
{$INCLUDE: 'M1.PAS'}
```

Возможность текстовой подстановки позволяет при вводе (редактировании) исходных текстов программ иметь дело с относительно небольшими фрагментами текста, каждый из которых содержит определенную группу функций.

1.3. Раздельно компилируемые модули и библиотеки процедур

В рассмотренном выше методе соединение текстов отдельных модулей происходит в самом начале процесса трансляции—при макроподстановке; основная же работа транслятора производится над полностью собранным текстом. Если составленная таким образом программа велика по объему, например содержит от 500 до 1000 и более строк, то процесс трансляции может занимать довольно много времени — до нескольких минут, что весьма обременительно при интенсивной отладке, когда приходится часто вносить небольшие изменения в исходные тексты и вновь транслировать программу.

Чтобы избежать указанного недостатка, применяется другой подкод к построению больших программ — составление отдельных модулей, которые транслируются совершенно независимо друг от друга и должны связываться лишь на стадии окончательного формирования исполняемой программы в машинных кодах.

На языке Паскаль для организации модуля используется слово UNIT.

М о д у л ь типа UNIT. Модуль типа UNIT, который можно назвать иначе «блоком», описывается с помощью двух компонентов. Один из них — *реализация блока* — содержит тела процедур и вспомогательные типы, переменные и константы. Другой компонент — *описание блока,* или *интерфейс,* — содержит описания типов, переменных и констант, а также заголовки процедур (без их тел), которые предназначены для использования в других модулях и в главной программе. В интерфейс блока включается общий список имен указанных объектов, что делает их «видимыми» из других модулей.

Пример реализации блока P2.PAS:

```
{_____Реализация блока P2.PAS_____}
IMPLEMENTATION OF P2;
PROCEDURE get_key;
BEGIN
<тело процедуры get_key>
 END;
  <описания и тела других процедур>
 BEGIN END.
```

Интерфейс этого блока может иметь следующий вид:

```
{_____Интерфейс блока P2.INT_____}
INTERFACE;
UNIT P2;
  (get_key, key_descr,... <имена других процедур> ...);
TYPE key_descr= RECORD scan_code, ascii:byte end;
PROCEDURE get_key;
……
<заголовки других процедур данного блока>
END;
```

Использование объектов модуля типа UNIT в главной программе требует, во-первых, включения его интерфейса перед заголовком программы; во-вторых, упоминания имени блока сразу после заголовка программы, для чего служит выражение вида:

USES <имя блока>;

Начальная часть программы приобретает в результате такой вид:

```
{_____Главная программа — PP.PAS_____}
{$INCLUDE: 'P2.INT'}
……
PROGRAM PP (input, output);
```

USES P2;
……
{_____}
Здесь текстовое включение интерфейса блока P2 в программу PP.PAS осуществляется с помощью рассмотренного выше выражения $INCLUDE.

1.4. Генерация объектных модулей и загрузочных файлов

В результате компиляции отдельных текстовых модулей порождаются так называемые объектные модули. Например, обращение к транслятору IBM-Pascal для трансляции исходной программы PP.PAS имеет вид:
PAS1 PP;
PAS2

Объектный модуль, который порождается после двух проходов трансляции (PAS1 и PAS2), можно считать «полуфабрикатом» — куском машинного кода, готовым к превращению в загрузочный файл. Объектный модуль заносится в особый файл, которому придается тип OBJ. Следовательно, Паскаль-транслятор осуществляет преобразование файла типа PAS в файл типа OBJ. Для рассматриваемого нами примера этот процесс можно изобразить следующей условной схемой:

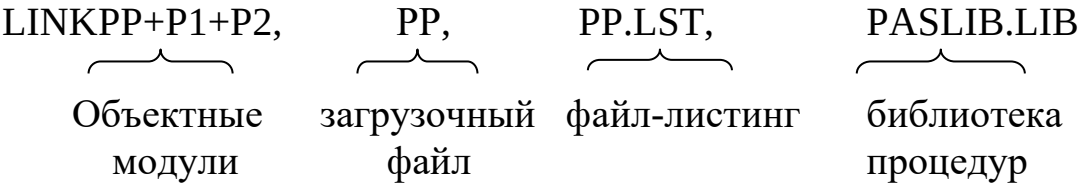
PP.PAS — транслятор → PP.OBJ

Следующий этап состоит в преобразовании совокупности отдельных объектных модулей в загрузочный файл, т. е. в файл типа EXE или COM. Этот процесс называется связыванием объектных модулей или сборкой задачи. Соответствующая схема преобразования:

PP.OBJ → PP.EXE или PP.OBJ →PP.COM

Данный этап реализуется специальной системной программой LINK, которую называют редактором связей или компоновщиком. При обращении к LINK указываются все объектные модули, которые должны быть объединены в общую программу; указывается также имя файла с результирующей программой, имя файла с листингом (необязательный параметр) и имя файла с библиотекой процедур.

Пример обращения к LINK для связывания модулей, оттранслированных с Паскаля:



1.5. Библиотеки объектных модулей

Один из способов оформления независимых частей программы состоит в создании библиотек объектных модулей, которые затем можно включать в формируемую задачу на стадии сборки. Для формирования библиотеки объектных модулей служит вспомогательная программа LIB, которая позволяет создать новую библиотеку или пополнить старую библиотеку процедурами, которые извлекаются из оттранслированных заранее объектных модулей или из другой библиотеки. Общий вид обращения к программе LIB:

LIB <старая –библиотека> <размер-страницы>
<операции>,
<файл-с-листингом>, <новая-библиотека>

При обращении к LIB могут указываться имена старой и новой библиотек. Размер страницы (= 16,32, ... 512) задает величину буфера для обмена; это необязательный параметр. Главные действия задаются *операциями.* Операция может иметь следующие разновидности (mm — имя модуля):

\+ mm {добавление модуля в библиотеку},
\- mm {удаление модуля из библиотеки},
* mm {извлечение модуля из библиотеки},
\- + mm {замена модуля в библиотеке},
\-* mm {извлечение модуля с удалением}.

Пример обращения к LIB:

LIB OLDLIB +P2, ,NEWLIB

Такое обращение вызывает создание новой библиотеки NEWLIB из старой OLDLIB с добавлением отдельно оттранслированного модуля P2. После того как библиотека создана, достаточно упомянуть ее имя при связывании объектных модулей в единую программу. Обращение к библиотечным процедурам в

главной программе или в любом другом модуле требует их упоминания в начале программы с указателем EXTERNAL, — так же как и в случае использования модулей типа MODULE. При этом необходимо помнить о согласовании типов параметров библиотечных процедур и функций.

Каждая система программирования обыкновенно имеет собственную библиотеку стандартных процедур/функций. Файлы с библиотеками для Паскаль- и си-трансляторов обычно имеют тип LIB. Эти файлы указываются при обращении к компоновщику программ LINK.

Итак, весь процесс трансляции складывается, в общем случае, из следующих стадий:

— формирование текстовых модулей (с использованием текстовой подстановки);

— синтаксический анализ и выдача ошибок, найденных транслятором в тексте программы;

— генерация объектных модулей в машинных кодах, оптимизация;

—сборка из объектных модулей исполняемого кода программы.

Отнюдь не во всех системах программирования присутствуют все эти стадии. Например, в популярной системе Turbo Pascal не поддерживается раздельная компиляция модулей. Как следствие, отпадает потребность в использовании редактора связей. Это возможно благодаря тому, что в системе Turbo Pascal транслятор порождает сразу исполняемый код и работает очень быстро. Это весьма привлекательная черта для многих пользователей, несмотря на отсутствие раздельной компиляции модулей.

Большинство трансляторов имеют несколько *фаз* или *проходов,* предназначенных для выполнения специфической работы над полным текстом программы. Так, в системе IBM-Pascal транслятор двухпроходный, а в системе С86—четырехпроходный.

Помимо фазы синтаксического анализа и генерации объектных модулей, часто отдельно выделяется фаза оптимизации. Чем больше таких фаз, тем дольше идет процесс трансляции, но зато результирующая программа получается более качественной.

Рассмотренные приемы позволяют при составлении исходных текстов программ иметь дело с несколькими относительно автономными компонентами, которые собираются вместе либо в начале процесса трансляции (текстовое включение), либо при сборке исполняемых программ (использование раздельно компилируемых модулей и библиотек процедур). Благодаря такому методу программы

становятся более удобными для анализа и модификации. Кроме того, появляется возможность нескольким программистам участвовать в разработке одной системы; каждый из них может заниматься своими модулями, при условии, что заранее оговорены «соглашения о связях», включающие имена и способы обращения к процедурам, входящим в разные модули. Наконец, раздельная компиляция позволяет собирать программы из модулей, составленных на разных языках программирования, при условии, что согласованы способы передачи и обработки параметров процедур и функций.

1.6. Реализация сегментированных программ с перекрытиями

Рассмотренные выше методы структурирования обеспечивают независимую разработку отдельных частей прикладной системы; однако получаемый в результате исполняемый программный код представляет собой единый файл, который при вызове программы должен полностью разместиться в оперативной памяти. Это далеко не всегда устраивает разработчиков системы. Необходимы способы деления программ на такие части *(сегменты},* которые могли бы постоянно находиться во внешней памяти и загружаться в оперативную память лишь по мере необходимости.

В рамках развитой системы программирования, базирующейся на языке высокого уровня типа Паскаль, один из распространенных приемов такого деления программ на части основан на создании *перекрывающихся (оверлейных} сегментов.* При таком подходе программа составляется из отдельных кусков, которые во время работы могут по мере необходимости загружаться в оперативную память и частично накладываться друг на друга.

Сегменты хранятся во внешней памяти (на диске) и лишь один из них — *корневой* — находится постоянно в оперативной памяти. Когда в корневом сегменте происходит обращение к процедуре, тело которой находится в одном из оверлейных сегментов, отсутствующих в данный момент в оперативной памяти, производится его загрузка с внешнего носителя в ОЗУ. При этом все связи между частями корневого сегмента и только что загруженным сегментом начинают выглядеть так, как если бы они составляли с самого начала единую программу. Точно так же происходит по мере необходимости загрузка других сегментов из внешней памяти.

Сегменты могут быть связаны в сложные деревообразные структуры и при загрузке размещаться в оперативной памяти так, чтобы наиболее эффективно, использовать выделенное пространство. Так, например, в системе Turbo Pascal можно объявлять оверлейные процедуры, тела которых после трансляции попадают в соответствующие оверлейные файлы. Пример такого объявления:

```
PROGRAM PP (input, output);
……..
OVERLAY PROCEDURE PI;
BEGIN
<тело процедуры PI>
 END;
OVERLAY PROCEDURE P2;
BEGIN
<тело процедуры P2>
 END;
PROCEDURE P3;
BEGIN
<тело процедуры P3>
 END;
OVERLAY PROCEDURE P4;
BEGIN
<тело процедуры P4>
 END;
……..
BEGIN
(тело программы PP)
 END.
```

Трансляция такой программы приведет к созданию трех перекрывающихся сегментов:

PP.COM
PP.000
PP.001

Первый из них — PP.COM — соответствует главной программе, два других сегмента — оверлейные. При этом по правилам системы TurboPascal процедуры P1 и P2 попадут в сегмент PP.000, поскольку в исходном тексте их описания следуют одно за другим.

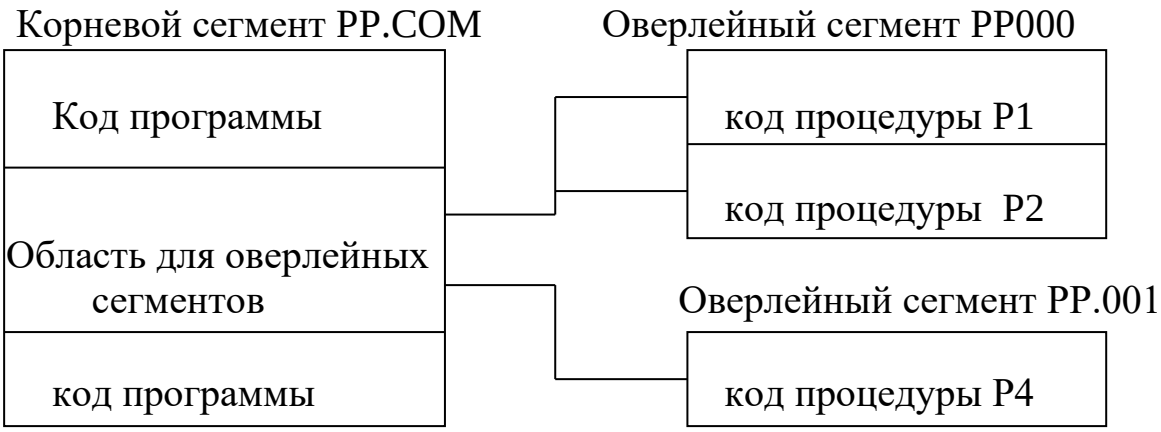


Рис. 1. Соотношение корневого и оверлейных сегментов

Процедура P4 окажется во втором оверлейном сегменте — PP.001. Такое разделение обусловлено тем, что описание P4 отделено от описаний двух первых процедур внутренней (не оверлейной) процедурой P3. Получившаяся структура программы может быть отображена схемой, представленной на рис.1

Область памяти, выделяемая в корневом сегменте для размещения оверлейных сегментов, рассчитывается системой автоматически, исходя из длины максимального оверлейного сегмента. Процедуры разных оверлейных сегментов, например, P1 и P4, не могут вызывать друг друга, так как соответствующие им сегменты не могут одновременно размещаться в оперативной памяти. Чем больше число оверлейных сегментов, тем больше будет происходить обменов с внешними накопителями, и работа системы будет замедляться.

Следовательно, применение перекрывающихся сегментов требует тщательного планирования, чтобы наилучшим образом использовать выделяемую оперативную память и при этом не слишком проигрывать в быстродействии системы.

1.7. Организация взаимодействия программ

Взаимодействие программных модулей, рассмотренное в предыдущем параграфе, основывалось на механизме взаимного вызова процедур для конкретных ЯВУ. При этом предполагалось, что отдельные модули хотя и разрабатываются независимо, но достаточно тесно увязаны друг о другом, т. е. имеют согласованные процедурные интерфейсы.

Довольно часто, однако, встает вопрос об организации взаимодействия программ, которые составлены независимо и к тому же на

разных языках программирования (например, си и Паскаль). В этом случае для взаимного вызова программ можно воспользоваться прерываниями DOS. Другой универсальный метод взаимодействия — использование подпрограмм на языке ассемблера, через посредство которых можно не только иметь прямой доступ к аппаратуре, но н осуществлять передачу параметров между разными программами на ЯВУ. Рассмотрим кратко особенности указанных методов.

1.7.1. Взаимодействие программ через прерывания DOS

В DOS имеется специальное прерывание с десятичным номером 33 (шестнадцатеричный номер 21 H), через которое любая прикладная программа, может иметь доступ к внутренним функциям операционной системы. В их число входит несколько функций, с помощью которых может быть организован взаимный вызов программ. Дадим краткое описание этих функций.

Функция 31 H (здесь символ H означает, что число является шестнадцатеричным) останавливает данную программу и оставляет ее в оперативной памяти резидентной, что дает возможность позднее вновь обратиться к ней через соответствующую ***точку входа***.

Функция 4B H обеспечивает вызов (загрузку с диска и переход на исполнение) другой программы. Когда вызванная таким образом программа закончится, управление автоматически возвращается вызывавшей программе. Имеется вариант этой функции, когда файл только загружается с диска, но не исполняется; это используется для загрузки перекрывающихся сегментов программ или загрузки данных.

Функция 4C H оканчивает работающую программу с засылкой в системный регистр AL ***кода возврата***. Этот код может быть взят и проанализирован вызвавшей программой, для чего используется функция 4D. Функция 4D позволяет выяснить, по какой причине окончилась вызванная программа. Причин может быть четыре:

— нормальное окончание;

— окончание в результате нажатия пользователем клавиш Ctrl + Break;

— окончание в результате фатальной ошибки внешнего устройства;

— окончание в результате применения функции 31. Функции 48 H и 49 H позволяют соответственно запросить у DOS оперативную память для работы программы и освободить ее. Функция 4A H

позволяет изменить (уменьшить или увеличить) выделенную память. Указанные функции дают возможность прикладной программе регулировать объем занимаемой оперативной памяти.

Большинство развитых систем программирования на основе ЯВУ обеспечивает обращение к прерываниям DOS через специальные процедуры. При этом в регистры микропроцессора засылаются необходимые параметры. Так, например, в системе Turbo Pascal обращение к прерыванию DOS осуществляется процедурой:

INTR (InterruptNo, Registers)

Здесь параметр InterruptNo: INTEGER — целое десятичное число, указывающее номер прерывания. Параметр Registers — это список базовых регистров микропроцессора:

Registers = RECORD
AX, BX, CX, DX, BP, SI, Dl, DS, ES, Flags : INTEGER;
END;

При обращении к прерыванию DOS необходимо сначала занести в регистры нужные значения, а после завершения прерывания из них можно извлечь результаты работы.

Каждый из этих двухбайтовых регистров состоит фактически из двух однобайтовых (полу)регистров. Например, двухбайтовый регистр AX состоит из двух однобайтовых — AH и AL. Однобайтовые регистры используются для передачи информации в DOS. Так, по принятому в DOS соглашению регистр AH служит для задания номера функции, вызываемой через любое прерывание.

Если, например, в прикладной программе применяется прерывание 21 H и через него вызывается функция 4B H (загрузка и запуск Другой программы), то регистры заполняются следующим образом:

AH = 4B — номер вызываемой функции;

AL—признак загрузки программы с исполнением (AL = 0) или без исполнения (AL == 1);

DS : DX—два указанных регистра содержат «длинный» адрес строки с расширенным именем файла;

ES: BX — два этих регистра содержат длинный адрес управляющего блока запускаемой программы, который должен быть оформлен соответствующим образом.

Чтобы пользоваться указанным методом для организации взаимодействия программ, от программиста требуется определенная профессиональная подготовка, в частности, знание архитектуры ПК и DOS, умение работать с значениями регистров и др.

Обладание этими приемами открывает доступ к мощным средствам управления программами. Использование разных языков программирования в этом случае уже не будет препятствием для сочетания различных программ в рамках одной прикладной системы. Кроме того, через прерывания DOS можно непосредственно обращаться к функциям, обслуживающим различные внешние устройства — дисплей, клавиатуру, дисковые накопители, принтер, коммуникационные каналы.

1.7.2. Взаимодействие с программами на языке ассемблера

Помимо возможности прямого обращения к прерываниям DOS из программ на ЯВУ, при создании сложных прикладных систем обычно возникает необходимость в использовании машинно-ориентированных программ на языке ассемблера (часто говорят «программ на ассемблере» или «ассемблерных программ»). Более того, с целью повышения быстродействия или сокращения требуемых объемов памяти на ассемблере иногда составляются значительные куски прикладных программ.

В этих случаях возникает задача организации взаимодействия программ на ЯВУ с ассемблерными программами. Прежде всего отметим, что ассемблерные программы могут объединяться с другими программами на уровне объектных модулей—с помощью компоновщика LINK. Главная проблема состоит во взаимной передаче параметров между такими программами. Рассмотрим общий метод передачи параметров в ассемблерные программы.

Допустим, имеется ассемблерная процедура для установки положения курсора в дисплейном окне, которая должна быть доступна из программ на паскале. Описание соответствующей процедуры в Паскаль- программе может иметь вид:

```
PROCEDURE CURPOS (LINE : INTEGER; POS: INTEGER);
EXTERNAL;
```

Соответственно обращение к этой процедуре в Паскаль-программе может иметь вид:

```
CURPOS (L, P);
```

Вызываемая программа на ассемблере имеет следующий примерный вид:

```
WINDOW  SEGMENT'CODE'     ;начало сегмента с именем WINDOW
        PUBLIC CURPOS     ;объявление «общедоступной»
                            процедуры CURPOS
```

```
CURPOS  PROC    FAR     ;обеспечение «дальнего» вызова
                          процедуры
        PUSH    BP      ;сохранение старого указателя на
                          «фрейм»
        MOV     BP,SP    ;установка нового положения
                          ; «фрейма»
        PUSH    AX       ;сохранение старых значений
                         ;регистров
        PUSH    BX       ;AX и BX
        MOV BX, [BP+6]   ;перенос в регистр BX 2-го
                          параметра
        MOV AX, [BP+8]   ; перенос в регистр AX 1-го
                          параметра
            ……..
        (выполнение необходимых действий с использованием
    параметров, сохраненных в регистрах AX и BX)
        POP     BX      ; восстановление регистров BX
                        ; и AX
        POP     AX
        POP     BP      ;восстановление старого указателя
                       ; на стек
        RET     8       ; возврат из процедуры со сдвигом
                        ;указателя
                        ;стека на 8 байт назад
CURPOS  ENDP            ;конец тела процедуры  CURPOS
WINDOW  END             ;конец сегмента WINDOW
```

В приведенном примере можно отметить несколько характерных деталей. Прежде всего необходимо пояснить, что передача информации между вызывающей программой и данной процедурой осуществляется через ***стек*** — последовательность машинных слов, в которую данные «заталкиваются» оператором PUSH и «выталкиваются» оператором POP. В стек автоматически заносится также ***адрес возврата*** из процедуры. На текущую ячейку стека указывает содержимое регистра SP. В регистре BP находится указатель на ***фрейм***, т.е. на ту часть стека, в которой хранятся данные, относящиеся к вызванной процедуре.

В начале процедуры происходит сохранение в стеке старого значения регистра BP, а также регистров AX и BX, которые далее понадобятся для работы. Затем происходит очень важная операция — из стека командой MOV извлекаются значения двух параметров (за-

данных в обращении к Паскаль-процедуре CURPOS как значения переменных L и P). Эти значения наносятся соответственно в регистры AX и BX и используются затем для выполнения основных действий — установки положения курсора в дисплейном окне.

Положения параметров в стеке фиксированы относительно начале фрейма, заданного значением указателя BP. Первые 4 байта заняты адресом возврата и старым значением SP, еще 2 байта — регистров счетчика команд, а параметры занимают по 2 следующих байта и поэтому адресуются конструкциями вида [BP + 6] и [BP + 8], Следует иметь в виду, что при обращении к данной процедуре из Паскаля стек заполняется от старших адресов к младшим, поэтому первый параметр находится дальше всего от позиции, на которую указывает регистр BP.

Если бы параметров было 4, то они адресовались бы с помощью следующих выражений:
[BP+6]—адрес 4-го двухбайтового параметра,
[BP+8]—адрес 3-го двухбайтового параметра,
[BP + 10]—адрес 2-го двухбайтового параметра,
[BP+12]—адрес 1-го двухбайтового параметра.

Важным моментом является то, что в данном примере процедура CURPOS не меняет значений указанных параметров. Это позволяет передать их из Паскаль- программы *по значению.,* т. е. попросту скопировать в стек. Если же предполагается изменение параметров в ассемблерной процедуре, то их нужно передавать *по ссылке.* Для этого описание процедуры CURPOS в Паскаль-программе должно было бы содержать описания параметров с указателем VAR или VARS, т. е. иметь, например, следующий вид:
PROCEDURE CURPOS (VAR LINE : INTEGER;
VAR POS:INTEGER); EXTERNAL;

В этом случае и извлечение параметров в ассемблерной процедуре должно быть оформлено по-другому. Сначала нужно извлечь из стека ссылку на параметр (его адрес), а затем уже взять по этому адресу значение ячейки. Соответствующие операторы для 1-го параметра выглядели бы следующим образом:
MOV BX, [BP+8] занесение в BX адреса 1-го параметра
MOV AX, [BX] занесение в AX значения 1-го параметра
Возврат значения этого параметра Паскаль -программе должен быть в этом случае осуществлен обратными операциями:
MOV BX, [BP-(-8] [установка в BX адреса 1-го параметра
MOV [BX],AX ;возврат значения из AX в 1-й параметр

Аналогичным образом следует поступить со 2-м и со всеми последующими параметрами.

Следует иметь в виду, что в фортране передача параметров осуществляется только по ссылке, вне зависимости от того, меняются ли значения параметров в вызываемых подпрограммах или нет.

В языке си по умолчанию параметры передаются по значению, но поскольку в этом языке очень часто сами значения являются адресами других ячеек, то при работе с передаваемыми параметрами следует проявлять очень большую аккуратность. Поскольку ни в языке си, ни тем более в ассемблере нет никакого контроля за правильностью передачи параметров, вся забота о правильной обработке ссылок и значений возлагается на программиста. В си можно явно указать, что переменная должна содержать адрес, а не значение; для этого переменная снабжается префиксом. Здесь проявляется некоторое сходство с языком паскаль, где для этой же цели служит указатель VAR.

Помимо того, в си порядок размещения параметров на стеке противоположен рассмотренному выше и принятому в Паскале, а размещаются они на два байта ближе к началу фрейма, чем в случае Паскаля. Наконец, в си, в отличие от Паскаля, допускается переменное число параметров при обращении к функциям, что может дополнительно усложнить их обработку в ассемблерной программе.

Из всего этого следует, что одна и та же ассемблерная подпрограмма не может обслуживать программы и на си, и на паскале, если только не применять специальных ухищрений. Для интерфейса с каждым из этих языков требуется спой подход к передаче и обработке параметров.

1.7.3. Резидентные программы

Часто бывает необходимо, чтобы служебная программа сработала один раз и осталась в памяти для того, чтобы к ней позднее могли обращаться другие программы. Такую программу называют *резидентной,* в отличие от обычных программ, которые по окончании работы освобождают занятую память.

Как указывалось выше, DOS-функция 31 H осуществляет остановку программы и оставляет ее резидентной в памяти. К этой функции можно обращаться непосредственно из программы на ЯВУ, если в нем обеспечивается доступ к прерываниям DOS. Можно

выполнить те же действия, используя соответствующую ассемблерную подпрограмму.

Фрагмент программы на ассемблере, предназначенной для реализации указанной операции, имеет следующий вид:

```
PUSH        ES          ;запоминание начала программы в стеке
……….                    ;
POP         AX          ;выталкивание адреса начала
                        ;программы из стека в регистр AX
MOV         DX, SEG ZZ  ;занесение адреса конца программы ZZ
                        ;в регистр DX
SUB         DX, AX      ;вычисление длины программы
INC         DX          ;увеличение длины на 1
MOV         AH, 31H     ;задание в регистре AH номера
                        ;функции 31 H (остановить задачу и
                        ;сделать ее резидентной)
INT         21H         ;обращение к прерыванию 21 H
ZZ SEGMENT 'ZZ'         ;конец программы ZZ
ZZ ENDS
```

Такая подпрограмма может быть соединена с любой другой программой на ЯВУ и использована для создания резидентной копии задачи.

1.7.4. Связывание программ через потоки ввода/вывода

Рассмотренные выше приемы основаны на проникновении внутрь операционной системы и требуют, как правило, тщательного программирования для обработки параметров, передаваемых между программами. Существует, однако, гораздо более простой путь для организации взаимодействия программ, основанный на стандартном сервисе, предоставляемом DOS.

В DOS имеется понятие ***стандартного входного*** и ***стандартного выходного*** устройства. По умолчанию эти устройства соответствуют клавиатуре и дисплею. Имеется возможность переопределять стандартные устройства (или потоки) ввода и вывода. Вместо клавиатуры *и* дисплея в этой роли могут выступать, во-первых, различные внешние устройства (например, принтер или коммуникационный канал), во-вторых, любые дисковые файлы, и, в-третьих, любые программы, работающие со стандартным входом и выходом.

В программе на ЯВУ стандартное входное и стандартное выходное устройства доступны через обычные операторы ввода/вывода: в па-скале — READ и WRITE, в языке си—getchar, putchar и printf. Любой начинающий программист знает, что такие операторы позволяют обмениваться, информацией с клавиатурой и дисплеем, не задумываясь о том, что здесь кроются более широкие возможности.

В командной строке, обращенной к DOS и предусматривающей запуск какой-либо программы, можно указать, откуда должен поступать в программу стандартный входной поток и куда должен направляться стандартный выходной поток. Благодаря этой возможности можно, не меняя программ, подключать к ним различные внешние устройства и файлы в качестве источников и приемников информации, а также организовывать взаимную передачу информации между отдельными программами. Если имя программы — PP, то изменить ее входной и выходной потоки можно командой следующего вида:

PP < from> to

Здесь символ «from» соответствует стандартному входному потоку, а символ «to» — стандартному выходному потоку. Приведенная выше запись может иметь варианты, когда указывается только входной или только выходной поток:

PP < from или PP > to

Вместо символов from и to могут фигурировать имена файлов или зарезервированные имена внешних устройств (CON:, PRN:, LPT1:, LPT2:, COM1:, COM2:, AUX:). Следует помнить, что одни внешние устройства могут использоваться только на выходе (например, принтер), другие только на входе (например, диджитайзер), но есть и такие устройства, которые способны передавать информацию в обоих направлениях (например, модемы).

Если источником или приемником информации для данной программы является другая программа, то используется иное обозначение. Пример такого обозначения для трех взаимосвязанных программ:

PP1 | PP2 | PP3

В данном примере стандартный выходной поток программы PP1 связывается со стандартным входным потоком программы PP2, а ее стандартный выходной поток, в свою очередь, поступает на стандартный вход программы PP3.

Для обозначения таких цепочечных связей между программами используют термин «канал» (англ. pipe). В DOS такой канал реализуется с помощью временного файла, который операционная система сама создает в корневом каталоге.

ГЛАВА 2. ИНТЕГРИРОВАННАЯ СРЕДА BORLAND PASCAL

2.1. Общие сведения об интегрированной среде Borland Pascal

Borland Pascal является не просто быстрым компилятором Паскаля, он представляет собой эффективный компилятор Паскаля с интегрированной интерактивной средой разработки программ, которую легко изучить и легко использовать в работе. При работе с Borland Pascal нет необходимости использовать отдельный редактор, компилятор и редактор связей (компоновщик) для создания и выполнения программ на Паскале. Все эти программные средства встроены в Borland Pascal, и ко всем из них есть доступ из интегрированной среды программирования Borland Pascal.

Borland Pascal обеспечивает получение контекстной экранной справочной диалоговой информации по нажатию одной клавиши. Эту подсказку можно получить в любой момент (за исключением того времени, когда выполняется программа), нажав клавишу F1. При этом на экран выведется полное описание функций того элемента, при работе с которым вы обратились за помощью.

Экран подсказки может содержать одно или несколько ключевых слов (элемент с подсветкой), по которым вы можете получить информацию. Используйте клавиши управления движением курсора (стрелки) для перемещения к любому ключевому слову, а для получения детальной подсказки по этому выбранному элементу, нажмите клавишу "Enter". Для перемещения, соответственно, на первое и последнее ключевое слово на экране вы можете использовать клавиши Home и End.

2.2. Загрузка Borland Pascal

Программное обеспечение Borland Pascal может быть записано на жестком диске или гибком диске.

Если вы используете дисковод для гибких дисков, вставьте системный диск Borland Pascal в дисковод A: и нажмите клавиши Ctrl-Alt-Del для перезагрузки системы. После перезагрузки введите в ответ на системную подсказку следующую команду:

A>bp.exe

и нажмите клавишу "Enter". Это приведет к запуску программы BP.EXE, которая установит интегрированную среду программирования и поместит вас в основное меню.

При использовании жесткого диска перейдите в подкаталог BP и запустите BP.EXE, напечатав следующее:

C:BP>bp.exe

Примечание: в примере приводится частный случай загрузки Borland Pascal .,
когда файл BP.EXE находится в подкаталоге BP
на диске C:.

После загрузки Borland Pascal появляется изображение, содержащее основной экран меню, представленное на рис. 2.

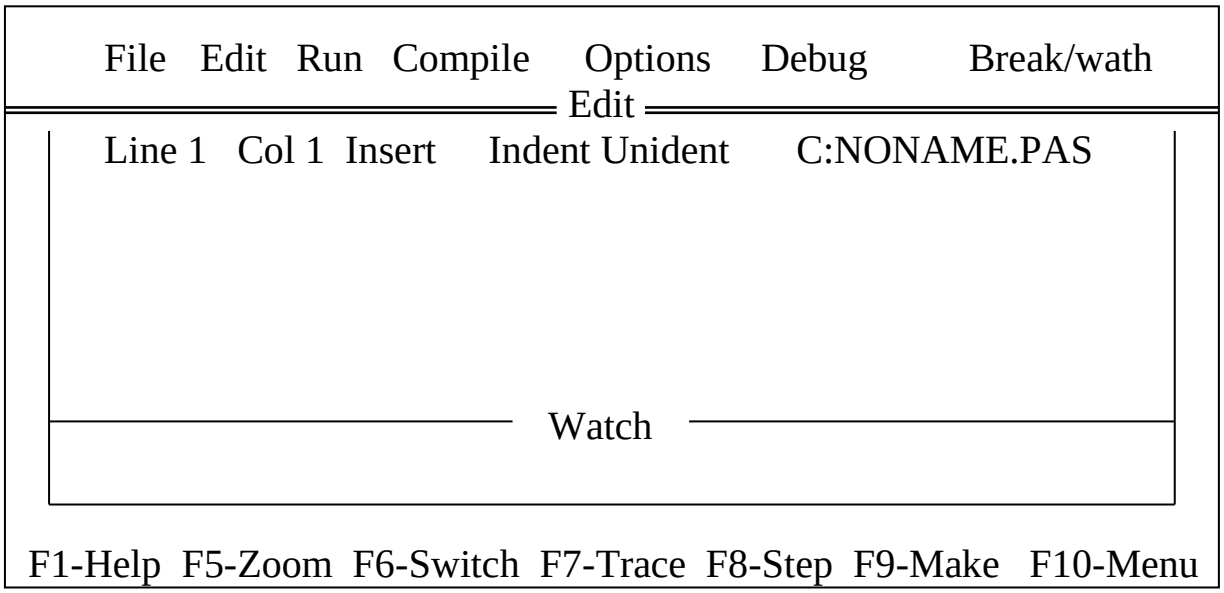


Рис. 2. Основной экран меню Borland Pascal

Посмотрите внимательно на основное изображение экрана; оно содержит четыре части: основное меню, окно редактирования, окно

просмотра и нижнюю строку (которая указывает, какая клавиша что выполняет в данном конкретном случае).

Для ознакомления с системой Borland Pascal ниже приводятся некоторые основные правила перемещения в ней.

Действия в среде меню:

- Используйте прописную букву с подсветкой для выбора элемента меню или используйте клавиши управления движением курсора для перемещения к нужному элементу меню, а затем нажмите клавишу "Enter".
- Нажмите клавишу Esc для выхода из меню.
- Для переключения между меню и активным окном нажмите клавишу F10.
- Находясь в режиме основного меню, нажмите клавишу Esc для возврата к окну, которое перед этим было в активном состоянии. (Находясь в активном состоянии, окно имеет двойную верхнюю границу, а его имя имеет подсветку).

Действия из любого положения в Borland Pascal :

- Нажмите клавишу F1 для получения информации о своем текущем положении (подсказка о том, что осуществляется: выполнение, компиляция и так далее).
- Нажмите клавишу Alt вместе с первой буквой любой команды из основного меню (F, E, R, C, O) для запуска указанной команды. Например, нажатие клавиш Alt-E из любого места в системе вызовет перемещение в окно редактирования; нажатие клавиш Alt-F переместит вас в меню File (Файл).
- Нажмите клавишу F5 для того, чтобы увеличить окно, находящееся в активном состоянии, до размеров экрана или уменьшить его до прежнего размера.
- Нажмите клавишу F6 для переключения окон.
- Для переключения содержимого окна нажмите клавиши Alt-F6. (Когда активным является окно редактора, то есть окно просмотра (Watch) или окно вывода (Output), клавиши Alt-F6 позволяют переключаться между последним и текущим файлом, а когда активно другое окно (окно просмотра или вывода), эти клавиши позволяют переключаться между окнами просмотра и вывода).

Примечание. Для выхода из Borland Pascal и возврата в операционную систему (OS), нажмите клавиши Alt-X или перейдите в меню File (Файл) и выберите Quit (Выход) (нажмите Q или переместите поле выбора на Quit и нажмите Enter). Если вы указываете Quit, не записав своего текущего рабочего файла на диск, то редактор дает запрос, следует ли сохранить этот файл или нет.

2.3. Клавиши оперативного вмешательства

Borland Pascal предлагает несколько клавиш быстрого оперативного вмешательства (сокращенные варианты команд), которые также можно использовать. Клавиши быстрого вмешательства представляют собой клавиши, нажатие которых вызовет выполнение команды или некоторого действия.

В таблице 1 приведен список всех клавиш оперативного вмешательства, которые можно использовать при работе в интегрированной среде.

Таблица 1

Клавиши оперативного вмешательства Borland Pascal

Клавиша (и)	Функция	Эквивалент меню
F1	Вызывает появление окна подсказки с информацией о вашем текущем положении	
F2	Записывает на диск файл, находящийся в настоящий момент в редакторе	File/Save
F3	Позволяет загрузить файл	File/Load
F5	Увеличивает окно, находящееся в активном состоянии, до размеров экрана и уменьшает его до прежних размеров	
F6	Переключает окно в активное состояние	
F7	Выполняет трассировку программы	
F8	Выполняет пошаговое выполнение программы	
F9	Выполняет команду "Make" (создать)	
F10	Позволяет переключаться между меню и активным окном	

Alt-F1	Вызывает появление экрана диалоговой информации, к которому вы перед этим обращались	
Alt-F3	Позволяет указать файл для загрузки	File/Pick
Alt-F5	Возвращает к сохраненному экрану	Run/User Screen
Alt-F9	Выполняет компиляцию программы	Compile/Compile
Alt-B	Перемещает вас в меню "Останов/ Просмотр"	(Break/Watch)
Alt-C	Включает меню Compile (Компилировать)	
Alt-E	Включает режим редактора	
Alt-F	Включает меню FILE (Файл)	
Alt-O	Включает меню Options (Опции)	
Alt-R	Запускает вашу программу на выполнение	
Alt-X	Завершает работу Borland Pascal и возвращает вас в OS	File/Quit
Ctrl-F1	Позволяет получить помощь по языку из среды редактора	
Ctrl-F2	Завершает сеанс редактирования	Run/Program Reset
Ctrl-F9	Выполняет запуск программы	Run/Run
Shift-F10	Выводит экран версии	

2.4. Основные команды текстового редактора

Команды редактора интегрированной среды Borland Pascal базируются на подмножестве команд популярного текстового редактора WordStar, а принципы работы во многом совпадают с системами Турбо-Бейсик, Турбо-Пролог, Турбо-Си и др.

Команды можно подразделить на команды управления курсором, команды вставки и удаления, прочие команды.

Для перемещения курсора на одну позицию влево и вправо служат клавиши <стрелка влево> и <стрелка вправо> соответственно. Для перемещения курсора на одну строку вверх и вниз служат клавиши <стрелка вверх> и <стрелка вниз> соответственно. Для перемещения экрана по тексту на одну страницу вверх и вниз используются клавиши PgUp и PgDn соответственно, другие команды перемещения курсора приведены в таблице 2.

Таблица 2

Команды перемещения курсора

Команда	Клавиша(и)
Перемещение на слово вправо	Ctrl-->
Перемещение на слово влево	Ctrl-<-
Плавное перемещение экрана вверх	Ctrl-W
Плавное перемещение экрана вниз	Ctrl-Z
Перемещение в начало строки	Home
Перемещение в конец строки	End
Перемещение на верхнюю строку экрана	Ctrl-Home
Перемещение на нижнюю строку экрана	Ctrl-End
Перемещение в начало текста	Ctrl-PgUp
Перемещение в конец текста	Ctrl-PgDn

Команды, осуществляющие вставку и удаление информации, приведены в таблице 3.

Таблица 3

Команды вставки и удаления

Команда	Клавиша(и)
Удаление символа в позиции курсора	Del
Удаление символа слева от курсора	BackSpace
Удаление слова справа от курсора	Ctrl-T
Удаление от курсора до конца строки	Ctrl-Q Y
Удаление строки	Ctrl-Y
Вставка разделителя строк	Enter или Ctrl-N

При наборе текста в Borland Pascal каждая строка обязательно завершается нажатием клавиши "Enter". При этом в конце каждой строки возникнет невидимый разделитель строк. Чтобы объединить две строки текста в одну (удалить разделитель строк между ними), следует установить курсор в любую позицию правее последнего символа первой из этих строк (удобно это делать клавишей End) и нажать клавишу Del.

2.5. Создание первой программы

Когда Вы в первый раз входите в Borland Pascal , Вы оказываетесь в основном меню. Нажмите клавишу E для перехода в окно редактирования (или же вы можете использовать клавиши управления движением курсора для выбора команды Edit, а затем нажмите клавишу "Enter"). Теперь вы находитесь в редакторе, и курсор расположен в верхнем левом углу. Можно начать набирать следующую программу, нажимая "Enter" в конце каждой строки:

```
program MyFirst;
var
  A,B     : integer;
  C       : real;
begin
  Write('Введите два числа: ');
  Readln(A,B); C:= A/B;
  Writeln('Отношение равно ', C);
  Writeln('Нажмите клавишу <Enter>');
End.
```

Для передвижения в окне редактирования используйте клавиши управления движением курсора. В случае, если вы допустите ошибку, можно использовать клавиши управления движением курсора для перемещения.

Для удаления символов можно использовать клавишу "Backspace", а для вставки просто набирайте новый текст.

После того, как Вы ввели Вашу программу, имеет смысл записать ее на диск. Для этого необходимо войти в меню (File), и выбрать команду "Сохранить" (Save). По умолчанию вашему файлу будет присвоено имя NONAME.PAS. Теперь Вы можете переименовать его. Для этого выберите команду "Записать" (Write to). В ответ на запрос "Новое имя?" (New name) наберите MYFIRST.PAS, а затем нажав клавишу "Enter".
Ваша программа будет сохранена на диске как MYFIRST.PAS.

Команду Save можно подать из любого состояния системы Borland Pascal , если нажать клавишу F2.

Если Вы хотите загрузить сохраненную программу с диска, то для этого необходимо выбрать команду "Загрузить" (Load).

После выбора команды на экране появляется прямоугольник, в котором надо задать имя файла. Можно задать в прямоугольнике не имя конкретного файла, а групповое имя, содержащее символы * и ? в качестве маски, например *.PAS, *.* и др. В этом случае на экране возникнет прямоугольник с именами файлов, удовлетворяющих условию, заданному маской. Пользуясь клавишами управления курсором, следует вывести имя нужного файла и нажать клавишу "Enter".

Команду Load можно подать из любого состояния системы Borland Pascal , если нажать клавишу F3.

Если Вы хотите очистить окно редактирования, то для этого необходимо выбрать команду "Очистить" (New). После выбора команды окно редактирования очищается, курсор устанавливается в левом верхнем углу. По умолчанию файлу в окне редактирования присваивается имя NONAME.PAS.

Для просмотра каталога необходимо выбрать команду "Каталог" (Directory). Перед просмотром предоставляется возможность установки интересующего дисковода, маршрута, группового имени файлов. Для просмотра имен всех файлов (маска *.* по умолчанию) достаточно просто нажать клавишу "Enter".

На экране появится прямоугольник с именами файлов и подкаталогов. Можно высветить имя нужного файла, пользуясь клавишами управления курсором. Для перехода в подкаталог более низкого уровня нужно выбрать соответствующее имя подкаталога и нажать клавишу "Enter". Для перехода в подкаталог более высокого уровня нужно выбрать ..\ и нажать клавишу "Enter".

Для компиляции Вашей первой программы возвратитесь в основное меню. Если Вы все еще находитесь в окне редактирования, то для этого нажмите F10. Нажмите C для выбора меню Compile (Компилировать), а затем нажмите C еще раз для вызова команды Compile из этого меню, или нажмите F9.

Borland Pascal оттранслирует Вашу программу, переведя ее с языка Паскаль (который понятен Вам) на машинный код 8086 для микропроцессора (который ваш персональный компьютер может выполнить). Вы не можете видеть машинный код 8086, поскольку он хранится в памяти (или на диске).

Паскаль, также как и обычный язык (например, английский) имеет определенные синтаксические правила, которые Вы должны соблюдать. Однако структура Паскаля не допускает неточностей и синтаксических ошибок. Компилятор всегда должен понимать, что

Вы имеете в виду. Если Вы используете в операторах неподходящие символы или строите их некорректно, компилятор Паскаля выводит во время трансляции программы сообщение о синтаксической ошибке.

С момента начала компиляции в середине экрана появляется прямоугольная рамка, в которой указываются сведения о происходящей компиляции. В рамке вспыхивает сообщение о необходимости нажать клавиши Ctrl-Break, если нужно прервать компиляцию. Если компиляция завершается успешно, то в рамке вспыхивает сообщение "Success: Press any key" ("Успешно! Нажмите любую клавишу"). Рамка остается видимой до тех пор, пока вы не нажмете какую-либо клавишу.

Если во время компиляции обнаруживается ошибка, то Borland Pascal прекращает компиляцию, помещает курсор в редакторе в позицию, где содержится ошибка, и распечатывает сообщение об ошибке в верхней части окна редактирования. Нажмите любую клавишу с тем, чтобы удалить сообщение об ошибке.
(Примечание: нажатие выбранной Вами клавиши используется редактором). Затем исправьте ошибку, сохраните на диске обновленный файл и оттранслируйте его заново.

После устранения всех обнаруженных ошибок возвратитесь в основное меню и для запуска программы выберете опцию "Выполнить" (Run).

Окно выходных данных увеличится до размера полного экрана, и на экране появится сообщение:

Введите два числа:

Введите два любых целых числа, разделенные пробелом, а затем нажмите "Enter". Появится следующее сообщение:

Отношение равно

вместе с числом, представляющим отношение первого числа ко второму. На следующей строке будет выведено сообщение "Нажмите клавишу <Enter>" и программа будет ожидать ввода клавиши <Enter>.

При завершении выполнения программы вы вернетесь в окно редактора. Для того чтобы просмотреть выведенную программой информацию, выберите "Запуск/Экран пользователя" (Run/User Screen) или нажмите Alt-F5.

ГЛАВА 3. ЭКРАН В ГРАФИЧЕСКОМ РЕЖИМЕ

Экран представляет собой прямоугольное поле, состоящее из большого количества точек. В графическом режиме (в отличие от текстового) мы имеем возможность изменять цвет каждой точки. Точки, окрашенные в различные цвета, могут образовывать линии, тексты и любые другие изображения. Количество цветов в худшем случае равно двум. Мы должны сказать нечто достаточно тривиальное для опытных программистов: дисплей может находиться либо в текстовом, либо в графическом режиме. Ситуация, когда часть экрана находится в графическом, а часть в текстовом режиме, невозможна (может быть, пока невозможна).

Дело в том, что все, что мы на экране видим, является результатом автоматического отображения содержимого видеопамяти.

В зависимости от того, какой режим в данный момент поддерживается, — текстовый или графический, — содержимое видеопамяти интерпретируется совершенно по-разному.

Видеопамять, контроллер лучевой трубки, порты ввода-вывода и т.д. располагаются на одной печатной плате, называющейся адаптером дисплея. Существует несколько типов дисплейных адаптеров. Различия между ними связаны в частности:

— с разрешающей способностью экрана, которую они в состоянии поддерживать;

— с максимальным количеством цветов, которое может быть одновременно показано на экране. Одни из наиболее известных адаптеров:

— CGA (Color Graphics Adapter);

— MCGA (Multi-Color Graphics Array);

— EGA (Enhanced Graphics Adapter);

— VGA (Video Graphics Arrey);

Какой бы адаптер не был установлен на нашем компьютере, мы можем использовать один и тот же набор процедур и функций Borland Pascal благодаря тому, что их конечная настройка на конкретный адаптер осуществляется автоматически. Эту настройку выполняют специальные программы, называемые графическими драйверами. Драйверы располагаются в файлах, имеющих расширение .BGI (Borland Graphics Interface). Например, драйверы для работы с адаптерами EGA и VGA находятся в файле EGAVGA.BGI, драйверы для работы с CGA и MCGA — в файле

CGA.BGI. Для указания различных драйверов в модуле Graph определены константы:

```
const
Detect = 0; { автоматическое определение драйвера }
CGA=1;
MCGA=2;
EGA=3;
EGA64=4;
EGAMono=5;
IBM8514=6;
HercMono = 7;
ATT400 = 8;
VGA=9;
PC327 = 10;
```

Адаптер данного типа часто способен поддерживать несколько графических режимов. Для указания этих режимов в модуле GRAPH, определены константы (приводится список констант для адаптеров CGA, EGA и VGA):

```
const
CGACO = 0 { 320 x 200,4 цвета, палитра 1 }
CGAC1 = 1 {320 x 200 ,4 цвета, палитра 1 }
CGAC2 = 2 {320 x 200 ,4 цвета, палитра 1}
CGAC3 = 3 {320 x 200,4 цвета, палитра 1}
CGAHi= 4 { 640 x 200 , 4 цвета, палитра 1 }
EGALo = 0 { 640 x 200, 16 цветов, 4 страницы}
EGAHi = 1 { 640 x 350, 16 цветов, 2 страницы }
EGA64Lo = 0 { 640 x 200,16 цветов, 1 страница }
EGA64Hi = 1 { 640 x 350, 4 цвета , 4 страница }
VGALo = 0 { 640 x 200 , 16 цветов, 4 страницы }
VGAMed = 1 { 640 x 350 , 16 цветов. 2 страницы }
VGAHi = 2 { 640 x 480 , 16 цветов, 1 страница }
```

3.1. Перевод экрана в графический режим

Обычным режимом для экрана является текстовый. Для того, чтобы перевести экран в графический режим, используется процедура модуля Graph InitGraph:

InitGraph(GD, GM, Path) — перевести экран в графический режим. GD — номер драйвера, GM — номер режима, Path — путь доступа к файлу, содержащему нужный драйвер. Если переменная Path

содержит просто пустую строку (Path="), то драйвер ищется в текущем каталоге.
И GD, и GM являются переменными-параметрами. Если при запуске InitGraph переменная GD равна нулю, то нужный драйвер и оптимальный графический режим для этого драйвера определяется автоматически. Для красоты в модуле GRAPH даже заведена константа Detect, равная 0. Обратите внимание на то, как она будет использована в примерах.

Симметричной процедурой к InitGraph является процедура **CloseGraph,** выгружающая драйвер из памяти и восстанавливающая исходный видеорежим.

Следующая программа инициализирует графический режим и сразу же его закрывает:

```
uses
Graph;
var
GDriver, G Mode: Integer;
begin
  GDriver.-=Detect; {автоматическое определение драйвера, пос-
кольку De1ect=0}
  I nitGrapfi(G Driver, GModc, 'u:\1p'); {инициализировать
  графический режим}
  ReadLn; {ждать нажатия клавиши "Enter" }
CloseGraph; {процедура CloseGraph выгружает графический драйвер
из памяти и восстанавливает предыдущий видео-режим.} end.
```

Возможны ситуации, когда по каким-либо причинам процедура InitGraph не может нормально выполнить свою работу, например, если неверно указан путь, где находится нужный .BG1 файл, или, если для инициализации графики не хватает оперативной памяти.

Для идентификации таких случаев можно использовать функцию GraphResult, возвращающую после попытки инициализации графики следующие значения:

0 — ошибок нет
-2 — нс обнаружена графическая плата;
-3 — не найден файл драйвера;
-4 — неверный драйвер;
-5 — для инициализации графики недостаточно памяти,

```
uses
Graph;
var
```

```
G Driver, GMode , ErrCode : Integer;
begin
   G Driver : = Detect; { автоматическое определение драйвера }
   InitGraph(G Driver, GMoue, "); { инициализировать
   графический режим}
   ErrCode := GraphResult;
if ErrCoue <> 0 then { ошибки есть }
WrileLnC Ошибки инициализации графики номер ',ErrCode)
else { ошибок нет }
CloseGraph: { выключить графический режим }
end.
```

3.2. Изображение точек, линий, цвета

Итак, мы умеем переводить экран в графический режим и хотим узнать, что же с ним еще можно делать. В модуле GRAPH находится около 80 процедур и функций. С их помощью можно рисовать точки, отрезки, эллипсы, прямоугольники, многоугольники, закрашивая все это в различные цвета и заштриховывая 11-ю стандартными и сколькими угодно нестандартными способами, можно выводить на экран тексты разными шрифтами, можно запоминать и передвигать области экрана. Все это довольно просто.

Начнем с системы координат. Каждая точка экрана имеет свои координаты. Верхний левый угол — точка с координатами (0,0). Координата x увеличивается слева направо, а координата y — сверху вниз. Например, в режиме VGAHi (640 x 480) координаты правого нижнего угла (639,479). А координаты середины экрана — (320,240). И если у Вас возникнет сильное желание поставить в середине экрана точку, сделайте это при помощи процедуры PutPixel.

Процедура **PutPixel (X, Y, Color)** закрашивает точку с координатами (X,Y) в цвет, определенный параметром Color.

Например, при вызове PutPixel(100,120,Red) появится красная точка с координатами (100,120).

Процедура PutPixel ставит в нужном месте точку нужного цвета, а с помощью симметричной ей функции GetPixel можно, наоборот, узнать, какой цвет имеет точка с данными координатами.

Функция **GetPixel(X, Y)** возвращает значение цвета в точке с координатами (X,Y).

Таким образом, если Col — целая переменная, то после выполнения оператора:

Col:=GetPixel (50,80);
Col будет иметь значение цвета в точке (50,80).

В модуле GRAPH имеются несколько процедур для рисования простых фигур: отрезков, окружностей, эллипсов, прямоугольников и т.д.

Процедура Line(Xl, Yl; X2, Y2) проводит отрезок из точки (XI,YD в точку (X2,Y2).

Процедура **Circle(X, Y, Radius)** чертит окружность с центром в точке 'X,Y) и радиусом Radius.

Процедура **Rectangle(Xl,Y),X2,Y2)** чертит прямоугольник с левым "верхним углом (X1,Y1) и правым нижним (X2,Y2).

Процедура SetColor(Color) устанавливает текущий цвет рисования.

Если процедура SetColor не установила какой-либо другой цвет, то текущим цветом является белый.

Для обозначения цветов в графическом режиме можно использовать те же константы, что и в текстовом:

```
const
Black = 0; { черный }
Blue= 1; {синий }
Green = 2; {зеленый }
Cyan = 3; {бирюзовый }
Red = 4; { красный }
Magenta = 5; { малиновый }
Brown = 6; { коричневый }
LightGray = 7; {светло-серый }
 DarkGray = 8; {темно-серый }
 LightBlue = 9; { ярко-голубой}
 LightGreen = 10; { ярко-зеленый}
 LightCyan = 11; { ярко-бирюзовый}
 LightRed = 12; { ярко-красный}
 LightMagenta = 13; { ярко-малиновый }
 Yellow =14; { желтый }
 White-IS; {белый}
```

Следующая программа, вычерчивая окружности, создает ажурный узор:

```
uses
 graph;
var
i, j: integer;
```

```
gd,gm:integer;
begin
gd: delect; { автоматическое определение драйвера }
initgraph( gd,gm,''); {инициализировать графический режим}
for i.-=0 to 20 do
        for j: =0 to 20 do
Circle(i*40,j 30,64);
readln;
CloseGraph; { выключить графический режим }
 end.
```

Мы часто будем использовать две удобные функции GetMaxX и GetMaxY, возвращающие разрешающую способность по горизонтали и вертикали (соответственно). |

Эти функции удобно использовать для написания программ, не зависящих от режима, в котором они будут работать. Следующая программа в любом видеорежиме чертит рамку точно на весь экран:

```
uses
   graph;
    var
   i,j:integer;
gd.gm: integer;
begin
   gd: =detect;
   initgraph(gd,gm,''); {инициализировать графический режим}
   Rectangle(0,0,GelMaxX,GelMax Y);
   readln;
   CloseGraph; {выключить графический режим}
    end.
```

3.3. Изображение фигур

Процедуры Line, Circle, Rectangle вычерчивают только линии. Borland Pascal предоставляет возможности для рисования заполненных цветом фигур. Подобно тому, как процедура SetColor устанавливает текущий цвет рисования, процедура SetFillStyle устанавливает текущий цвет наполнения и текущую штриховку:

SetFillStyle (Style, Color) —

установить текущий цвет наполнения Color и текущую штриховку (стиль наполнения). Для указания различных стилей наполнения можно пользоваться следующими константами:

```
const
EmptyFill = 0; {заполнить область фоновым цветом }
SolidFill = 1; { непрерывное заполнение области заданным цветом }
LineFill = 2; {заполнить толстыми горизонтальными линиями —}
LtSlasnFill = 3; {заполнить тонкими косыми линиями /// }
SlashFill = 4; { заполнить жирными косыми линиями /// }
BkSlashFill = 5; {заполнить жирными косыми линиями \\\ }
LtBRSlashFiH = 6; { заполнить косыми полосами \\\ }
HatchFill = 7; { заполнить клеткой }
XHatchFill = 8; { заполнить косой клеткой }
InterLeaveFill = 9; { заполнить очень частой косой штриховкой }
WideDotFill = 10; { заполнить редкими точками }
CloseDotFill = 11; { заполнить частыми точками }
UserFill = 12; { штриховка, определенная пользователем }
```

Процедура Bar(x1,y1,x2,y2) рисует прямоугольник, используя текущий цвет наполнения и текущую штриховку. Левый верхний угол прямоугольника — точка (x1,y1), а правый нижний — точка (x2,y2).

Процедура Bar3D(xl,yl,x2,y2,Depth,Top) рисует параллелепипед, используя текущие цвет наполнения и штриховку. Переменная Depth содержит "глубину" параллелепипеда. Логическая переменная Top сигнализирует, нужно ли вычерчивать верхнюю грань. Если Top имеет значение True, то верхняя грань вычерчивается, если Top имеет значение False, верхняя грань не вычерчивается.

Процедура FillEllipse(X, Y, XRadius, YRadius) рисует залитый цветом эллипс, используя текущий цвет наполнения и текущую штриховку. Оси эллипса параллельны осям координат. XRadius — ширина эллипса, а YRadius — его высота.

```
uses
Graph;
var
  GD,GM:integer
  begin
GD.=Detecf;
  InitGraph(gd,gm,'d:\tp'); {инициализация графики}
  SetFillStyle(7,Blue);
  Bar( 0.0, GetMaxX, GelMax Y);
  SetColor(Cyan);
  SelFillSlyle( 1 l,LightRed);
  FillEllipse(GelMuxX div 2,GelMaxY div 2,90.100);
ReadLn;
```

```
Closegraph;
end.
```

3.4. Набор текста в графическом режиме

В графическом режиме можно использовать растровый шрифт и, кроме него, еще несколько векторных шрифтов. Растровый шрифт задается матрицей точек, а векторный — рядом векторов, составляющих символ.

При увеличении растрового символа мы начинаем различать составляющие его точки, а при увеличении векторного символа качество изображения не меняется и зависит только от разрешающей способности экрана.

Файлы шрифтов имеют расширение .CHR. При загрузке шрифта соответствующий файл должен находиться либо в том же каталоге, где и графический драйвер .BGI (этот каталог, как Вы помните, задается в процедуре InitGraph), либо в текущем каталоге.

Масштабирование и выбор шрифта осуществляется с помощью процедуры SetTextStyle:

SetTextStyle(Font, Direction, Size) — установить текущий шрифт, направление вывода текста и размер символов. Font определяет шрифт, Direction — направление вывода текста (слева направо или снизу вверх), Size определяет размер шрифта. Для растрового шрифта нормальный размер достигается при Size =l, а для векторных шрифтов — при Size =4.

Для указания различных шрифтов и направлений вывода текста определены следующие константы:

```
const
    {шрифты}
    DefaultFont = 0 {8x8 стандартный растровый шрифт}
    TriplexPont = 1 { векторный шрифт}
    SmallFont = 2 {векторный шрифт}
    SansSerifFont= 3 { векторный шрифт}
    GothicFont = 4 {векторный шрифт}
    {ориентация текста }
    HorizDir = 0 { слева направо}
VertDir = 1 { снизу вверх }
```

Процедура OutTextXY(X, Y. TextString) — выводит строку TextString, начиная с точки (X,Y). Строка выводится текущим шрифтом, с текущими направлением и размером символов.

Процедура SetTextJustif y (Horiz, Vert) устанавливает автоматическое выравнивание текста, используемое затем процедурами OutTextXY и OutText. Horiz — горизонтальное, а Vert — вертикальное выравнивание.
Для указания выравнивания определены константы:

```
const
    {для горизонтального выравнивания могут использоваться:}
    LeftText = 0 { выравнивание слева}
    CenterText = 1 {выравнивание по центру }
    RightText = 2 { выравнивание справа}
    {для вертикального выравнивания могут использоваться:}
    BottomText = 0 {выравнивание снизу }
    CenterText = 1 {выравнивание по центру }
    TopText = 2 {выравнивание сверху }

program Game_0ver;
uses
 Graph;
var
GD,GM: Integer
begin
GD.=Detect;
InitGraph(GD,GM,'d:\tp\bgi');{инициализация графики}
SetTextJustify(CenterText,CenterText); {определить выравнивание}
SetTextStyle(TriplexFont,HorizDir,8); {определить шрифт, ни правление и размер}
{с небольшим смещением и различными цветами выводится одна и та же надпись}
SelColor(White);
OutTkxtXYfGetMaxX div 2.GetMaxY div 2; GAME OVER' );
SetColor(LightBlue);
OutTextXY(GetMaxX div 2+2,GetMaxY div 2; GAME OVER');
SetColor(LightRed);
OulTextX Y(GetMaxX div 2+3, GelMax Y div 2, ' GAME 0 VER');
SetColor(LightRed);
OutTextX Y(GelMaxX div 2+4, GelMax Y div 2, ' GAME 0 VER' );
SelColor( White);
read In;
CloseGraph;
end.
```

Векторные шрифты Triplex, SmallFont, SansSerif и Golhic не содержат русских символов, однако, в упоминавшемся уже программном продукте фирмы Borland "BGI TOOLKIT" содержится редактор шрифтов, который позволяет создавать новые векторные шрифты или менять отдельные символы в уже имеющихся.

3.5. Области экрана

Используя процедуры Getlmage, Putlmage и функцию ImageSize, мы имеем возможность запоминать и выводить на экран прямоугольные области изображений.

Функция ImageSize(xl,yl,x2,y2) возвращает размер памяти в байтах, необходимый для сохранения заданного прямоугольного фрагмента экрана, (x1,y1) левый верхний, а (x2,y2) правый нижний углы фрагмента.

Процедура Getlmage(xl,yl,x2,y2, Area) сохраняет прямоугольный фрагмент изображения в области памяти Area. (xl,yl) левый верхний, (x2,y2) правый нижний углы фрагмента.

Процедура Putlmage(x,y, Area, Mode) выводит в заданное место экрана фрагмент изображения. (x,y) — левый верхний угол области экрана, куда будет скопировано изображение из области памяти Area.Mode — режим вывода изображения на экран. Изображение может просто налагаться на данную область экрана, закрывая ее прежнее содержимое, либо может определенным образом взаимодействовать с ним.

Для описания различных режимов вывода используются константы:

```
  const
{ константы для процедуры Putlmage }
NormalPut=0; { замена существующего изображения }
XorPut = 1; { исключающее ИЛИ ( XOR )}
OrPut " 2; { логическое ИЛИ ( OR ) }
AndPut =3;{ логическое И (AND) }
NotPut = 4; { логическое НЕ ( NOT ) }
```

В режиме NormalPut каждая точка выводимого на экран изображения просто перекрывает ту точку, на которую она налагается. В остальных режимах каждый бит двоичного представления цвета налагаемой точки взаимодействует с парным битом цвета той точки, на которую происходит наложение по соответствующему логическому закону. Результатом может быть

цвет, отличный от цветов обеих исходных точек. Для того, чтобы немного освоиться с возникающими эффектами, мы предлагаем поэкспериментировать с данной программой:

```
NornalPut XorPut OrPut ftndPlit
uses
Crt, Graph;
var
GraphDriver, GraphMode: Integer;
P: Pointer;
   i, size: Integer;
begin
GraphDriver := Detect;
InitGraph( GraphDriver, GraphMode, 'd:\tp\bgi');
Set{FillStyle(l,14);
Bar(0,0, GetMaxX, GetMax Y);
{ создается некоторая картинка в прямоугольнике (0,0,40,40)}
selcolor( 1);
for i.=0 to 10 do
rectangle( 20-2*i,20-2*i.20+2*i,20+2*i);
OutTextXY( 80.20,'This is the image');
{определяется размер памяти, необходимый для запоминания
фрагмента (0,0,40,40)}
size.=ImageSize(0,0,40,40);
 {выделяется требуемая область памяти и на нее адресуется
 указатель P}
 GetMem(P, size);
 {в той области, на которую указывает P, запоминается
 фрагмент (0,0,40,40)}
 GetImage(0,0,40,40,P^);
 { вывод в режиме NormalPut}
for i.=l to 10 do
PutImage(i*20,60,P'',NormalPul);
 OutTextXY( 250,80; NormalPuf);
 { вывод в режиме XorPut}
 for i.=l to 10 do
 PutImage(i*20,^00,P^\,XorPut);
 OutTextX Y( 250,120, 'XorPut );
{ вывод в режиме OrPut }
 for i.=1 to 10 do
PutImage(i*20,140,P^,OrPut);
```

```
OutTextXY( 250,160;OrPuf);
{ вывод в режиме AndPut}
for i=1 to 10 do
PutImage(i*20,]80,P^,AndPut);
OutTextXY( 250,200;AndPut);
f^eadLn;
CloseGraph;
end.
```

3.6. Установка палитры цветов

Палитра — это соответствие между номерами цветов и цветами, которые реально появляются на экране. Мы рассмотрим три процедуры, позволяющие работать с палитрой.

Процедура **SetPalette(Coll, Col2)** устанавливает цвет палитры с номером Col 1 в цвет, указанный Col2.

```
    uses
Graph:
var
Driver, Mode: Integer;
begin
  Driver:=Detect;
  InitGraph( Driver, Mode, ' d:\ tp\bgi');
  SetPalette( Black, Blue);
ReadLn;
SetPalette(Black,Red);
ReadLn;
CloseGraph;
end.
```

Процедура *SetAllPalette(Palette)* устанавливает все цвета палитры одновременно. По адресу Palette должна находиться область, описывающая палитру, причем в первом байте указывается длина палитры, следом располагаются цвета. Переменную Palette часто удобно описывать, как имеющую предопределенный тип PalelleType:

```
type
PalelteType=record
Size:Byte;
    Colors.-Array[0... MaxColors] of ShortInt;
    end;
```

где MaxColors — предопределенная константа, равная 15.

В следующей программе при помощи SctAllPalette цвета циклически переходят один в другой, что используется для имитации циклического движения.

```
 uses Crt,Gruph;
     var
        Gd.Gm, I: integer;
     P:PulelleType;
     begin
    Gd.=Delect;
        lnitGruph(Gd,Gm.'d:\tp\bgi');
{закрасит экран белым цветом}
Bar(0,0, GelMaxX, Get Max Y);
{установить цвета для палитры}
for I:=0 to MaxColors do P.Colors [I ]:=I
P. Size.=MaxColors;
{рисовать последовательно всеми цветами ( кроме белого ) }
for I.=0 to MaxColors-1 do
begin
SetFillStyle(1,I);
Bar(50+I*10,0,50+(I+1)*l0.GetMaxY);
end;
{менять циклически цвета}
repeat
for i.=0 to MaxColors-1 do
P.Colors[I]:=(P.Colors[ I ]+l) mod MaxColors;
SetAllPaletle(P);
until keypressed;
readin;
CloseGraph;
end.
```

Каждый цвет может быть представлен, как Комбинация красной, зеленой и синей составляющих. Процедура SetRGBPalette позволяет изменить составляющие каждого цвета:

SetRGBPalette(Col, R, G, B) — изменить красную, зеленую и синюю составляющие цвета с номером Col на R, G и в соответственно. Поскольку учитываются только шесть бит чисел ^ G и B, имеет смысл указывать значение составляющих цвета в пределах от 0 до 63. Процедура SetRGBPalelte работает на адаптерах VGA и IBM 8514.

```
uses Crt, Graph;
    var
```

```
Gd,Gm:integer;
begin
      Gd:=Defect;
  InUGraph(Gd,Gm,'d:\tp\bgi');
  { черный экран }
  ReadLn;
{молочно-розовый оттенок — красная составляющая чуть-чуть преобладает } Set RGB Palettef Black, 55,45,45);
ReadLn;
{фиолетовый цвет — доминируют синяя и красная составляющие}
SetRGBPalette( Black, 30,5,30);
ReadLn;
CloseGraph;
end.
```

3.7. Обработка ошибок

Уже упоминалось, что при инициализации графического режима ошибки инициализации возвращаются функцией **GraphResult.** Однако, назначение GraphResull шире: эта функция возвращает код ошибки и при многих других графических операциях. Следующие константы соответствуют кодам возврата GraphResult:

```
const
{ Коды возврата функции GraphResult}
grOk = 0; { ошибок нет }
grNoInitGraph = -1; {(BGI) графика не установлена (используйте процедуру InitGraph) }
grNotDetected = -2; {аппаратное обеспечение для графики не найдено}
grFileNotFound = -3; { не найден файл драйвера устройства }
grInvalidDriver= -4; {неверный файл драйвера устройства }
grNoLoadMem = -5; { нехватка памяти для загрузки драйвера }
grNoScanMem = -6; { выход за границы памяти при сканировании области}
grNoFloodMem = -7; { выход за границы памяти при заполнении закрашиваемой области}
grFontNotFound = -8; { не найден файл шрифта }
grNoPontMem = -9; { нехватка памяти для загрузки шрифта}
grInvalidMode = -10;{недопустимый для выбранного драйвера графический режим}
```

3.8. Процедуры и функции модуля GRAPH

Процедура Arc — рисует дугу окружности.

Описание: Arc(X, Y : integer; StAng, EndAng, Radius : word);

X, Y — координаты центра окружности, StAng и EndAng — соответственно начальный и конечный углы, Radius — радиус окружности.

```
uses Crt.Graph;
var
   Gd, Gm, I:integer
   begin
   Gd:=Delect;
   InitGraph(Gd,Gm,'d:\fp\bgi');
   {серый фон }
   SetBkColor(LighlGray);
{дуги окружностей постепенно поворачиваются и расширяются, меняя цвет }
for I:=1 to 200 do
begin
SetColor(l div 15);
Arc(GetMaxX div 2,GetMaxY div 2, I,I+300,I+10);
end;
ReadLn;
CloseGraph;
end.
```

Процедура Bar — рисует заполненный цветом прямоугольник, используя текущий цвет и стиль закраски.

Описание: Bar(Xl,Yl,X2,Y2: integer);

X I и Y 1 — координаты левого верхнего, а X2 и Y2 — правого нижнего угла прямоугольника.

Процедура Bar3D — рисует заполненный цветом параллелепипед, используя текущий цвет и стиль закраски.

Описание: Bar3D (X1,Y1,X2,Y2 :integer;Depth:Word;Top:Boolean);

XI и Yl — координаты левого верхнего, а X2 и Y2 — правого нижнего угла прямоугольника. Depth — "глубина" , Top — логическая переменная, указывающая рисовать или не рисовать верхнюю грань.

```
uses Crt,Graph;
const
```

```
   Heigh=10;
   Width=10;
Depth=20;
var
Gd, Gm, I: integer;
begin
   Gd:=Detect;
   lnitGraph(Gd,Gm,'d:\tp\bgi' );
   SetColor(LightRed);
for I:=1 to 50 do
Bar3D(I*Width,I*Heighl,(I+l)*Width,(I+l)*Height,Depth, TopOn);
   ReadLn;
CloseGrciph;
end.
```

Процедура Circle — рисует окружность.

Описание: Circle(X,y : integer; Radius :word);

X, Y — координаты центра окружности, Radius — радиус.

Процедура CloseGraph — сбрасывает графический режим.

Описание: CloseGraph;

Процедура DrawPoly — рисует многоугольник.

Описание: DrawPoly (NumPoints : word;var PolyPoints);

NumPoints— количество вершин многоугольника, PolyPoints может ;
быть массивом, где перечисляются координаты точек.

```
uses Graph;
const
N=100; { количество вершин многоугольника}
var
   Gd, Gm, I:Integer;
Poly: Array [1.. 100] of PointType;
begin
   Gd:=Detect;
   InitGraph(Gd,Gm,'d:\tp\bgi');
   Randomize;
   {вершины, будут иметь случайные координаты }
   for I:=l to N do
begin
Poly[I ].X:=Random(GetMaxX);
Poly[I].Y.=Random(GetMaxY);
   end;
```

```
   DrawPoly(N,Poly);
   ReadLn;
CloseGraph;
end.
```

Процедура Ellipse — рисует дугу эллипса.

Описание: **Ellipse(X',** Y **:** integer , StAngle,EndAngle : word; XRadius,YRadius : word);

X,Y — координаты центра, StAngle и EndAngle — начальный и конечный угол дуги, XRadius и YRadius высота и ширина соответственно.

```
{ Дуга эллипса. "дрейфует", иногда меняя цвет}

uses Crt,Graph;
var
Gd.Gm.X, Y: integer;
begin
Cd: =Detect;
InitGraph(Gd,Gm,'d:\tp\bgi');
{синий фон}
 SetBkColor(Blue);
{ сначала центр дуги в центре экрана }
X:=GetMaxX div 2; Y:GetMax Y div 2;
repeat
{ с вероятностью 1/100 поменять цвет на случайный }
 ifrandom( 100)= I then SetColor( Random( 16));
{ Random(5)-2 — это просто случайное число от -2 до2 }
X:=X+ Random(5)-2; Y:=Y+Random(5)-2;
{ рисовать дугу эллипса }
Ellipse(X,Y,340,200,50,40);
until keypressed;
ReadLn;
CloseGraph;
end.
```

Процедура FillPoly — рисует закрашенный многоугольник.

Описание: FillPoly(NumPoints : word: var PolyPoints);

NumPoirtts — количество вершин многоугольника, PolyPoints может быть массивом, где перечисляются координаты точек. См. пример для DrawPoly.

Процедура FloodFill — заливает цветом ограниченную область, используя текущий цвет и стиль закраски.

Описание: FloodFill (X.Y : integer; Bordcr:word);
X,Y— координаты точки, из которой производится заливки, Border указывает, какой цвет при заливке считать граничным.

```
uses Graph;
var
Gd,Gm: Integer;
begin
Gd:=Detect;
InitGraph(Gd,Gm,'d:\tp\bgi');
SelColor(LightRed);
Circle( 100,100,80);
Circle(200,100,80);
   SetFillStyle( 1, Cyan);
FloodFill (150, lOO.LightRed);
ReadLn;
CloseGraph;
end.
```

Процедура GetArcCoords — запрашивает координаты последней процедуры ARC.
Описание: GetArcCoords (var ArcCoords : ArcCoordsType.);
При вызове GetArcCoords по параметрам последнего вызова процедуры Arc заполняются поля переменной ArcCoords, имеющей тип ArcCoordsType.

```
type
ArcCoordsType=record
 X,Y,
XStart, YStart,
XEnd, YEnd:Integer;
end;
```

X и Y — координаты центра окружности, XStart и YStart — координаты начальной, а XEnd и YEnd — конечной точки дуги.

```
uses Graph;
var
   Gd,Gm,I,J,Angle:Integer;
ArcCoord:ArcCoordsType;
begin
   Gd:=Detect;
   InitGraph(Gd,Gm,'d:\tp\bgi');
   Angle:=30;
for I:= 1to 12 do
```

```
for J:= l to I do
begin
Arc( GetMaxX div 2,GelMaxY cliv 2,I*Angle,J*Angle, 100);
{ запоминаем параметры, дуги в ArcCoord }
GelArcCoords(ArcCoord);
{ соединяем крайние точки дуги }
Li:ne(ArcCoord. XStart, ArcCoord. YStart, ArcCoord. XEnd .ArcCoord.
YEnd);
end;
ReadLn;
CloseGraph;
end.
```

Функция GetColor — возвращает текущий цвет.

Описание: GetColor: word;

Функция GetGraphMode — возвращает текущий графический режим**.**

Описание: GetGraphMode : integer;

Процедура Getlmage — сохраняет образ заданной области экрана в
буфере.

Описание: GetImage(Xl,Yl,X2,Y2 : integer; varArea).

```
uses  Crl, Graph;
var
Gd, Gn:, Size: integer;
P:=Pointer,
begin
Gd:-=Detect;
InitGraph( Gd, Gm, ' d:\ tp\bgi');
{в области экрана (0,0,40,40) рисуется картинка}
SetFillStyle( lO.LightGreen);
Bar( 0,0,40,40);
Rectangle(0,0,40,40);
{ Size станет равной числу байтов, требуемому для запоминания
области экрана (0,0,40,40')}
Size:=ImageSize(0,0,40,40);
{выделяется область памяти в Size байт; на нее указывает P}
GetMem(P,Size);
{область экрана (0,0,40,40) запоминается в той области памяти, на
которую указывает P }
```

```
Gellmage(0,0,40,40, P^);
{запомненное изображение выводится в точки со случайными координатами до тех пор, пока не будет нажата какая-нибудь клавиша}
repeat
PutImage(Random(GetMaxX),Rundom(GetMaxY),P^,NonnalPut);
until keypressed;
ReadLn;
CloseGraph;
end.
```

Функция GetMaxColor — возвращает максимальное значение цвета, которое можно передать процедуре SetColor.

Описание: GetMaxColonWord;

Функция GetMaxX — возвращает для текущего драйвера и режима разрешение (количество точек) по горизонтали.

Описание: GetMaxX:Integer;

```
uses Graph;
var
Gd,Gm: Integer;
begin
  Gd:Detect;
  InUGraph(Gd, Gm, ' d:\tp\bgi');
  { зеленая точка в центре экрана }
  PutPixeK GetMaxX div 2, GetMax Y div 2, LighlGreen);
  ReadLn;
CloseGraph;
end.
```

Функция GetMaxY — возвращает для текущего драйвера и режима разрешение (количество точек) по вертикали.

Описание: GetMaxY:Integer;

См. примеряла GetMaxX.

Функция GetPixel — возвращает цвет данной точки.

Описание: GetPixel(X,Y:Integer):Word;

```
uses Crt,Graph;
var
Ged,Cm,X, Y: integer;
I:Longlnt;
begin
Gd:=Detect;
InitGraph(Gd,Gm;d:\tp\bgi');
```

```
SetTextSlyle(DefaultFonl,HorizDir,7);
SelColor(DarkGray);
OutTextX Y( 50,50, ' Frank, ');
OutTextXY(50,150; thank you !');
for I:=1 to 50000 do
begin
X:=Random( GetMaxX);
Y:=Random( GetMaxY):
If GelPixel(X, Y)=DarkGray then PutPixel(X, Y, 14);
end;
ReadLn;
CloseGraph;
end.
```

Функция GraphErrorMsg — возвращает строку сообщения об ошибке для заданного кода ошибки.

Описание: GraphErrorMsg(Code:Integer):String;

Функция GraphResult— возвращает код ошибки последней графической операции.

Описание: GraphResult:Integer;

Функция ImageSize — возвращает число байтов, требуемых для сохранения данной области экрана. Описание: ImageSize(Xl,Yl,X2,Y2:Integer):Word;

См. пример для Getlmage.

Процедура InitGraph — инициализирует графический режим.

Описание: InitGraph (var Driver,Mode:Integer;Path:String);

Процедура Line — рисует отрезок, соединяющий две данные точки.

Описание: Line(Xl,Yl,X2,Y2:Infeger);

```
uses Crt,Graph;
var
Gd,Cm: integer;
begin
Gd:=Detect;
InitGraph(Gd,Gm;d:\tp\bgi');

{До нажатия произвольной клавиши проводить прямые из центра экрана  ( GetMaxX div 2, GetMax Y div 2 ) до случайной точки экрана}
repeat.
{изредка ( с вероятностью 1/2000) менять цвет}
```

```
if Random(2000)=I then SetColor(Random(16));
Line( GetMaxX div 2, GetMax Y div 2,
Random( GetMaxX),Random( GetMaxY));
until keypressed;
ReadLn;
CloseGraph;
end.
```

Процедура **LineRel** — рисует отрезок до точки, заданной относительно текущего указателя.

Описание: LineRel (DX,DY:Integer);

Примечание: указатель смещается на вектор (DX,DY).

```
uses Crt,Graph;
var
Gd,Gm,I:integer;
begin
   Gd:=Detect;
   InitGraph( Gd, Gm, ' d: \ tp \ bgi'):
   repeat
{Передвинуть указатель в случайную точку экрана}
MoveTo(Random(GetMaxX),Random(GetMaxY));          {установить случайный цвет}
SetColor(Random( 16));
{рисовать ломаную; Random( 5)-2 — это просто случайное число от -2 до 2; поэтому очередная вершина ломаной немного смещается относительно предыдущей }
for I:=l to 100 do
{ отрезок от текущего указателя до близко-лежащей точки }
 LineRel(Random(5)-2,Random(5)-2);
until keypressed;
ReadLn;
CloseGraph;
end.
```

Процедура LineTo — рисует отрезок от текущего указателя до данной точки. Описание: LineTo (X,Y:Integer);

Процедура MoveRel — перемещает текущий указатель на заданное смещение от текущей позиции..

Описание: MoveRel (DX,DY:Integer);

Процедура MoveTo — перемещает текущий указатель в данную точку.

Описание: MoveTo(X,Y:Integer);

Процедура OutText — выводит строку , начиная с текущего указателя.

Описание: OutText(S:String);

Процедура OutTextXY — выводит строку , начиная с данной точки

Описание: OutTextXY(S:String);

```
uses Crt,Graph;
var
Gd,Cm, I:integer;
begin
   Gd:=Detect;
   InitGraph(Gd,Gm:d:\tp\bgi');
{Автоматическое выравнивание текста}
SetTextJustify(CenterText, CenterText);
SetColor( LightCyan);
SetTextStyle(SmallFont,Horiz.Dir, I);
{режим рисования прямых линий XorPul}
Set WriteMode(XorPut);
{основной цикл; благодаря режиму рисования прямых XorPut, надпись при повторном выведении стирается}
repeat
for I:=1 to 30 do
begin
SetUserCharSized(I, 2,1.2);
OutTextX Y(GelMaxX div 2, GelMax Y div 2+I, ' How are your?'):
Delay(20);
OutTextXY(Ge1MaxXdiv2, GetMaxYdiv2+I:Howare your?');
end;
for I:=30  down to 1 do
begin
SetUserCharSized (I,2,I,2);
OutTextX Y(GetMaxX div 2, GetMax Y div 2+I,’I am well!'):
Delay(20);
OutTextX Y(GetMaxX div 2, GetMax Y div 2+I, ' I am well!');
end;
until keypressed; { пока не будет нажата к.-л. клавиша}
ReadLn;
CloseGraph;
end.
```

Процедура PieSHce — рисует сектор.

Описание: PieSlice(X,Y : integer; StAng, EndAng, Radius : word);

Процедура Putlmage — выводит на экран образ заданной области экрана.

Описание: Putlmage(X,Y, var Area, Mcthod:Word).

См. пример для Getlmagc.

Процедура PutPixel — рисует точку в заданном месте экрана.

Описание: PutPixel (X ,Y: Integer,Color:Word).

См. пример для GetPixel.

Процедура Rectangle — рисует прямоугольник.

Описание: Rectangle(Xl,Yl,X2,Y2:Integer);

{По экрану, отражаясь от его границ, двигается прямоугольник; используются свойства вывода прямых в режиме XORPut}

```
uses
Crt,Graph;
var
{(X, У) — координаты центра, DX, DY— шаги по горизонтали и вертикали, W,H — половины ширины и высоты}
Gd.Gm.X, Y,
 DX,DY,W,H: Integer;
{щелчок}
procedure Click;
begin
Sound(1000); Delay(5); Nosound;
end;
begin
Gd:=Detect;
InitGraph(Gd,Gm,'d:\tp\bgi');
SetWriteMode(XORPul);
X:=l00; Y:=100;DX:=4;DY:=4; W:=15:H:=l5;
repeat
X:=X+DX; Y:=Y+DY;
{при выходе за границы экрана изменить направление  движения и щелкнуть } if(X>GelMaxX) OR (X<0) then begin Click; DX:=-DX; end;
if (Y>GetMax Y) OR (Y<0) then begin Click: DY:=-DY; end;
Rectangle(X-W,Y-H,X+W,Y+H);
Delay(50);
Rectangle(X-W,Y-H,X+W,Y+H);
PutPixel(X,Y.Yellow);
until keypressed; { пока не будет нажата к.-л. клавиша}
```

```
ReadLn;
CloseGraph;
end.
```

Процедура RestoreCRTMode —восстанавливает режим экрана, установленный при инициализации графики.

Описание: RestoreCRTMode;

Процедура SetAllPalette — изменяет все цвета палитры одновременно. Описание: SetAllPalette(var Palette);

Процедура SetBkColor — устанавливает текущий цвет фона.
Описание: SetBkColor(Color:Word);

Процедура SetColor — устанавливает текущий цвет рисования.
Описание: SetColor(Color: Word);

Процедура SetPillStyle — устанавливает цвет и стиль заполнения.
Описание: SetFillStyIe(Style,Color:Word);

Процедура SetGraphMode — устанавливает заданный графический режим и очищает экран.
Описание: SetGraphMode(Mode:Integer);

Процедура SetPalette — изменяет заданный цвет палитры.
Описание: SetPalette(Coll:Word; Col2:ShortInt);

Процедура SetGRBPalette — изменяет красную , синюю и зеленую составляющие данного цвета (для драйверов IBM-8514 и VGA).
Описание: SetRGBPalette(Col, R, G, B.-Integer);

Процедура SetTextJustify — устанавливает значения для выравнивания текста.
Описание: SetTextJustify (Horiz, Vert: Word);

Процедура SetTextStyle — задает текущий шрифт, ориентацию и размер текста.
Описание: SetTextStyle(Fonl, Direction, Size: Word);

Процедура SetUserCharSize — настраивает для векторных шрифтов высоту и ширину символа.
Описание: SetUserCharSize(Xl,Yl,X2,Y2:Word);

```
uses Crt.Graph;
var
Get, Gm, I:integer;
begin
Gd:=Detect;
InitGraph(Gd,Gm,'d:\tp)\bgi');
```

```
SetTextJustify(CenterText.CenterText);
OutTextX Y(GctMaxX div 2, GetMax Y div 2, ' How are your?');
Delay(1000):
SetColor(LightCyan);
SetTextSlyle(SmallFont,HorizDir,l);
for I:=20 downto 15 do
begin
   SetUserCharSiz.e( 1,2,1,2);
   OutTextXY(GetMaxX div 2, GetMaxY div 2+I, 'I am well!');
end;
ReadLn;
CloseGraph;
end.
```

Процедура SetWriteMode — устанавливает режим рисования линий (просто копирование, или XOR).

Описание: SetWriteMode(Modc:Integer);

См. пример для Rectangle. Интересно, что эта процедура влияет так же на режим рисования векторных шрифтов (см. пример для OutTextXY).

Функция TextHeight — возвращает высоту текста.

Описание: TextHeight(S:String) :Word;

Функция TextWidth — возвращает ширину строки.

Описание: TextWidth (S:String) :Word;

3.9. Программа меню в графическом режиме

Нами уже была описана программа, строящая меню. Сейчас будет приведен ее вариант, работающий в графическом режиме. Поскольку весь вывод на экран был сосредоточен в процедурах Draw_Menu и New_Menu, то для модификации программы потребовалось изменить только эти две процедуры (и еще понадобилась легкая косметическая обработка процедуры Init). Кроме того, чтобы внести некоторое разнообразие, мы изменили содержание процедуры Do_Select, получающей управление при нажатии клавиши " Enter", т.е. при выборе того или иного пункта меню. Процедура Do_Select теперь заставляет звучать ноту, соответствующую номеру пункта меню. Мы надеемся, что легкомысленность приложения, для которого мы использовали программу меню, не помешает Вам, при

случае, использовать ее для "серьезных" целей. Как и прежде , начнем с описания переменных и констант.

Константы:

константа nmenu обозначает количество строк (пунктов) меню;
D0_, RE , MI , FA , SOL, LIA, S - частоты нот первой октавы;
В программе используются глобальные переменные:
- ch : символьная переменная будет содержать код последней нажатой клавиши;
- **functionkey** : логическая переменная; имеет значение True, если была нажата специальная клавиша, и False в противном случае;
- **strmenu:** массив строк от 0 до nmenu-1 , содержащих текст соответствующих пунктов меню;
Все остальные переменные будут описаны , как целые числа:
- text_color: цвет текста на неподсвеченных "шпалах" меню;
- back_color: цвет фона на неподсвеченных "шпалах" меню;
- select_text_color: цвет текста на подсвеченной "шпале" меню;
- select_back_color: цвет фона на подсвеченной "шпале" меню;
- screen_back_color: цвет экрана;
- select_framc_color: цвет контура подсвеченной "шпалы";
- frame_color: цвет контура "шпалы";
- width : ширина меню;
- height: высота "шпалы" меню;
- **depth:** глубина "шпалы" меню;
- **wx** : столбец левого верхнего угла меню;
- **wy** : строка левого верхнего угла меню;
- **dy** : промежуток между "шпалами" в строках;
- **crm:** номер "шпалы" , подсвеченной в данный момент;

Вот краткое описание процедур, построенных в программе:
- процедура **Init** устанавливает значения цветов, положение и ширину меню, расстояние между строками и т.д.;
- процедура Draw_Menu рисует меню;
- процедура New_Menu(old,sel:intcgcr) обновляет изображение при движении подсветки;
- процедура **Up** обрабатывает нажатие клавиши "стрелка вверх";
- процедура **Down** обрабатывает нажатие клавиши "стрелка вниз";;
- процедура **Do_Select** обрабатывает нажатие клавиши "Enter";;
- процедура **Do_Command** выполняет диспетчерские функции , передавая управление процедурам Up, Down , или Do_Select в зависимости от нажатой клавиши;

Program Menu;

```
uses Crt,Graph;
const
   nmenu=7; {количество пунктов меню}
D0_=262;
RE =294;
Ml = 330;
FA=349;
SOL=392;
LI A= 440:
S = 494;
var
ch: char;
functionkey: boolean;
GD.GM ; Integer;
width, height, depth, text_color, back_color, select_text_color,
select_frame_color,frume_color,
  selecl_back_color,screen_back_color,
  wx, wy, dy, i, crm:integer;
  strmenu:array [0.. nmenu-1] of string [40];
{инициализируются значения переменных}
Procedure Init;
begin
  GD:=Detect;
  lnilGraph(GD,GM:'d:\tp\bgi'):
  {устанавливаются цвета}
  text_color:=Black;
  back_color:=LightBlue;
  select_texl_color:=White;
  select_frame_color:= Yellow;
  frame_color:= White;
  select_back_color:=LightRed;
  screen_back_color:'=LightGray;
  dy:=20; {расстояние между строками}
  wx:=15; { координата х левого верхнего угла }
  wy:=35; { координата у левого верхнего угла }
  width:=250; { ширина меню }
  heights:=18; {высота "шпалы" меню}
depth:=10; {глубина "шпалы" меню} crm:=0; { номер строки меню,
которая будет подсвечена первоначально ( нумерация начинается с
нуля )}
```

```
strmenu[0]:='ДО';
strmenu[l]:='РЕ';
strmenu[2]:='МИ';
strmenu[3]:='ФА ';
strmenu[4]:=' СОЛЬ ';
strmenu[5]:='ЛЯ ';
strmenu[6]:='СИ ';
end;
{процедура первоначально выводит на экран меню }
 Procedure Draw_Menu;
var  i,j,y:integer;
begin
SetFillStyle( I, screen _back_color);
Bar(0.0,GetMaxX,GctMax Y);
SetTextJustify(CenterText,CenterText);
for i:=0 to nmenu-I do
if i<>crm then
begin
y:=wy+i*( heighi+dy);
SetColor( frame_color);
SefFillStyle( 1,back_color);
Bar3D(wx,y,wx+width,y+height,depth,TopOn);
SetColor( text_color);
OutTextX Y(( wx+width) div 2,y+( height div 2),strmenu[i]);
end
else
begin
y:=wy+i* (height+dy);
SetColor( select_frume_color);
Set FillStyle( I, select_back_color);
Bar3D(wx,y,wx+width,y+heigbt,deplh,TopOn);
.SelColor(select_text_color);
OutTextXY((wx+width) div 2, y+(height div 2},strmenu[i]);
end;
end;
{обновляет изображение меню, соответственно значениям old -
старый номер текущей ( подсвеченной ) строки, set - новый номер
текущей ( подсвеченной ) строки. }
Procedure New_Menu (old,sel: integer);
var
```

```
y:Integer;
begin
        y:=wy+old*( height+dy);
    SetColor( frame_color);
    SetFi!lStyie( I, back_color);
    Bar3D( wx,y, wx+width,y+height,depth. TopOn );
    SetColor(text_color);
     OutTextX Y((wx+widyh) div 2, y+( height div 2),strrmenu[old]);
   Y:= wy+sel*( height+dy);
   SetColor( select_frame_color):
   SetFillStyle( I,select_back_color):
Bar3D(wx,y, wx+width,y+height,depth, TopOn):
. SetColor( select_text_color):
()utTextXY((wx+width) div 2, y+( height div 2),strmenu[sel]);
end;
{Реакция на нажатие клавиши "стрелка вверх" }
 Procedure Up;
var old: integer;
begin
   old:'=crm;
   crm:=crm-I;
   if crm<0 then crm:=nmenu-l;
New_Menu(old,crm);
end;
{ Реакция на нажатие клавиши "стрелка вниз" }
 Procedure Down;
var old: integer;
begin
   old:=crm;.
   crm:=crm+l;
   if crm >=nmenu then crm:=0;
New_Menu( old, crm);
end;
{процедура получает управление при нажатии клавиши Enter}
procedure Do_Select(crm:Integer);
begin
if crm=0 then
begin Sound(DO_); Delay(300); Nosound; end;
if crm=l then
begin Sound(RE); Delay(300); Nosound; end;
```

```
if crm=2 then
begin Sound(MI); Delay(300); Nosound; end;
if crm=3 then
begin Sound(FA); Delay(300); Nosound; end;
if crm=4 then
begin Sound(SOL); Delay(300); Nosound; end;
if crm=5 then
begin Sound( LIA); Delay(300); Nosound; end;
if crm=б then
begin Sound(S); Delay( 300); Nosound; end;
end;
{эта процедура анализирует код нажатой клавиши и, в зависимости от
него, передает управление процедурам Up и Down}
Procedure DoCommand(Key:char;functionkey:boolean);
begin
{если была нажата клавиша "Enter" }
      if(not functionkey) AND ( upcase(key)=#l3) then
       Do_Select(crm);
if functionkey then { если клавиша специальная }
case upcase (key) of
          #72: Up; {была нажата клавиша "Стрелка вверх" }
          #80: Down; {была нажата клавиша "Стрелка вниз" }
           end;
end;
begin
Init;
Draw_Menu;
repeat
ch:=readkey;
if ch=#0 then
begin
functionkey:=True;
   ch:=readkey;
end
else functionkey:=False;
DoCommand( ch,funcionkey);
until(upcase(ch)=#27);
end.
```

ГЛАВА 4. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

В главе 3 было показано, как писать стандартные программы на Паскале. Но как быть в случае нестандартного программирования - частного случая программирования на персональной компьютере IBM PC, с экранным управлением, с обращениями к операционной системе OS и графикой. Для того, чтобы самостоятельно писать такие программы, необходимо иметь представление о модулях или об аппаратнык средствах персонального компьютера. В настоящей главе разъясняется, что такое модуль, как он используется, какие встроенные модули доступны пользователю, как писать собственные программные модули и как компилировать их.

Borland Pascal обеспечивает вам доступ к большому числу встроенных констант, типов данных, переменных, процедур и функций, некоторые из них специфичны для Borland Pascal ; другие специфичны для персонального компьютера IBM PC (и совместимых с ним компьютеров) или для операционной системы OS. Их количество велико, однако, в своей программе вы редко используете их все сразу. Поэтому они разделены на связанные группы, называемые модулями. В этом случае можно использовать только те модули, которые необходимы в программе.

Программный модуль представляет собой набор констант, типов данных, переменных, процедур и функций. Каждый модуль аналогичен отдельной программе на Паскале; он может иметь основное тело, которое вызывается перед запуском вашей программы и осуществляет необходимую инициализацию. Короче говоря, модуль представляет собой библиотеку описаний, которую можно вставить в свою программу и которая позволит разбить.

Все описания внутри модуля связаны друг с другом. Например, модуль Crt содержит все описания, необходимые для подпрограмм работы с экраном на вашей персональном компьютере.

Borland Pascal предоставляет пользователю семь стандартных модулей. Пять из них - System, Graph, DOS, Crt и Printer обеспечивают поддержку обычных программ на Borland Pascal . Два других - Turbo3 и Graph3 - предназначены для обеспечения совместимости с программами и файлами данных, созданными под версией 3.0 Турбо-Паскаля. Все семь модулей хранятся в файле BP.TPL.

4.1. Структура модуля

Модуль обеспечивает набор средств благодаря процедурам и функциям при поддержке констант, типов данных и переменных, однако действительная реализация этих средств скрыта в силу того, что модуль разделен на две секции: интерфейса и реализации. Если программа использует модуль, то все описания модуля становятся доступными этой программе, как если бы они были определены в ней самой.

Структура модуля аналогична структуре программы, однако есть несколько существенных различий. Например, рассмотрим модуль:

```
unit <идентификатор>;
interface                    { Необязательный }
uses <список модулей>;       {открытые описания}
implementation       { Закрытые описания }
{ процедуры и функции}-
begin { код инициализации }
```

Заголовок модуля начинается зарезервированным словом unit, за которым следует имя модуля (идентификатор) точно так же, как и в случае имени программы. Следующим "элементом в модуле является ключевое слово interface. Оно обозначает начало интерфейсной секции модуля - секции, видимой всем другим модулям или программам, в которым: он используется.

Программный модуль может использовать другие модули, для этого они определяются в предложении uses. Предложение uses, если имеет место, то следует сразу после ключевого слова interface. Отметим, что здесь выполняется общее правило использования предложения uses; если модуль, имя которого указано в предложении uses, использует другие модули, то имена этих модулей также должны быть указаны в предложении uses, причем до того, как они будут использованы.

4.1.1. Интерфейсная секция

Интерфейсный раздел - "общедоступная" часть в модуле начинается зарезервированным словом interface, следует сразу после заголовка модуля и заканчивается перед зарезервированным словом implementation. Интерфейс определяет, что является "видимым" для любой программы (или модуля), используюмой данный модуль.

Любая программа, использующая "этот модуль, имеет доступ к'этим "видимым" элементам.

В интерфейсе модуля можно определять константы, типы данные, переменные, процедуры и функции. Как и в программе, они могут быть расположены в любом порядке, и секции могут встречаться повторно (например, *type* ... var ... < процедуры > ... const ... type

Процедуры и функции, видимые для любой программы, использующей данный модуль, описываются в секции интерфейса, однако их действительные тела - реализации - наводятся в секции реализации. Если процедура (или функция) является внешней, то в интерфейсе должно быть указано ключевое слово external, а в секции, реализации не нужно повторно указывать описание процедуры. Если процедура (или функция) является машинный код (список целочисленных констант), а в секции реализации не нужно указывать никакого другого описания процедуры. Описания forward (предописания) не являются необходимыми и не разрешаются,. Тела всех обычных процедур и функций наводятся в секции реализации, если заголовки этих процедур и функций перечислены в интерфейсной секции.

4.1.2. Секция реализации

Секция реализации - "приватная" часть - начинается зарезервированным словом implementation. Все, что описано в секции интерфейса, является видимым в секции реализации: константы, типы, переменные, процедуры и функции. Кроме того, в секции реализации могут быть свои дополнительные описания, которые не являются видимыми для программ, использующих этот модуль. Программа не знает о их существовании и не может ссылаться на них, или обращаться к ним. Однако, эти спрятанные элементы могут использоваться (и, как правило, используются) "видимыми" процедурами и функциями, то есть теми подпрограммами, чьи заголовки указаны в секции интерфейса.

Если какие-то процедуры были описаны как внешние, то в исходном файле должны быть указаны одна или несколько директив {$L имя_файла}. Если в исходном файле отсутствует секция инициализации, то директива {$L имя_файла} может быть указана в любом месте до завершающего end модуля. Директива $L позволяет

компоновать объектные модули языка ассемблера, разрешающие внешние процедуры.

Обычные процедуры и функции, описанные в интерфейсной секции (то есть те из них, которые не имеют тип inline), должны быть повторно указаны в секции реализации. Заголовок procedure/function должен быть или идентичным тому, который указан в секции интерфейса, или иметь более краткую форму. В случае краткой формы напечатайте ключевое слово (procedure или function), а за ним укажите имя подпрограммы (идентификатор). Затем подпрограмма должна содержать все свои локальные описания (метки, константы, типы, переменные и вложенные тело самой подпрограммы. Пусть в интерфейсной секции указаны следующие описания:

```
procedure ISwap(var V1.,V2 : integer);
function Imax(Vl,V2 : integer) ; integer;
```

Секция реализации тогда будет иметь вид;

```
procedure ISwap; var
    Temp : inteqer; begin
    Temp :=VI; VI ;= V2; V2 = Temp end; { конец процедуры Swap }
     function IMax(Vl,V2 ; integer) ; integer; begin
    if VI > V2
    then Imax:= Vl
    else 1Max:=V2 end; { конец функции Max }
```

Подпрограммы, локальные для секции реализации (то есть не описанные в секции реализации), должны иметь полный, несокращенный заголовок procedure/ function.

4.1.3. Секция инициализации

Обычно вся секция реализации модуля "заключена между зарезервированными словами implementation и end. Однако, если перед end поместить зарезервированное слово begin, а между ними - операторы, то получившийся составной оператор, очень похожий на основное тело программы, становится серией инициализации модуля.

Секция инициализации представляет собой место, где инициализируются структуры данных (переменных), которые использует программный модуль или которые он делает доступными программе, использующей данный модуль. Вы можете использовать эту секцию для открытия файлов, которые программа использует позже. Например, стандартный модуль Printer использует секцию инициализации для выполнения запросов на открытие (для вывода)

текстового файла Lst, который затем можно использовать в операторах Write и Writein в вашей программе.

При выполнении программы, использующей некоторый допуском основного тела программы. Если программа использует более одного модуля, то секции инициализации всех модулей вызываются в порядке, указанном (в операторе uses в программе перед тем, как выполнить основное тело программы.

4.2. Использование модулей

Модули, которые использует ваша программа, уже оттранслированы и хранятся, как машинный код, а не как исходный код на Паскале, поскольку они не являются включаемыми файлами (файлами типа Include). Даже интерфейсная секция хранится в специальном двоичном формате символьной таблицы, используемом в Borland Pascal . Более того, определенные стандартные модули хранятся в специальном файле (BP.TPL_) и автоматически загружаются в память вместе с Borland Pascal .

В результате использование одного или нескольких модулей очень незначительно увеличивает время компиляции вашей программы (обычно менее, чем на секунду). Если программные модули загружаются из отдельного файла на диске, то может потребоваться несколько дополнительные секунд для чтения с диска.

Как указывалось ранее, для использования специального модуля или набора модулей необходимо в начале программы пометить предложение uses, после которого указать список имен тех модулей, которые будут использоваться; имена должны быть разделены запятыми:

```
program  MyProg;
uses thisUnit, thatUnit, theOtherUnit;
```

Когда компилятор встречает это предложение uses, он прибавляет информацию из секции интерфейса каждого модуля к таблице символов и присоединяет машинный код, представленный в секции реализации, к самой программе.

Модули присоединяются к таблице символов в указанном порядке. Этот порядок может быть существенным, если один модуль использует другой. Например, если thisUnit использует

```
uses  thatUnit, thisUnit, the0therUnit;
```

или

```
uses  thatUnit, the0therUnit; thisUnit,
```

Короче говоря, в списке модуль должен быть указан после всех тех модулей, которые он использует.

Если в программе не указано предложение uses, Borland Pascal в любом случае присоединит стандартный модуль System. Этот модуль обеспечит выполнение некоторых стандартных паскалевских подпрограмм, а также нескольких подпрограммах, специфических для Borland Pascal.

4.3. Ссылки на описания модуля

Как только вы включили модуль в свою программу, все константы, типы данных, переменные, процедуры и функции, описанные в секции интерфейса этого модуля, становятся доступными для вашей программы. Например, допустим, что имеется следующий модуль:

```
unit My Stuff;
interface
const
My Value = 915;
type
My Stars   =   (Deneb,  Antares, Betelgeuse);
var
MyWord : strinq[20];
prcicedure SetMyWord(Star ; My Stars);
function TheAnswer : integer;
```

Та часть модуля, которая находится в интерфейсной секции, является видимой для вашей программы (и может быть ею использована):

```
proqram  TestStuff;
uses  MyStuff;
var
I     : integer;
AStar ; MyStars;
begin
Writeln(my Value);
AStar := Deneb;
SetMyWord(AStar);
Writeln(MyWord);
I : = TheAnswer;
Writeln (I)
end.
```

После того, как включения в программу предложения uses вы можете ссылаться на все идентификаторы, описанные в интерфейсной секции модуля MyStuff (MyWord, MyValue и так далее). Однако, рассмотрим следующую ситуацию;

```
program    TestStuff;
uses  MyStuff;
const
MyValue = 22;
var
   I    : integer;
   AStar : My Stars;
function TheAnswer : integer;
begin
TheAnswer := 1
end;
begin
Writeln (myVa1ue);
AStar := Deneb;
writeln(MyWord);
I := TheAnswer;
   Writeln(I)
    end.
```

В этой программе переопределяются некоторые из идентификаторов, описанных в MyStuff. Будучи оттранслированной и выполненной, эта программе будет использовать собственные определения для MyValue и TheAnswer, поскольку они были описаны позже, чем определения в MyStuff

Вероятно, вам интересно знать, каким образом в такой ситуации можно ссылаться на идентификаторы в MyStuff. Для этого необходимо перед каждым идентификатором помещать имя MyStuff с точкой (.). Например, рассмотрим еще одну версию этой программы:

```
program TestStuff;
uses MyStuff;
const
MyValue = 22;
var
   I    :  integers
   AStar : MyStars;
function TheAnswer ; integer;
begin
```

```
TheAnswer := 1
end;
begin
Write1n (MyStuff. MyVa1ue);
AStar :Deneb;
SetMyWord(AStar);
Writeln(MyWord);
I := MyStuff.TheAnswer
Writeln(I)
end.
```

Эта программа даст такие же ответы, что и первая, даже в том случае, если вы переопределите MyValue и TheAnswer. В действительности вы имели полное право (хотя и довольно сомнительное) написать первую программу следующим образом:

```
prograrn TestStuff;
uses My Stuff;
var
   I     : integer;
   AStar : MyStuff.MyStars;
begin
   Writeln(MyStuff .MyValue);
   AStar := My.Stuff.Deneb;
   MyStuff.SetMyWord(AStar);
   Writeln(My .Stuff .My Word);
   I := My Stuff .TheAnswer;
   Writeln(I)
   end.
```

Отметим, что имя модуля может предшествовать любому идентификатору: константе, типу данных, переменной или подпрограмме.

4.4. Предложение uses секции реализации

Borland Pascal позволяет вам размещать в секции реализации предложение использования (uses). В случае его присутствия предложение uses должно следовать непосредственно за ключевым словом implementation (аналогично тому, как в интерфейсной секции

предложение clause должно следовать непосредственно за ключевым словом interface).

Размещение секции реализации предложения uses позволяет "скрыть" внутренние детали модуля, поскольку используемые в секции реализации модули оказываются "невидимыми" для того, кто этот модуль использует. Более важным, однако, является то, что это позволяет вам строить взаимозависимые модули, иметь строго иерархическую структуру, то допускается использовать циклические ссылки на модули. В следующем разделе показан пример, демонстрирующий полезное использование циклических ссылок.

4.5. Циклические ссылки на модули

В следующей программе показаны два модуля, которые "используют" друг друга. Основная программа Circular использует модуль с именем Display. Модуль Display содержит в своей интерфейсной секции одно программу WriteXY, которая имеет три параметра: пару координат (x,y) и сообщение для вывода на экран. WriteXY перемещает курсор в точку (x,y) и выводит там сообщение. В противной случае она вызывает простую программу обработки ошибки.

Пока мы не видим здесь ничего интересного: процедура WriteXY просто используется вместо процедуры Write. Однако далее, когда программа обработки ошибки будет выводить сообщение на экран, начинаются циклические ссылки (ведь при этом она снова использует WriteXY). Таким образом, мы имеем процедуру WriteXY, выбывающую процедуру обработки ошибки SwapError, которая в свою очередь вызывает WriteXY для вывода сообщения на экран. Если у вас уже от всего этого закружилась голова, не беда. Давайте рассмотрим исходный код в примере и увидим, что все это не столь уж запутано.

Основная программа Circular очищает экран и выполняет три обращения к процедуре WriteXY:

```
program Circular;
{выводит текст, используя MriteXY }
uses
Crt, Display;
begin
CIrScr;
WriteXY(l, 1, 'Левый верхний угол экрана');
```

```
WriteXY(81 – Lenght(Cnова в 'экран..'), 15, 'Снова в 'экран..');
end.
```

Взгляните на координты (x,y) при втором обращении к процедуре WriteXY. В точке с координатами (100,100) на 80 x25 - символьном экране вывести текст невозможно. Давайте теперь посмотрим, как работает процедура MriteXY. Далее приведен текст исходного кода модуля Display, в котором содержится процедура WriteXY. Если координаты (x,y) являются допустимыми, она выводит на экран сообщение. В противном случае она выводит сообщение об ошибке.

```
unit Display;
{ содержит простую программу вывода информации на экран }

interface
procedure WriteXY(X,Y : integer, Message ; string);
inp1ementation
uses
Crt, Error;
procedure WriteXYCX.Y : integer, Message : string);
beqin
if (X in [I. .80] and Y in [1..25] then
begin
Goto (X,Y);
Write(Message);
end;
else
ShowErrог(‘Неверные координаты в процедуре WriteXY');
end;
end,.
```

Процедура ShowError, вызываемая в процедуре WriteXY, всегда выводит сообщение об ошибке на 25-й строке экрана.

```
unit Error;
{ содержит простую программу сообщения об ошибке }
interface
procedure ShowError(ErrMsg ; string);
inp1ementation
uses
Display;
procedure ShowError (ErrMsg ; string);
begin
```

```
WriteXY(l,25, 'Ошибка: '+ ErrMsg);
end;
end.
```

Обратите внимание, что предложения uses в секции реализации обоих модулей (Display и Error) ссылаются друг на друга. Эти два модуля могут ссылаться друг на друга в секции реализации благодаря тому, что Borland Pascal может для обоих модулей выполнять полную компиляцию интерфейсных секций. Другими словами, компилятор Borland Pascal воспринимает ссылку на частично скомпилированный модуль A в секции реализации модуля B, если интерфейсные секции модуля A и модуля B не зависят друг от друга (и, следовательно, строго соблюдаются правила Borland Pascal , касающиеся порядка описания).

4.6. Совместное использование описаний

Можно модифицировать процедуру WriteXY таким образом, чтобы она воспринимала дополнительный параметр, задающий прямоугольное окно на экране:

```
procedure WriteXY(SomeWindow : WindRec;
          X, Y : integer;
          Message : string);
procedure ShowError(Somewindow : WindRec;
ErrMsg : string);
```

Нужно учитывать, что две процедуры находятся в разных модулях. Даже если вы описываете WindData в интерфейсной секции одного модуля, то нет такого допустимого способа, с помощью которого это описание могло бы быть доступно в другом модуле. Решение состоит в том, чтобы описать третий модуль, в которой содержится только определение записи Mind Rec;

```
unit   Wind Data;
integface
type
Wind Rec= record
XI, Yl, X2, Y2 : integer;
ForeColor,
BackColor     : byte;
Active          : boolean;
end;
inp1ementation
```

end.

В добавление к тому, что модификация кода процедур WriteXY и ShowError позволяет использовать новый параметр, в интерфейсной секции модулей Display и Error теперь может использоваться WindData. Это допустимо, така как модуль WindData не зависит от своего предложения uses, а модули Display и Error ссылаются друг на друга только в соответствующих секциях реализации.

4.7. Компиляция модуля

Модуль компилируется точно так же, как компилируется программа: он создается с помощью редактора, а затем вызывается ко м а н да Compile/Compile (Компилировать/Компилировать) (или нажимаются клавиши Alt-C). Однако, вместо файла с расширением .EXE Borland Pascal создает файл с расширением .TPU (**Borland** Pascal Unit - модуль Borland Pascal). После этого Вы можете оставить этот файл как есть или же вставить его в BP.TPL с помощью TPUMOVER.ESE .

В любом случае имеет смысл переслать файлы с расширением .TPU (вместе с исходными файлами) в каталог модулей, который определен с помощью команды 0/D/Unit directories (Каталоги модулей). Таким образом, вы можете ссылаться на эта файлы, не указывая директивы {$U}- (Команда Unit directories (Каталоги модулей) позволяет задавать компилятору несколько каталогов и для поиска файлов модулей).

В одном исходном файле может находиться только модуль, поскольку компиляция прекращается, как только обнаружен завершаающий оператор end.

Пример.

Теперь напишем небольшой модуль. Назовем его IntLib и вставим в него две простые подпрограммы для целых чисел - процедуру и функцию:

```
unit IntLib;
interface
procedure Iswap(var I,J ; integer);
.implementation
procedure  ISwap;
var
Temp : integer;
begin
```

```
Temp := I; I := J; J ;= Temp
end; { конец процедуры ISwap }
function IMax;
begin
if I > J
      then Imax := I
      else IMax := J
      end; { конец функции Imax}
end. {конец модуля IntLib }
```

Введем эту подпрограмму, запишем ее в файл INTLIB.PAS, а затем оттранслируем на диск. В результате получим код модуля в файле INTLIB.TPU. Перешлем его в каталог модулей (если таковой имеется).

Следующая программа использует модуль IntLib:

```
program IntTest;
uses IntLib;
var
A, B : integer;
begin
  Write (‘Введите два целочисленных значения:    ');
  Readln(A, B);
  ISwap(A,B);
  Writeln (‘A = ',A,' B = ',B);
Write1n ('Максимальное значение равно ',IMax(A,B));
end. {конец программы IntTest}
```

4.8. Модули и большие программы

Вероятно, до сих пор вы представляли программные модули только как библиотеки - наборы полезных подпрограмм, которые могут использоваться несколькими программами. Однако, у модуля есть еще одна функция - разбивать большую программу на составные части.

Два аспекта Borland Pascal способствуют использованию модулей в такой функции; (1) высокая скорость компиляции и компоновки и (2) способность работать с несколькими файлами одновременно, например, с программой и несколькими модулями.

В обычном случае большая программа разбивается на модули, которые группируют процедуры по их функциям. Например, программа редактора может быть разделена на части, выполняющие

инициализацию, распечатку, чтение и запись файлов, форматирование и так далее. Кроме того, как имеет место основная программа, определяющая глобальные константы, типы данных, переменные, процедуры и функции, так же может иметь место и "общий" модуль, который используется всеми другими модулями.

Структура большой программы может выглядеть следующим образом;

```
program Editor;
uses
  DOS,Crt,Printer {Стандартные модули из BP.TP'L }
  EditGlobals,   { Модули, написанные пользователем }
  EdiFtInit,
  EditPrint,
  EditRead, Edit Write,
EditFormat; { описания программы, процедуры и функции }
begin
{основная программа}
end. {конец программы Editor}
```

Отметим, что модули, указанные в этой программе, с должным расширением означает, что при повторном компилировании программы Editor Borland Pascal проверит последние обновления для каждого файла с расширением .TP'L и заново оттранслирует их в случае необходимости.

Другая причина использования модулей в больших программах определяется ограничениями на размер сегмента. Процессоры 8086 (и связанные с ними) ограничивают размер куска программы или сегмента до 64К. Это означает, что основная программа и любой данный сегмент не должны превышать по размеру 64К**.** Borland Pascal разрешает "эту ситуацию, превращая каждый модуль в отдельный сегмент. Верхним пределом является количество памяти, поддерживаемое аппаратными средствами и операционной системой, то есть 640К на большинстве персональных компьютерах PC. Без использования модулей пределом для вашей программы является 64К.

4.9. Использование модулей в качестве оверлеев

Иногда даже возможность использования нескольких модулей не помогает решить проблему нехватки памяти - ведь у вас может оказаться меньше 640К памяти, или одновременно используете в

памяти большое количество данных. Другими словами, может оказаться, что ваша программа не может целиком поместиться в память. Borland Pascal предлагает решение такой проблемы — оверлеи. Оверлей представляет собой часть программы, которая эагружается в память при необходимости ее использовать и выгружается, когда она не нужна. Это позволяет вам помещать в память только те секции программы, которые в данный момент необходимы.

Оверлеи в Borland Pascal основываются на модулях; наименьший участок кода может загружаться и выгружаться, как целый модуль. Вы можете определить сложный набор оверлеев, определяя, какой модуль должен или не должен находится в памяти в данный момент. Кроме того в Borland Pascal имеется превосходная развитая подсистема управления оверлеями, поэтому вам не нужно беспокоиться о загрузке и разгрузке.

4.10. Создание оверлейных файлов

Оверлеи - это части программы, которые разделяют общую область памяти. Только те части программы, которые требуются для выполнения данной функции, размещаются в памяти в это время; затем они могут быть перекрыты другими программами.

Оверлеи могут значительно сократить количество памяти, требуемое при выполнении программы. В действительности, Вы можете выполнять программы, которые намного больше, чем доступная в системе память, поскольку в каждый момент в памяти размещается только часть программы.

Borland Pascal управляет оверлеями на уровне модулей; это наименьшая часть программы, которую можно сделать оверлейной. Когда оверлейная программа компилируется, Borland Pascal генерирует оверлейный файл (с расширением .OVR) в дополнение к выполнимому файлу (с расширением .EXE). .EXE файл содержит статическую (неперекрываемую) часть программы и .OVR файл содержит все модули, которые будут перекачиваться в/из памяти во время выполнения программы.

За исключением нескольких правил программирования, оверлейные модули идентичны с неоверлейными модулями. В действительности, если Вы соблюдаете эти правила, Вам даже не нужно перекомпилировать эти модули, чтобы сделать их

оверлейными. Решение, будет модуль оверлейным или нет, принимает главная программа.

Когда оверлей загружается в память, он помещается в оверлейный буфер, который размещается в памяти между стеком и кучей. По умолчанию, размер оверлейного буфера устанавливается как можно меньше, но он может быть легко увеличен во время выполнения, выделением дополнительной памяти из кучи. Как сегмент данных и минимальный размер кучи, размер буфера оверлеев (по умолчанию) распределяется при загрузке .EXE файла. Если памяти недостаточно, DOS выдает сообщение об ошибке (программа слишком велика для загрузки в память) или интегрированной средой Borland Pascal (недостаточно памяти для выполнения программы).

Есть очень важная возможность монитора оверлеев при загрузке оверлейного файла в случае, когда в системе достаточный объем расширенной памяти. Для этих целей Borland Pascal поддерживает Спецификацию Расширенной Памяти фирм Lotus/Intel/Microsoft (Expanded Memory Specification - EMS) версии 3.2 и выше. После загрузки в EMS, оверлейный файл закрывается и загрузка оверлеев выполняется быстрым копированием в памяти.

ГЛАВА 5. ТЕХНИЧЕСКАЯ ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПЕРЕВОЗОЧНЫМ ПРОЦЕССОМ НА Ж.Д. ТРАНСПОРТЕ

Для перевозки грузов между городами, населенными пунктами и государствами широкое распространение получил железнодорожный транспорт, при котором связь между городами осуществляется по железнодорожным путям.

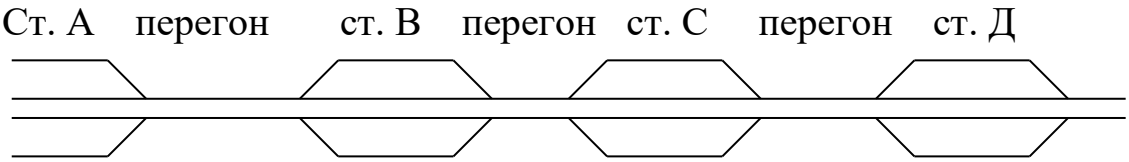


Рис. 3

На данном рисунке показан железнодорожный участок, который включает в себя 4 станции и три перегона между станциями А-В; В-С и С-Д.

Для организации движения и обеспечения безопасности движения поездов используются технические средства, которые подразделяются на перегонные системы и станционные системы.

В настоящее время к перегонным системам можно отнести:

автоблокировку;

автоматическую локомотивную сигнализацию;

автоматическую переездную сигнализацию;

диспетчерский контроль;

диспетчерскую централизацию.

К станционным системам относятся:

электрическая централизация стрелок и сигналов;

горочная электрическая централизация;

диспетчерская централизация.

5.1 Организация движения поездов на перегонах.

5.1.1. Автоматическая блокировка

Автоблокировка представляет собой систему регулирования движения поездов, при которой перегоны делятся на блок-участки, ограждаемые путевыми светофорами рис. 4

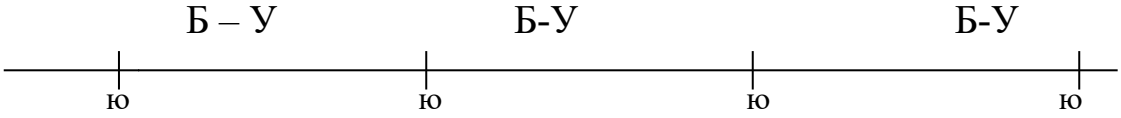


Рис. 4

Блок-участки оборудуются дискретными датчиками информации рельсовыми цепями рис. 5.

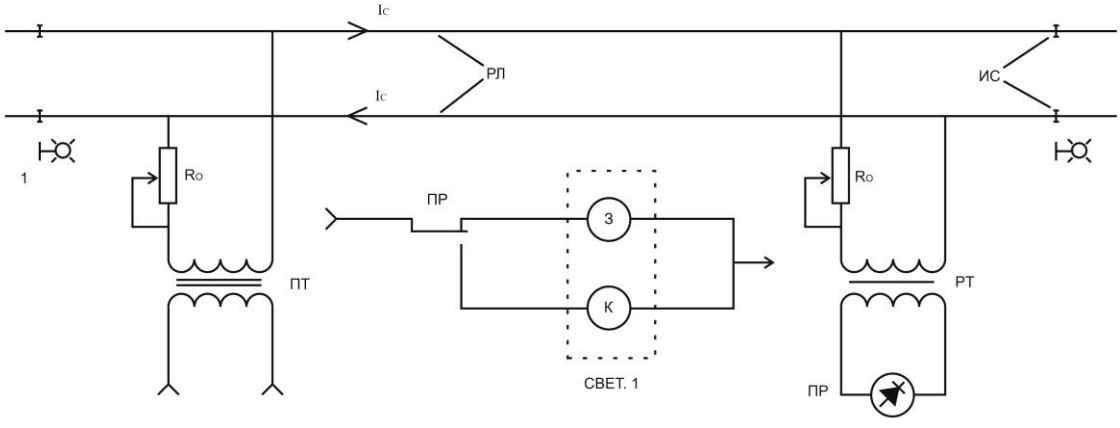


Рис. 5. Схема рельсовой цепи при отсутствии подвижной единицы

Элементарная рельсовая цепь состоит из следующих элементов:

РЛ – рельсовая линия – железнодорожные рельсы

ИС – изолирующие стыки, электрически отделяющие соседние

рельсовые цепи между собой;

ПТ – путевой трансформатор;

РТ – релейный трансформатор;

ПР – путевое реле;

R_0 – ограничивающий резистор.

При отсутствии подвижной единицы, как показано на рис. 5 через РТ протекает сигнальный ток Iс, который индуктирует во вторичную обмотку релейного трансформатора ЭДС и путевое реле находится под током, фиксируя свободность блок-участка и включает на светофоре разрешающий огонь.

.

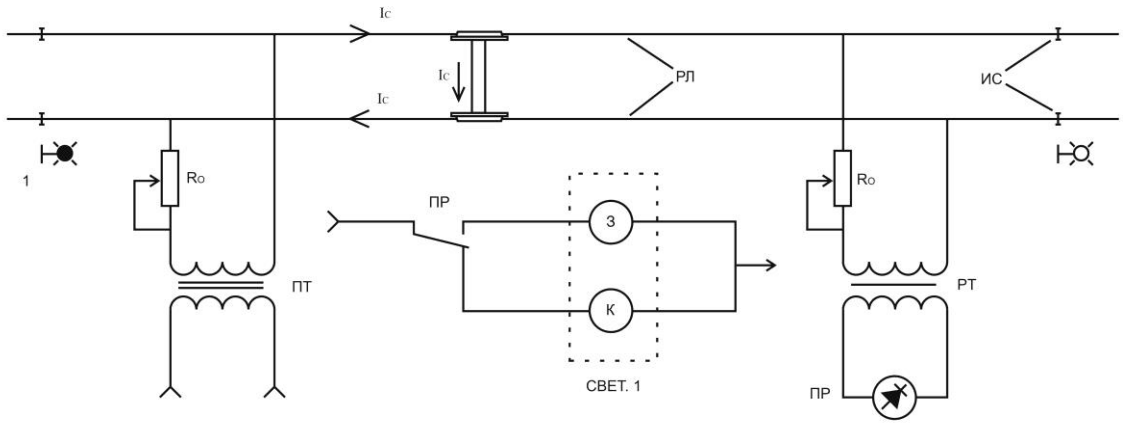


Рис. 6. Схема рельсовой цепи при наличии подвижной единицы.

При наличии подвижной единицы происходит шунтирование колесной парой рельсовой цепи и основная величина тока начинает протекать через колесную пару. Ток, протекающий через реле, резко уменьшается, реле обесточивается и включает на светофоре ограждаемого участка запрещающий огонь, рис 6.

После освобождения блок- участка через реле опять начинает протекать ток и реле становится под ток, включая своим контактом разрешающий огонь на светофоре рис. 5.

Как видно из рисунков сигнальные показания путевых светофоров находятся в зависимости от состояния (свободен или занят) блок-участков и изменяются автоматически в результате воздействия движущихся поездов на соответствующие устройства.

5.2. Организация движения поездов на станциях

Железнодорожные линии делятся раздельными пунктами на перегоны. К раздельным пунктам относятся станции, разъезды, обгонные пункты и путевые посты, а при автоблокировке, кроме того, проходные светофоры. Наличие раздельных пунктов позволяет выпускать на участок несколько поездов, разграничивая их расстоянием (перегонами или блок-участками), что обеспечивает безопасность движения и повышает пропускную способность.

Станциями называются раздельные пункты, имеющие путевое развитие и другие устройства рис. 7, которые позволяют, кроме приема, отправления, обгона и скрещения поездов, производить грузовые операции, обслуживание пассажиров, а также формирование и расформирование поездов, их технический осмотр, смену локомотивов, ремонт подвижного состава и т.д.

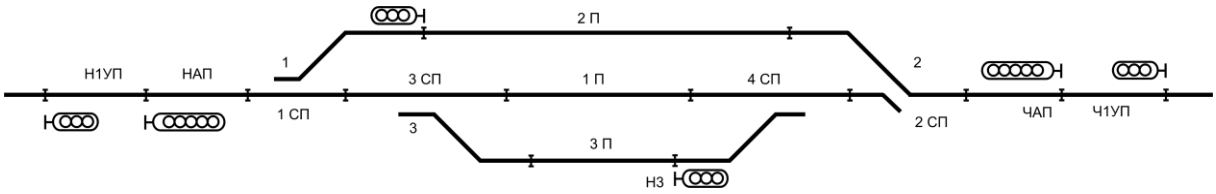


Рис. 7

Для эффективной организации движения поездов по станции и станционной работы, станция делится на участки – стрелочные (например 1СП, 3 СП, 2 СП, 4 СП), бесстрелочные (НАП, ЧАП), а также приемо-отправочные пути (1п, 2п, 3п). Все эти участки оборудуются рельсовыми цепями.

Основными элементами путевого развития всех станций являются:

- стрелочные переводы;
- входные светофоры;
- выходные светофоры;
- маневровые светофоры;
- рельсовые цепи.

Стрелочные переводы служат для задания того или иного пути поезду. Если поезду установлен по стрелке прямое направление, то стрелка считается находящейся в плюсовом положении (+), если на ответвлении в минусовом (-). На рис. 7 показано, что стрелки 1,3, 4 находятся в плюсовом положении, а стрелка 2 в минусовом положении.

Входные светофоры имеют пятизначное показание и предназначены для разрешения или запрещения въезда состава на станцию.

Выходные светофоры имеют трехзначное показание и предназначены для разрешения или запрещения выхода поезда со станции.

Маневровые светофоры имеют двузначную сигнализацию и предназначены для регулирования движения маневрового локомотива в пределах станции.

На железных дорогах нашей страны и стран СНГ принят порядок нумерации поездов, позволяющий определять направление их движения. Так, поезда, движущиеся с юга на север и с запада на восток, имеют четную последнюю цифру номера, а в противоположном направлении – нечетную.

Стрелки, расположенные со стороны прибытия четных поездов, получают порядковые четные номера, а со стороны прибытия нечетных – нечетные.

Границей, разделяющей одну нумерацию от другой является ось пассажирского здания.

Светофоры нумеруются по цифре пути и по направлению движения поездов, например, $Ч_2$, $Н_3$ и т.д.

В нормальном состоянии на станционных светофорах должны гореть лампы красного огня. В процессе приема-отправления поездов показания станционных светофоров находятся в зависимости от многих факторов, обеспечивающих безопасность движения поездов.

5.3. Разновидности систем автоматики и телемеханики на станциях

С целью повышения пропускной способности и повышения безопасности движения поездов промежуточные и участковые станции оборудуют устройствами электрической централизации ЭЦ. Основной элементной базой системы ЭЦ является релейная аппаратура, поэтому эта система управления получила название релейной централизации. Релейная централизация в соответствии с требованиями ПТЭ не допускает: открытие входного светофора при маршруте, установленном на занятый путь; перевод стрелок под составом; открытие сигналов, соответствующих данному маршруту, если стрелки не установлены в надлежащее положение и не заперты в этом положении, а сигналы враждебных маршрутов не закрыты; перевод входящей в маршрут стрелки или открытие сигнала

враждебного маршрута при открытом сигнале, ограждающем установленный маршрут.

В состав релейной централизации входят: аппарат управления; релейная аппаратура, обеспечивающая требования по безопасности движения поездов; источники питания; стрелочные электроприводы для централизованного управления и контроля положения стрелок; светофоры, электрические рельсовые цепи; кабельные сети.

По способу размещения аппаратуры управления и источников питания релейную централизацию строят с местными и центральными зависимостями и источниками питания. При местных зависимостях релейную аппаратуру размещают в релейных будках в горловинах станции; при центральных—в центре станции на посту ЭЦ или в станционном здании. Местные источники в виде аккумуляторных батарей устанавливают в батарейных шкафах у входных светофоров и в районе стрелочных горловин.

В устройствах релейной централизации применяют два способа управления—индивидуальный (раздельный) и маршрутный.

При индивидуальном управлении перевод стрелок, входящих в маршрут, и открытие светофоров осуществляют нажатием отдельных кнопок или переводом коммутаторов, расположенных на пульте дежурного; маршрутном — перевод стрелок и открытие светофора осуществляют последовательным нажатием двух кнопок—начала и конца маршрута.

Применяют несколько разновидностей систем релейной централизации.

Релейная централизация с местными зависимостями и местными источниками питания (РЦМ). Система РЦМ применялась на малых станциях (до 15 стрелок). Релейная аппаратура и источники питания размещались в релейных будках или шкафах в горловинах станции. Недостатком системы является рассредоточенность аппаратуры и источников питания, что усложняет обслуживание и удорожает строительство. Эту систему в новом строительстве не применяют.

Релейная централизация с центральными зависимостями и местными источниками (РЦЦМ). В системе РЦЦМ пост электрической централизации не строят, и релейную аппаратуру размещают в станционном здании, где находится дежурный по станции (ДСП), и частично в релейных шкафах, установленных у входных и выходных светофоров станции; источники питания в виде аккумуляторных батарей помещены в батарейных шкафах,

установленных у входных светофоров и в районе стрелочных горловин. В системе применен принцип раздельного управления, которое ведется с пульта управления. Недостатками системы являются: рассредоточенность аппаратуры, источников питания, применение низковольтных электроприводов, большого числа аккумуляторов, отсутствие маневровых маршрутов. Данную систему применяют ограниченно на промежуточных станциях малодеятельных участков.

Релейная централизация с центральными зависимостями и центральными источниками питания (РЦЦ). Релейную аппаратуру и источники питания размещают на посту электрической централизации, что улучшает условия обслуживания, позволяет применять более совершенные источники питания. Сначала данную систему применяли на участковых станциях, где управление ведется с пульт-табло, на котором размещены стрелочные и сигнальные кнопки.

Нажатием стрелочных кнопок производят раздельный перевод стрелок, сигнальных кнопок— открытие сигналов.

В данной системе электрические схемы строят по плану станции, что значительно упрощает схемы, сокращает расход релейной аппаратуры и позволяет, кроме поездных маршрутов, включать централизованные маневровые маршруты. С целью унификации полная схема для всех видов маршрутов разделена на типовые схемные узлы, из которых может быть построена полная схема централизации для станции с любым путевым развитием.

Начиная с 60-х годов систему РЦЦ применяют на промежуточных станциях.

Управление ведется с пульта блочного типа с желобковой сигнализацией, на котором у повторителей поездных и маневровых светофоров расположены маршрутные кнопки.

Последовательным нажатием кнопок начала и конца маршрута выполняют упрощенный маршрутный набор простых поездных и маневровых маршрутов.

Релейная централизация с центральными зависимостями, центральными источниками питания и маршрутным управлением. Релейная аппаратура и источники питания размещены на посту ЭЦ. где для управления имеется пульт-табло или пульт-манипулятор с маршрутными кнопками.

При установке маршрута последовательным нажатием кнопок начала и конца маршрута осуществляют набор задания поездных и

маневровых маршрутов. По окончании набора происходит одновременный перевод всех стрелок в маршруте и после их перевода — открытие сигнала. Маршрутное управление позволяет устанавливать самый сложный маршрут за 5—7 с вместо 30—40 с при раздельном управлении, что значительно повышает пропускную способность участковых станций.

Релейная аппаратура размещена в типовых блоках. Система в таком исполнении получила название блочной маршрутно-релейной централизации (БМРЦ). На заводе-изготовителе организовано массовое производство типовых блоков. Блочная структура упрощает проектирование, сокращает сроки строительства и улучшает условия эксплуатации. Преимущества блочной структуры позволяют применять ее и на промежуточных станциях в виде блочной электрической централизации с раздельным управлением (БРЦ).

5.4. Принципы работы станционных систем автоматики и телемеханики

5.4.1. Алгоритм установки и размыкания маршрутов для малых станций релейной централизации

Система релейной централизации малой станции выполняет следующие функции: установка маршрута переводом стрелок; замыкание маршрута для исключения возможности перевода стрелки в установленном маршруте до прохода по нему поезда или до отмены этого маршрута; открытие светофора, при этом система автоматически выбирает один или несколько разрешающих огней на светофоре в соответствии с характером установленного и замкнутого маршрута (прием, отправление, сквозной пропуск); автоматическое размыкание маршрута под действием проходящего поезда или размыкание при отмене маршрута, когда маршрут разделывается до вступления на него поезда, а также искусственное размыкание маршрута.

Принципиальные электрические схемы, реализующие данные функции, построены так, чтобы выполнять требования по обеспечению безопасности движения поездов по станции.

В функциональной схеме системы РЦЦМ (рис. 8) маршрут устанавливают раздельным способом, т. е. каждую стрелку, входящую в маршрут, переводят нажатием стрелочной кнопки МК или ПК. Окончательный перевод стрелок контролируется зажиганием ламп минусового и плюсового положения.

Затем в системе автоматически контролируются все условия правильности установки маршрута: 1) правильное положение ходовых и охранных стрелок; 2) свободность путевых и стрелочных секций, участвующих в маршруте, и негабаритных; 3) свободность приемного пути; 4) отсутствие враждебных маршрутов в данной горловине станции; 5) отсутствие встречных (лобовых) маршрутов на один и тот же приемный путь с разных концов станции; 6) отсутствие горения на входном светофоре лунно-белого пригласительного огня; 7) целость нити ламп разрешающих огней входного светофора: 8) отсутствие искусственной разделки маршрута; 9) и 10) отсутствие враждебных передач стрелок на местное управление в данной и противоположной горловинах станции соответственно.

Для маршрутов отправления необходимо контролировать условия 1), 2), 4), 9), 10), как и для маршрута приема, а также: 11) отсутствие горения мигающего лунно-белого огня на выходном светофоре на станциях двухпутных участков; 12) свободность не менее одного блок-участка удаления на перегоне при автоматической блокировке или всего перегона при полуавтоматической блокировке; 13) соответствие установленного направления движения отправляющемуся поезду на станциях однопутных участков; 14) отсутствие ранее отправленного на перегон хозяйственного по езда с ключом-жезлом; 15) отсутствие искусственной разделки маршрута.

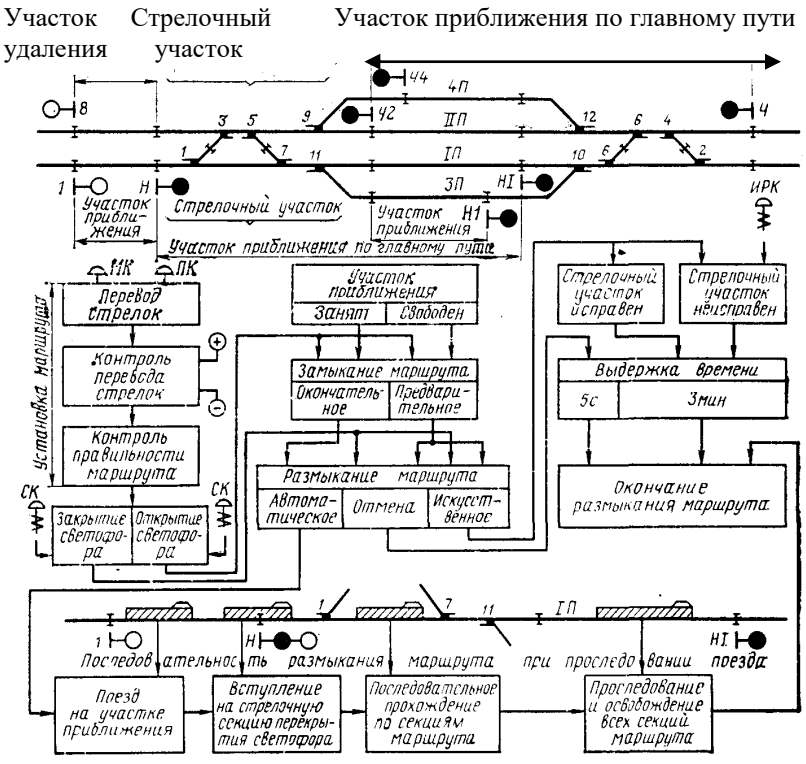


Рис. 8. Функциональная схема системы РЦЦМ

При установке маневровых маршрутов контролируют условия 1), 2), 4), 8), 9).

Некоторые из условий в зависимости от путевого развития и характера станционной работы могут не выполняться: условие 3) не контролируется, так как маневровые передвижения могут производиться на занятые вагонами приемо-отправочные пути; условие 5) не контролируется, так как встречные маневровые маршруты на один и тот же путь с разных концов станции не считаются враждебными.

После окончания установки маршрута, получив контроль правильной установки, дежурный по станции нажимает сигнальную кнопку *СК* для открытия светофора. После нажатия кнопки *СК* маршрут автоматически замыкается, чем исключается перевод любой стрелки маршрута в противоположное положение. С проверкой замыкания маршрута образуется цепь открытия светофора, дежурный по станции вытягивает кнопку *СК,* отчего на светофоре зажигается красный огонь.

Замыкание маршрута является наиболее ответственной функцией, осуществляемой релейной централизацией. Кроме электрического запирания всех стрелок маршрута, при замыкании установленного маршрута исключаются все ему враждебные. Стрелки замкнутого маршрута заперты до момента освобождения маршрута поездом или до принудительной разделки маршрута дежурным по станции. Чтобы произвести замыкание маршрута, необходимо учесть: имеется ли непосредственная опасность, несвоевременного выполнения маршрутных замыканий и возможность вступления поезда в пределах маршрута раньше, чем эти замыкания наступят, а также есть ли возможность отменить установленный маршрут при возникновении препятствий по его реализации. Исходя из этого выполнение маршрутных замыканий ставят в зависимость не только от момента воздействия на открытие сигнала, но и от приближения к нему поезда.

При необходимости маршрутные замыкания можно отменить, если перед открытым светофором отсутствует приближающийся поезд и не существует опасности несвоевременного выполнения этих замыканий. Дежурный по станции, закрыв светофор, отменяет замкнутый маршрут. Через несколько секунд маршрут размыкается, снимается запирание со стрелок, ДСП может установить новый маршрут. При занятом участке перед открытым светофором возникает опасность несвоевременного размыкания маршрута, и

поэтому для разделки маршрута ДСП использует искусственное размыкание маршрута с выдержкой времени.

При наличии двух указанных условий применяют два вида замыкания маршрутов—предварительное и окончательное.

Предварительное замыкание маршрута наступает с момента нажатия сигнальной кнопки *СК* и открытия светофора при условии, что участок приближения к светофору свободен (см. рис.). Участком приближения для маршрутов приема является блок-участок, расположенный между входным и предупредительным светофорами (первый участок приближения) длиной не менее 1000 м; для маневровых—одна изолированная стрелочная или путевая секция перед маневровым светофором.

Окончательное замыкание маршрута наступает с момента нажатия кнопки *СК,* открытия светофора и нахождения поезда на участке приближения перед ним. Такой маршрут может разомкнуться автоматически под действием проходящего поезда или в режиме искусственного размыкания. После установки и замыкания маршрута открывается светофор, после чего релейная централизация выполняет функции размыкания маршрута.

Автоматическое размыкание маршрута происходит только под действием проходящего по нему поезда. Релейная централизация автоматически контролирует местоположение поезда и размыкает маршрут в такой последовательности. При вступлении поезда на участок приближения перед входным светофором *Н* наступает окончательное замыкание маршрута. Вступление первых скатов поезда за входной светофор приводит к автоматическому закрытию светофора. Прохождение поезда по маршруту контролируется с помощью рельсовых цепей стрелочных и путевых секции. После проследования всех стрелок горловины и занятия приемо-отправочного пути 1П станции появляется контроль полного проследования поезда по маршруту и наступает размыкание маршрута.

Отмену маршрута производит ДСП при свободном участке приближения вытягиванием сигнальной кнопки *СК.,* после чего закрывается светофор, и через 5 с маршрут размыкается.

Искусственное размыкание производит ДСП при невозможности использования окончательно замкнутого маршрута или в случаях неполного размыкания при неисправностях рельсовых цепей стрелочных участков, входящих в маршрут.

Используют два режима искусственного размыкания окончательно замкнутого маршрута—при исправных и свободных стре-

лочных участках, при неисправных рельсовых цепях стрелочных участков маршрута.

В первом случае ДСП вытягиванием сигнальной кнопки перекрывает светофор, после чего с выдержкой времени 3 мин маршрут размыкается.

Во втором случае ДСП, вытягивая сигнальную кнопку, перекрывает светофор и, кроме того, нажимает кнопку искусственного размыкания *ИРК,* после чего с выдержкой 3 мин маршрут размыкается. Введение кнопки *ИРК* необходимо для исключения возможности искусственного размыкания маршрута без участия ДСП в случае повреждения рельсовой цепи и автоматического перекрытия светофора.

Большая выдержка времени при искусственном размыкании маршрута введена из-за требований обеспечения безопасности движения. Без выдержки времени маршрут размыкается сразу после закрытия светофора, и ДСП может переводить стрелки для установки нового маршрута. Если поезд, приближающийся к светофору, не будет своевременно остановлен и проследует на стрелки, то возможна аварийная ситуация схода состава. Использование выдержки времени исключает такую ситуацию, так как в случае проследования поезда на стрелки они остаются замкнутыми в маршруте после закрытия светофора еще 3 мин.

Все описанные выше функции, выполняемые устройствами релейной централизации, реализуются схемами, в которых предусмотрена защита от последствий отказов, возникающих как в самих схемах, так и в напольных устройствах—светофорах, электроприводах, рельсовых цепях и т. д.

Отказы маршрутных схем делят на опасные и защитные. Опасные отказы могут привести к переводу стрелки под составом, приему поезда на занятый путь, получению ложного контроля положения стрелки. Для исключения опасных отказов предусматривают схемы контроля и замыкания, которые собирают с нормально возбужденными реле. В случае отказа (обрыв цепи, короткое замыкание) реле выключается, размыкая цепь, исключается перевод стрелок и открытие светофоров. Схемы исключения опасных отказов строят с применением в ответственных цепях реле I класса надежности (НШ, НМШ, КМШ), которые обеспечивают надежное отпускание якоря при выключении тока. Также предусматривают взаимосвязь между реле отдельных схем для контроля защитных отказов и исключения опасных. Характерным примером может служить взаимосвязь между

сигнальным С и замыкающим *З* реле. При обесточенной обмотке, но залипшем якоре реле *С* реле *З* выключено, и маршрут замкнут. При залипании якоря реле З не создается цепи включения реле С, и светофор не открывается. В случае обрыва цепи реле С светофор остается закрытым; при обрыве цепи реле *З* маршрут не размыкается, чем исключается перевод стрелок.

5.4.2. Алгоритм установки и размыкания маршрутов для участковых станций

Участковые станции имеют сложное путевое развитие. Для них характерна большая поездная и маневровая работа. Для ускорения поездной и маневровой работы, а также обеспечения безопасности движения поездов на этих станциях все поездные и маневровые передвижения маршрутизированы. По условию выполнения маневровой работы только в отдельных случаях предусматривают немаршрутизированные передвижения с передачей стрелок на местное управление с маневровых колонок.

Для повышения эксплуатационных показателей на участковых станциях вместо системы релейной централизации с раздельным управлением, как на промежуточных станциях, разработана и широко внедряется маршрутно-релейная централизация (МРЦ). В этой системе для ускорения установки маршрутов стрелки в маршруте переводятся не раздельно последовательно, а одновременно (все стрелки, входящие в маршрут). Маршрутное управление осуществляют с помощью кнопок на пульте управления по границам поездных и маневровых маршрутов. Последовательным нажатием кнопок по границам маршрута, по принципу «откуда— куда», включают пусковые цепи для одновременного перевода стрелок, входящих в маршрут. При маршрутном управлении общее время на установку самого сложного маршрута складывается из времени нажатия кнопок, времени параллельного перевода одиночных стрелок, входящих в маршрут, и последовательного перевода спаренных стрелок, что составляет примерно 5—10 с. При последовательном переводе стрелок, входящих в маршрут (примерно 15 стрелок), время на установку маршрута составит 4,5Х15=67,5 с. За счет сокращения времени на установку маршрутов при маршрутном управлении пропускная способность горловины станции повышается на 15—20%. Система МРЦ также повышает производительность и культуру труда эксплуатационных работников станции.

Релейная аппаратура маршрутно-релейной централизации разделяется на наборную и исполнительную группы. Наборную группу называют маршрутным набором и используют для формирования пусковых цепей управления стрелками. Исполнительная группа осуществляет установку и замыкание маршрутов, управление светофорами поездных и маневровых маршрутов, а также размыкание маршрутов. Наборная группа не выполняет зависимостей по обеспечению безопасности движения поездов, поэтому реле маршрутного набора берут II класса надежности типа КДРШ. Исполнительная группа выполняет все требования по обеспечению безопасности движения поездов, поэтому в этой группе применяют реле I класса надежности НМШ и КМШ. Схемы исполнительной группы унифицированы, и их можно использовать при раздельном и маршрутном управлении.

В зависимости от конструктивной компоновки аппаратуры система МРЦ может быть неблочного и блочного типов (БМРЦ). Блоки представляют собой типовые изделия, изготавливаемые на заводе.

В начале внедрения маршрутно-релейная централизация проектировалась и строилась неблочного типа. Наборная группа выполнялась на реле КДРШ открытым монтажом, исполнительная группа — в виде типовых схемных узлов на реле НМШ также с открытым монтажом. Затем появилась блочная система, в которой применялись блоки закрытого типа только для исполнительной группы, наборная группа по-прежнему выполнялась на аппаратуре с открытым монтажом. Такую систему строили до 1966 г., после этого периода нашла применение блочная структура маршрутного набора, и одновременно с этим стали применять стрелочные пусковые блоки. Появилась полностью блочная маршрутно-релейная централизация БМРЦ. Только небольшая часть релейной аппаратуры не компонуется в блоки и монтируется открытым монтажом.

5.4.3. Принципы работы системы БМРЦ

Работа системы БМРЦ при установке, например, маршрута приема на путь/Я начинается с наборной группы (рис. 9). Нажатием кнопки *НН* начала маршрута определяется направление и категория маршрута, после чего исключается набор маршрута другого направления и категории. Для данного направления и категории маршрута при нажатой кнопке *НН* определяется начало маршрута от светофора *Н,* Нажатием кнопки конца маршрута *НК* определяется

конец маршрута. После этого в пределах установленных границ происходит маршрутный перевод всех стрелок, входящих в маршрут. По окончании перевода стрелок специальной схемой соответствия контролируется правильность набора и положения переведенных стрелок. При наличии соответствия включаются реле *Н* начала и *КМ* конца набранного поездного маршрута, происходит переход к исполнительной группе.

Работа исполнительной группы начинается с установки режима *Установка маршрута,* В зависимости от установленных границ набранного маршрута выбирают путевые и стрелочные секции, входящие в этот маршрут. После этого с помощью контрольно-секционных реле *КС* контролируются все условия правильности установленного маршрута. Если маршрут установлен правильно и возбудились реле КС всех секций маршрута, то выключаются замыкающие реле *З* этих секций, секции замыкаются. С контролем замыкания маршрута включается сигнальное реле *С,* открывающее светофор. При необходимости экстренного закрытия светофора нажимают кнопку начала маршрута НН данного светофора.

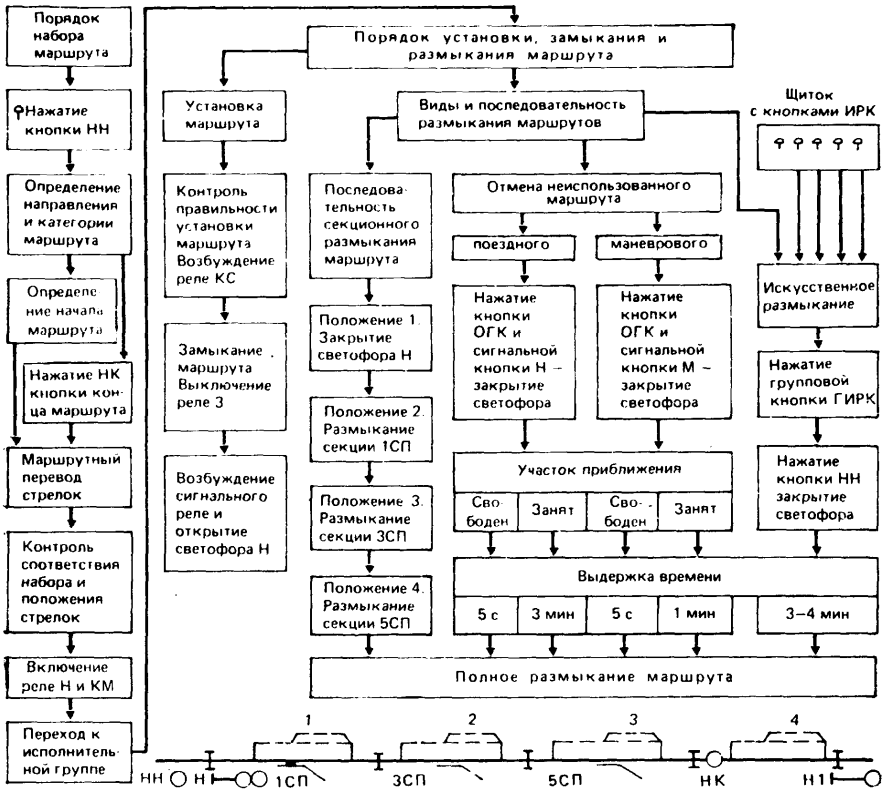


Рис. 9. Схема режимов работы БМРЦ

После готовности маршрута и открытия светофора размыкание маршрута производится в режимах автоматического секционного

размыкания, отмены маршрута и искусственного размыкания. Автоматическое размыкание происходит в процессе проследования поезда по секциям маршрута. При вступлении поезда на секцию 1СП (положение 1) закрывается светофор, и секция подготавливается к размыканию. После полного освобождения секций 1СП и занятия секции ЗСП (положение 2) размыкается секция 1СП. Аналогично размыкаются секции ЗСП и 5СП.

В случае невозможности использования маршрута его отменяют. Для этого нажимают кнопку групповой отмены ОГК и кнопку *Н* начала маршрута для закрытия светофора. После закрытия светофора маршрут размыкается с выдержкой времени 5 с при свободности участка приближения; в случае занятости этого участка—с выдержкой времени 3 мин. Аналогично отменяют маневровый маршрут с выдержкой времени 5 с при свободном участке приближения; I мин при занятом.

Искусственное размыкание производят нажатием кнопок ИРК путевых и стрелочных секций, входящих в маршрут, групповой кнопки искусственной разделки ГИРК, кнопки начала маршрута НН для закрытия светофора. После нажатия всех кнопок маршрут размыкается с выдержкой времени 3—4 мин.

ГЛАВА 6. КОМПЬЮТЕРНАЯ ЦЕНТРАЛИЗАЦИЯ

6.1. Принципы построения систем ЖАТ с использованием компьютерной техники

Нарушение экономических и научно-технических связей со странами СНГ и в основном с Россией, которая является основным поставщиком систем железнодорожной автоматики и телемеханики, а также комплектующей аппаратуры (при их резком удорожании, например, реле 1-го класса надежности оцениваются в настоящее время около 50 долларов за штуку) позволяет сформулировать единственно возможную концепцию по поддержанию на должном уровне работоспособности устройств жел.дор. автоматики и телемеханики, а также внедрению современной импортно-защищенной технологии – компьютеризацию устройств СЦБ, что позволит сократить постовые устройства ЭЦ в 2-3 раза. При этом ЭВМ возьмут на себя функции всех трех систем АБ, ЭЦ и ДЦ.

Внедрение ЭВМ позволит существенно повысить надежность всего комплекса устройств регулирования движения

поездов, а также практически без затрат, реализовать такие возможности работы устройств СЦБ, как максимальную автоматизацию процесса регулирования движения поездов, двухкратную попытку перевода стрелок при недоходе остряка до рамного рельса, автоматическую установку охранного маршрута и многие другие функции.

В частности если рассмотреть системы электрической централизации (ЭЦ), то совершенствование ЭЦ позволило существенно повысить оперативность работы дежурного, упростить порядок набора маршрута, увеличить наглядность поездной ситуации, а также осуществить управление с одного пульта удаленными объектами. Вместе с тем, резко возрос объем аппаратуры приходящейся на одну стрелку (1,2 статива на стрелку). Это значительно увеличивает капитальные и эксплуатационные затраты, снижает надежность действия ЭЦ. Дальнейшая модернизация релейных схем не позволяет улучшить функциональные возможности ЭЦ.

Наиболее целесообразным методом совершенствования систем ЭЦ является соединение микропроцессорного контроля и управления с релейными схемами реализации команд (управление и контрольные цепи стрелок и сигналов). При этом функциональные возможности микропроцессорной электрической централизации (МЭПЦ) должны быть не хуже, чем в эксплуатируемой ЭЦ. Эксплуатационные, надежностные параметры, а так же затраты на строительство и содержание МПЭЦ должны выгодно отличать ее от эксплуатируемой ЭЦ.

Самым существенным аргументом в пользу разработки микропроцессорной электрической централизации является:

- сокращение релейной аппаратуры на 70%;
- экономия энергии на тягу поездов;
- повышение пропускной способности;
- снижение эксплуатационных расходов по всем службам (оптимизация регулирования движения и скорости посредством ЭВМ сокращает необходимый штат дежурных, диспетчеров и др. работников, снижает износ подвижного состава пути, расходуемых материалов и т.д.).

Разрабатывая МПЭЦ на этапе окончательной доработки должна иметь, дополнительно уже к реализованным в устройствах ЭЦ следующие технические возможности:

1. Подача различных акустических сигналов (музыкальных фаз) при подходе поездов различных направлений и категорий, а также при завершении выполнения поездных и маневровых передвижений, в случаях неправильных действий или действий повышенной опасности дежурного по станции и др. эксплуатационных работников.
2. Оптическая сигнализация установки, использования и размыкания маршрутов с использованием цветовой палитры (установка поездного маршрута - зеленая полоса, маневрового – белая, занятость участка – красная; подробная детализация напольных элементов на дисплее и другая информация.
3. Возможность обращения к нескольким страницам дисплея (четная горловина, нечетная горловина, вся станция, указатели немаршрутных команд, указатели характера и времени подтверждений, справочные страницы)
4. Управление стрелками, сигналами, инициализация других команд осуществляется одним органом управления «мышью». А в особых случаях с клавиатуры. При этом задание поездных и маневровых маршрутов производится аналогично (нет маневровых или поездных элементов управления). Отмечается начало маршрута (участок приближения перед сигналом) и конец (приемо-отправочный путь, участок удаления, стрелочный или бесстрелочный участок). Для исключения совпадения манипуляций при установке маршрута с пути на перегон, маневровый составной маршрут задается как два раздельных маневровых маршрута.
5. На приемо-отправочных путях и всех контролируемых блок-участках перегона предусматривается индикация номеров поездов с отражением цветной палитры степени их опоздания и возможностью вызова другой информации об этих поездах путем фиксации стрелочного указателя на этом поезде с помощью «мыши».
6. Накопление поездных маршрутов с возможностью изменения порядка их реализации в любой момент времени.
7. Регулирование скоростью поезда на станциях и прилегающих перегонах с использованием пассивных датчиков для переключения локомотивных приемников.
8. Накопление маневровых маршрутов при угловых заездах и других случаях с индикацией о на набранных маршрутах и порядке реализации.

9. Телеуправление стрелками с локомотива.
10. Контроль места нахождения маневровых и поездных локомотивов с выдачей информации о их номере
11. Контроль заполнения путей без использования дополнительных путевых приборов.
12. Регулирование маневровыми передвижениями и скоростью движения маневровых локомотивов
13. Совмещение функций АБ, ЭЦ, ДС в программе одной ЭВМ с троированием систем обеспечения безопасности движения поездов и возможностью непродолжительной (до 12 часов) работы двух ЭВМ четырехкратным контролем правильности действия программы каждой ЭВМ.
14. С целью повышения надежности действия и устранения избыточных действий (переводов стрелок, обслуживания напольных устройств и др. операций) предусматриваются следующие дополнительные функции:

- установка маршрута осуществляется только после проверки всех требуемых условий его установки;
- очередность перевода стрелок в маршруте соответствует порядковым номерам стрелок в списке надежности (первой переводится малонадежная стрелка);
- отслеживание времени перевода стрелок с фиксацией и последующим накоплением отметок о тех стрелках, время перевода которых увеличивается;
- контроль понижения напряжения на путевом приемнике свободной рельсовой цепи относительно среднего значения напряжения с выдачей информации в процентном выражении упомянутого параметра и интервале времени несоответствия;
- коррекция напряжения на путевых приемниках за счет регулирования источников питания рельсовых цепей не имеющих существенных отклонений от нормы;
- трансляция информации о повреждениях на пост ДЦ с возможностью выдачи твердой копии о характере, времени и продолжительности повреждений или отклонении параметров;
- применение цифровой обработки сигналов с рельсовых цепей существенно повышает надежность самого малонадежного элемента железнодорожной автоматики – рельсовой линии.

Перечисленные новые технические возможности, кроме упомянутых экономических преимуществ (повышение пропускной способности, снижение затрат на топливо и др.) позволяет получить и

другие положительные качества, совокупность которых может быть даже еще важнее:

1. Повышение информативности, наглядности, удобства управления ориентации эксплуатационного персонала в поездной обстановке, маневровой работе, техническом обслуживании.
2. Повышение надежности устройств управления и контроля.
3. Автоматизация управления скоростным режимом поездов и маневровых передвижений.
4. Совмещение функций трех систем (ЭЦ, ДЦ, АБ) на базе ЭВМ имеющей четырехкратный режим контроля программ и трехкратное аппаратное дублирование
5. Сокращение времени строительства, проектирования, затрат на новое строительство и эксплуатацию в несколько раз.

6.2. Определение надежности использования ЭВМ в системах автоматики и телемеханики

Внедрение ЭВМ позволит существенно повысить надежность всего комплекса устройств регулирования движения поездов, а также практически без затрат, реализовать такие возможности работы устройств СЦБ, как максимальную автоматизацию процесса регулирования движения поездов, двухкратную попытку перевода стрелок при недоходе остряка до рамного рельса, автоматическую установку охранного маршрута и многие другие функции.

С точки зрения обеспечения надежности и безопасности функционирования систем управления и контроля, построенных с использованием компьютеров, будем считать, что необходимо соединить два и более компьютеров в систему. Для этого обоснования рассмотрим схему, состоящую из параллельно соединенных двух и более элементов (рис. 10).

Будем считать, что надежность каждого из элементов не зависит от отказа любого другого.

Если надежности отдельных элементов равны соответственно: p1, p2, ... , pn, то их ненадежности (т.е. вероятности отказа) будут: ql=l-pl; q2=l-p2; ... ; qn=l-pn, а результирующая надежность схемы (т.е. вероятность отказа работы всей схемы).

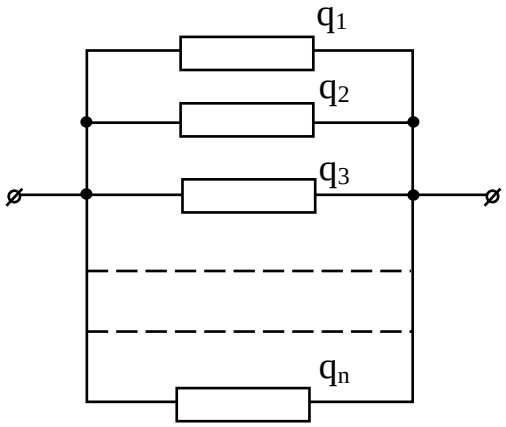


Рис 10. Параллельное соединение элементов

$$q\Sigma = q_1 q_2 q_3 .. q_n = \prod_{j=1}^{j=n} q_j = \prod_{j=1}^{j=n} (1 - pj) \qquad (1)$$

Надежность работы схемы, из параллельно соединенных элементов

$$p\Sigma = 1 - q\Sigma = 1 - \prod_{j=1}^{j=n} (1 - pj) \qquad (2)$$

j=l Для частей имеющих одинаковую надежность: ql=q2=q3= ... =qn=q, получим

$$q\Sigma = q^n = (1 - p)^n \qquad (3)$$

$$p\Sigma = 1 - q\Sigma = [1 - (1 - p)^n] \qquad (4)$$

Если считать, что вероятность безотказной работы вычислительного устройства p=0,8, то число параллельно включенных устройств при результирующем значение pE=0,99, должно быть равным:

$$p\Sigma = [1 - (1 - p)^n]$$

$$n = \frac{\lg(1 - p\Sigma)}{\lg(1 - p)} = \frac{lf\,(1 - 0{,}99)}{\lg(1 - 0{,}8)} = \frac{\lg(0{,}01)}{\lg(0{,}2)} = 2{,}86 \qquad (5)$$

Таким образом, для надежного функционирования системы компьютерной централизации необходимо подключение трех компьютеров.

6.3. Структурная схема компьютерной централизации

Компьютерная централизация в отличие от традиционной электрической имеет ряд особенностей. Основными из них являются: циклический контроль напольных объектов путевых и стрелочно-путевых участков, стрелок и сигналов; поддержание в возбужденном состоянии сигнальных и стрелочно-управляющих реле с проверкой всех условий безопасности движения в каждом цикле работы. Работа ПК происходит циклически, а время цикла колеблется от 02 до 0,0025с. в зависимости от количества одновременно установленных маршрутов, переводимых стрелок и наличия других управляющих команд.

Следует отметить, что указанное время составляет полный цикл, который состоит из двух малых циклов - нулевого и первого. В дальнейшем малый цикл будет называться просто циклом, а полный цикл - большим. Первый малый цикл, за малым исключением, содержит почти все части нулевого, но вместе с тем имеет и некоторые дополнительные. Оба малыхх цикла повторяются друг за другом непрерывно постянно проверяя наличие необходимых условий для поддержания в открытом состоянии сигналов или стрелояно-управляющих реле на время перевода стрелок. Замена кнопок пульта-табло на манипулятор, который называется мышью, влечет некоторые изменения в действиях дежурного по станции (ДСП) по установке маршрута, отмене и формированию других команд.

В программе ЭВМ совмещаются функции АБ, ЭЦ, ДЦ с троированием систем обеспечения безопасности движения поездов и возможностью работы трех ЭВМ с четырехктатным контролем правильности действия программы каждой ЭВМ.

Как было обосновано выше система должна состоять из трех параллельно включенных взаимосвязанных компьютеров А, В и С (рис.11), где (С-горячий резерв). Работа системы происходит следующим образом, компьютеры А и В, которые каждый в отдельности проверяет правильность задания, затем выдают соответствующие сигналы управления и контроля, которые проходят через блоки усиления и сопряжения. После срабатывания управляющих реле СУ1А и СУ1В, по принципу логической операции (и) происходит процесс исполнения задания (рис.12).

В программе каждой машины заложен контрольный бит, который проверяет рабочее состояние машины. Если например выходит из строя компьютер А, то выключается его контрольное реле (КА), подключающее компьютер С к компьютеру В, а выходы компьютера С к выходам компьютера А; при выходе из строя компьютера В выключается его контрольное реле (КВ), подключающее компьютер С к компьютеру А. а выходы компьютера С к выходам компьютера В, за счет чего не происходит сбоя в работе компьютерной централизации.

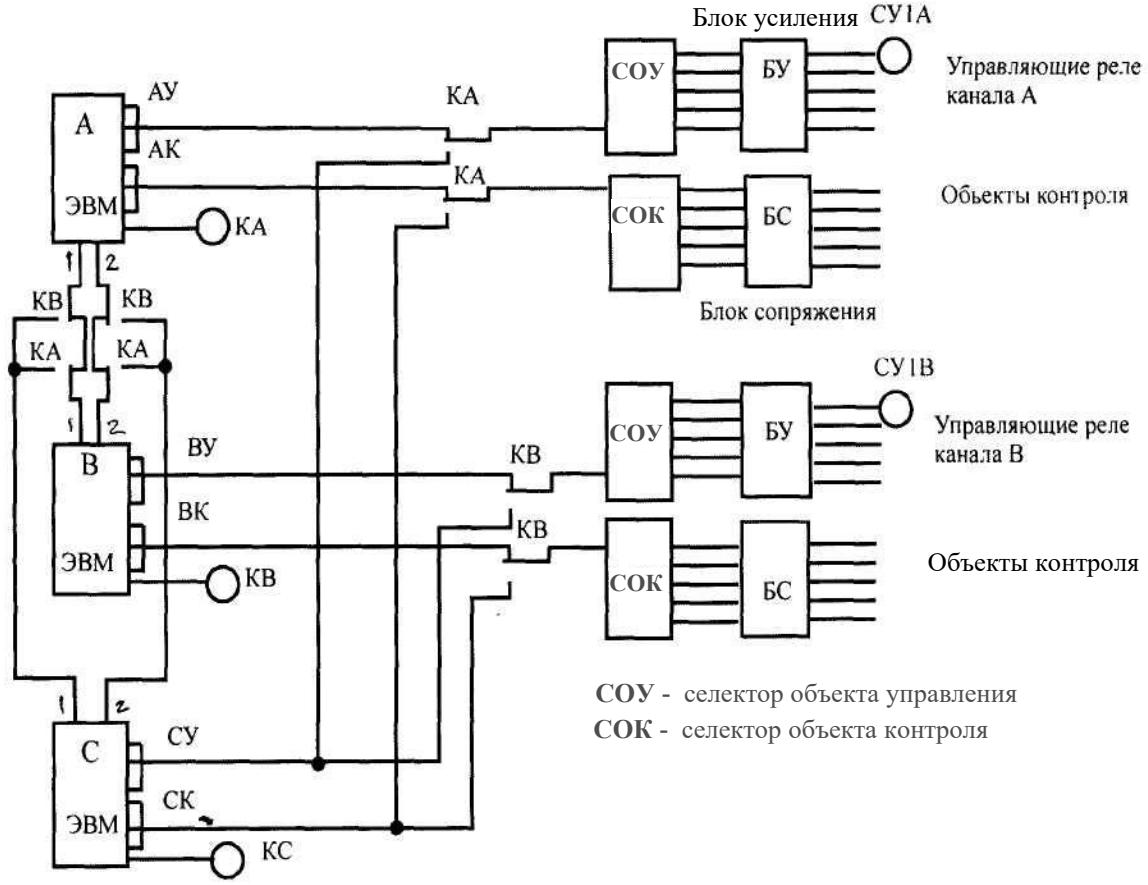


Рис. 11. Структурная схема компьютерной централизации

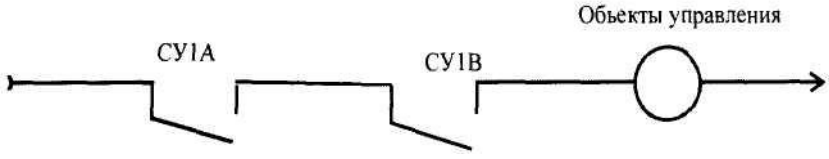


Рис. 12. Структурная схема логических операций

Для связи ПК с напольными объектами предусмотрены схемы, исключающие опасные отказы при повреждениях. Для каждого объекта предусмотрено по два управляющих реле, а контрольный объект транслирует информацию в ЭВМ посредством двух контактов.

Первые управляющие реле подключены к одной из схем сопряжения, которая соединена с компьютером через отдельный порт. Вторые управляющие реле имеют свою схему сопряжения, которая соединена с компьютером через другой порт.

Первые и вторые контакты контрольного реле также имеют индивидуальные схемы сопряжения, которые подсоединены тоже к отдельным портам.

Таким образом, для связи ЭВМ с напольными объектами предусматриваются 4 независимых схемы сопряжения, каждая из которых соединена с отдельным портом. Все эти порты предусматриваются на каждой из двух машин

6.4. Сопряжение ЭВМ с объектами управления и контроля

Сопряжение ЭВМ с объектами управления и контроля осуществляется через параллельные порты 278, 279, 378 и 379 рис. 12.

Для контроля объектов используются порты 278 (биты D4, D5, D6 и D7) 378 (биты D1-D7).

Четыре бита порта 278 предназначены для селекции 16 микросхем типа КР580ВА87 (DD3-DD15), к которым могут быть подключены 128 объектов контроля, через схему демультиплексора DD2 типа Л155ИД3. В программе каждому байту контроля присвоены свои объекты контроля рис. 13, например, контроль плюсового положения стрелки № 1 осуществляется по нулевому биту нулевого байта, а участка НАП четвертый бит первого байта.

Для повышения надежности передачи и приема сигнала контроля между схемами DD3-DD15 и контролируемыми контактами включены рис. 15 – общий выходной усилитель на транзисторе типа КТ805 и на каждый контакт контролируемого объекта входной усилитель на транзисторе типа КТ315.

Для передачи сигнала управления на управляющие реле рис. 12 используются порты 379 (биты D4, D5, D6, D7), 279 (биты D4, D5, D6, D7) и 278 (биты D0, D1, D2, D3).

Четыре бита (D0, D1, D2, D3). Порта 278 совместно с демультиплексором DD16 предназначены для выбора одной из 16

микросхем параллельного буфера данных КР80ВА87, через который передается сигнал управления на управляемый объект.

Так же, как и в программе контроля в модуле управления программы предусмотрены свои байты, например, для возбуждения третьего стрелочного управляющего реле сигнал управления передается по второму биту шестнадцатого байта.

Для приведения схемы передачи информации в исходное состояние после нулевого цикла программы в схеме передачи информации предусмотрен генератор импульсов частоты 4 МГц., который обнуляет все выходы микросхемы DD16 с данной частотой.

На выходе микросхем DD17-DD28 включены усилительные блоки БУР рис. 16.

Сопряжение с ЭВМ

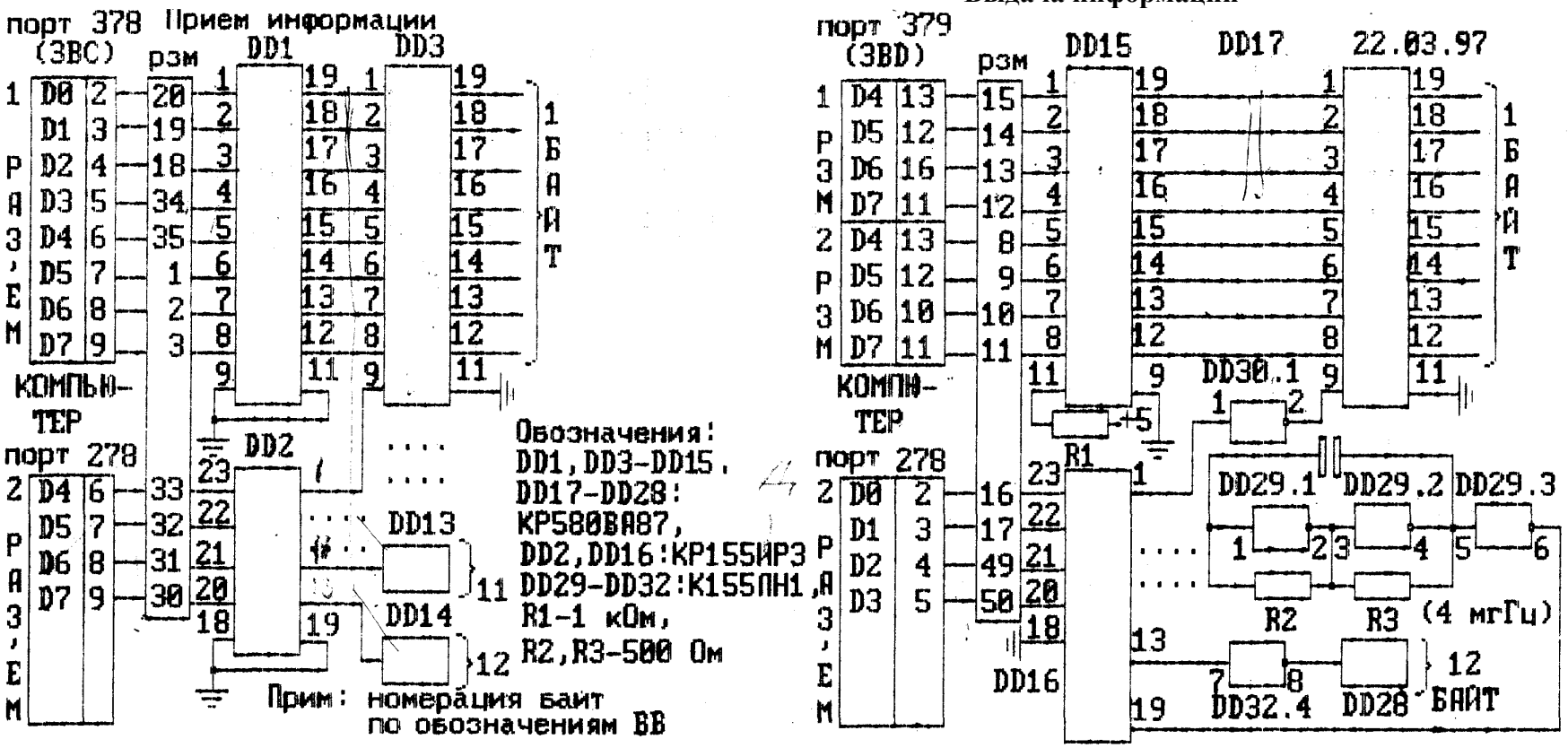


Рис.12

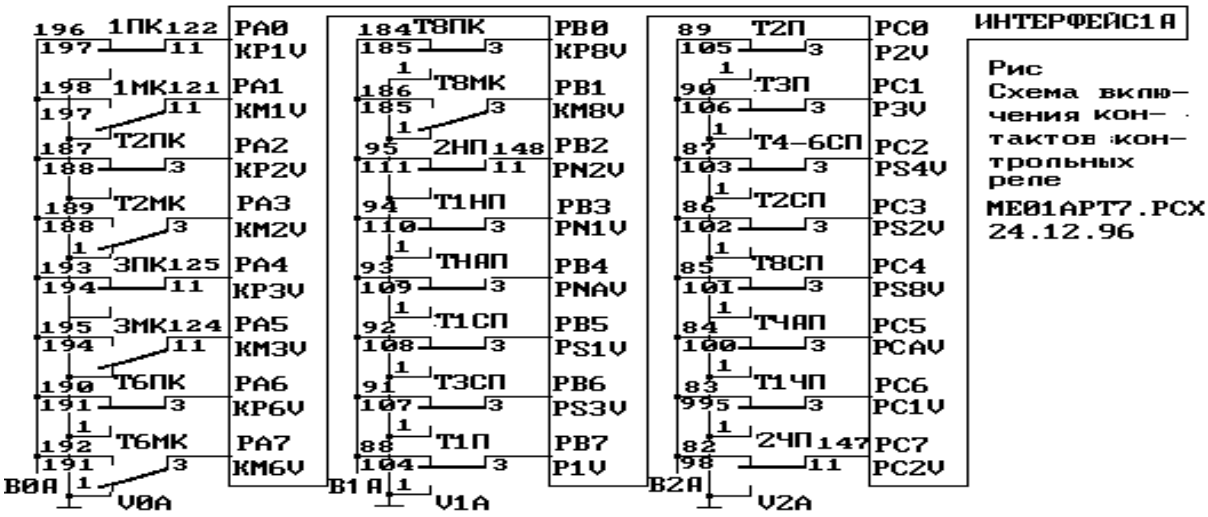


Рис.13

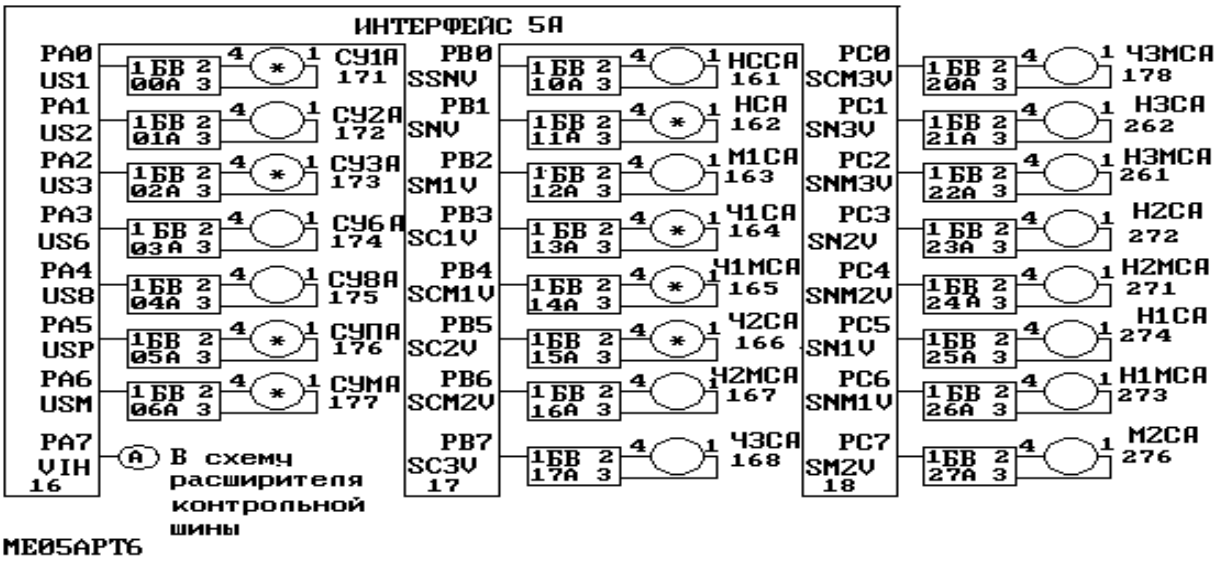


Рис. 14

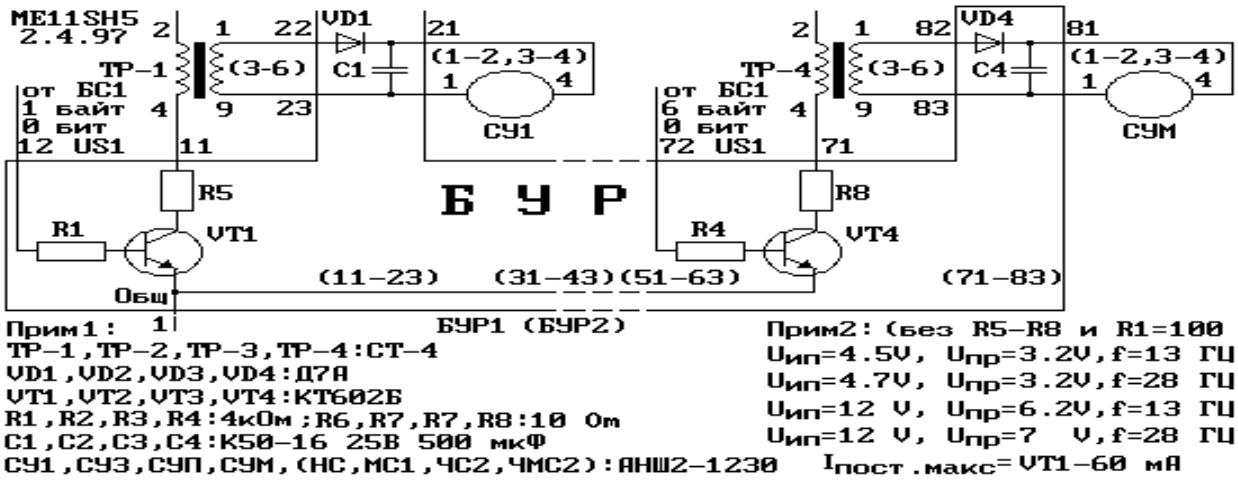


Рис. 16

Сопряжение 2 (вид со стороны монтажа)

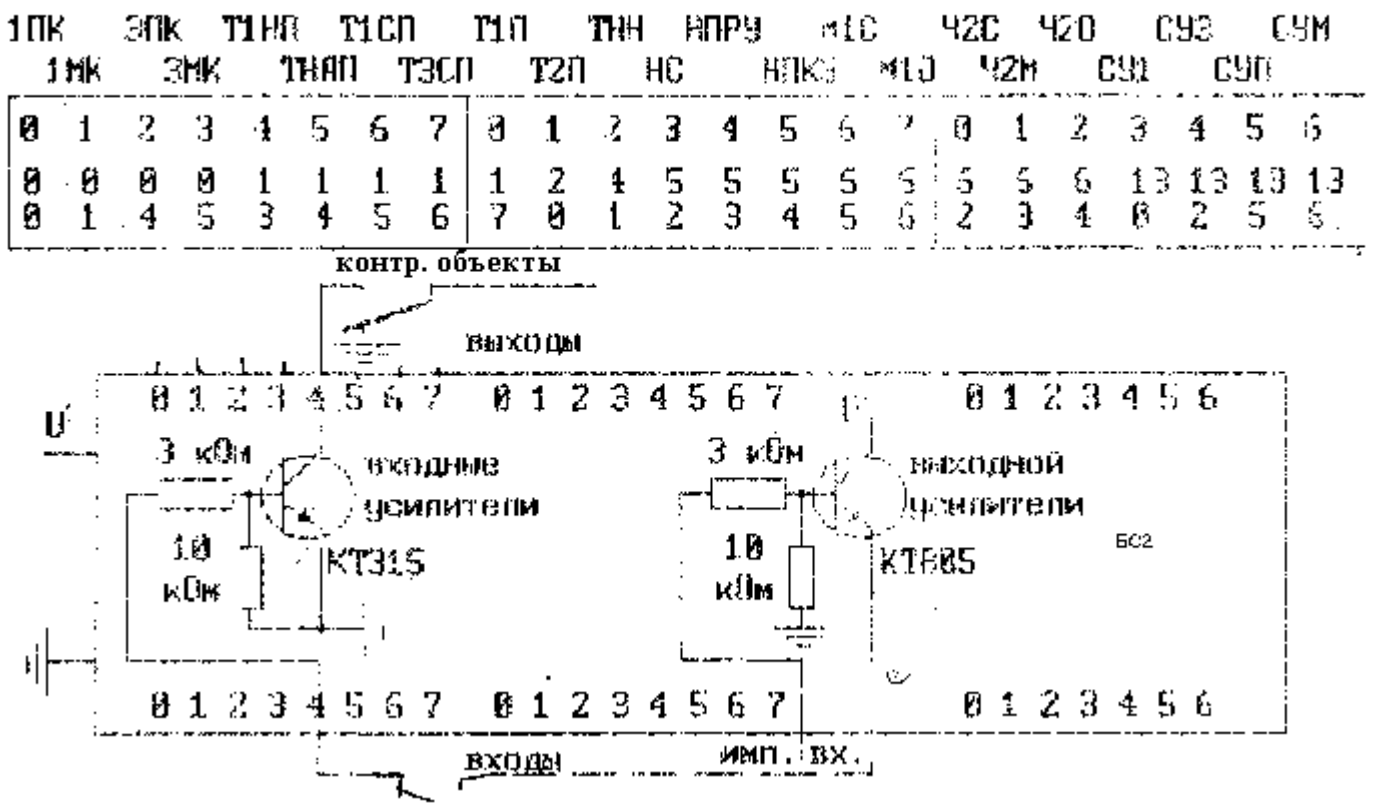


Рис. 15

Как было сказано ранее для повышения надежности и безопасности работы управляющих реле, они включаются от импульсов нулевого и первого циклов программы, т.е. на выходе ЭВМ подаются импульсы постоянного тока с частотой работы программы.

Усилительный блок БУР, как видно из схемы рис. 15 состоит из транзисторного усилителя КТ 606, импульсного трансформатора ТР, на выходе которого через диод и конденсатор включается управляющее реле. Схема усилителя собрана с учетом контроля всех опасных ситуаций.

Связь между компьютерами осуществляется по последовательному порту, как показано на рис. 18.

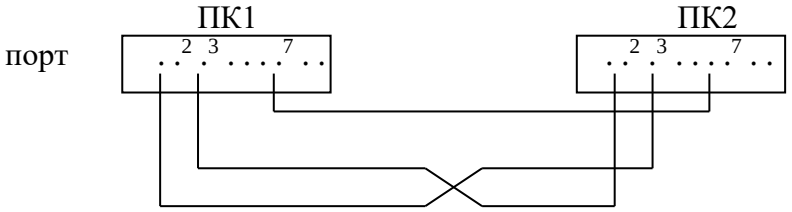


Рис. 18

6.5. Алгоритм установки и размыкания маршрутов компьютерной централизации

Разрабатываемая КЦ в плане обеспечения безопасности движения поездов не уступает существующим системам электрической централизации несмотря на то, что персональные ЭВМ не обладают свойствами приборов первого класса надежности. Кроме того в алгоритмы установки и разделки маршрутов могут быть внесены любые коррективы логики направленные на повышение безопасности движения без изменения монтажа на стативах.

Компьютерная централизация в отличии от традиционной электрической имеет ряд особенностей. Основными из них являются: циклический контроль напольных объектов - путевых и стрелочно-путевых участков, стрелок и сигналов; поддержание в возбужденном состоянии сигнальных и стрелочно- управляющих реле с проверкой всех условий безопасности движения в каждом

цикле работы. Работа обоих ПК происходит циклически, а время цикла колеблется от 02 до 0,0025 с в зависимости от количества одновременно установленных маршрутов, переводимых стрелок и наличия других управляющих команд. Следует отметить, что указанное время составляет полный цикл, который состоит из двух малых циклов - нулевого и первого. В дальнейшем малый цикл будет называться просто цикл - большим. Первый малый цикл, за малым исключением, содержит почти все части нулевого, но вместе с тем имеет и некоторые дополнительные. Оба малых цикла повторяются друг за другом непрерывно постоянно проверяя наличие необходимых условий для поддержания в открытом состоянии сигналов или стрелочно- управляющих реле на время перевода стрелок. Замена кнопок пульта-табло на манипулятор, который называется мышью, влечет некоторые изменения в действиях дежурного по станции (ДСП) по установке маршрута, отмене и формированию других команд.

Управление работой станции в компьютерной централизации осуществляется через компьютер, на дисплее которого изображается станция со всеми управляющими и вспомогательными пиктограммами рис. 19.

Установка маршрута осуществляется за счет наведения указателя мыши на шильдик сигнала, который отображен на дисплее, с последующим нажатием кнопки мыши, а затем указатель наводится на элемент пути конца маршрута и так же отмечается нажатием кнопки мыши. Указанные действия для большинства маршрутов малой станции однозначно воспринимаются ПК (программой ПК эквивалентной наборной группе), после чего выясняется направление движения и категория маршрута. Однако в некоторых случаях (установка вариантного маршрута, установка составного маршрута с пути с выездом на перегон, который совпадает с маршрутом отправления) указанный способ задания маршрута не позволяет установить категорию маршрута. Для установки поездных и маневровых вариантных маршрутов отмечается начало маршрута (шильдик открываемого сигнала), затем отмечаются маневровые сигналы на трассе маршрута, и, наконец, фиксируется конец маршрута. Если в начале маршрута стоит совмещенный сигнал, то категория маршрута определяется местом фиксации конца маршрута: маневровые заканчиваются в горловине, поездные - на участке удаления. Для маневрового маршрута с пути с выездом на перегон осуществляется набор двух маневровых маршрутов. Индикация

начала и конца маршрута в отличии от принятых в ЭЦ обозначений следующая: начало маршрута указывается мигающим от поездного или совмещенного светофора малиново-красным цветом; начало маршрута от малинового светофора - мигающим сине-голубым цветом; конец маршрута на занятый участок пути - мигающим красно-малиновым цветом; конец маршрута на участок пути не имеющий контроля свободности - мигающим черно-голубым цветом. В отсутствии поездов и замкнутых секций стрелочно-путевые участки и участки пути имеющие контроль свободности обозначаются синим цветом, а участки пути не имеющие такого контроля - черным. При занятии участка он окрашивается в красный, а при замыкании - в белый. Такая индикация соответствует принятой для ЭЦ. Так же в соответствии с ЭЦ предусмотрена индикация искусственного размыкания свободного (мигает сине-белый) и занятого участка (мигает красно-белый). Сохранена и индикация положения стрелки при наличии контроля положения стрелки, а также при его потере при свободном, замкнутом и занятом состоянии.

Отмена маршрута осуществляется посредством наведения указателя мыши на малиновый прямоугольник с надписью "Отмена маршрута " и нажатия на кнопку. При этом прямоугольник начинает мигать малиново-красным цветом. Для отмены упомянутого действия следует повторно нажать кнопку при наличии указателя в том же прямоугольнике.

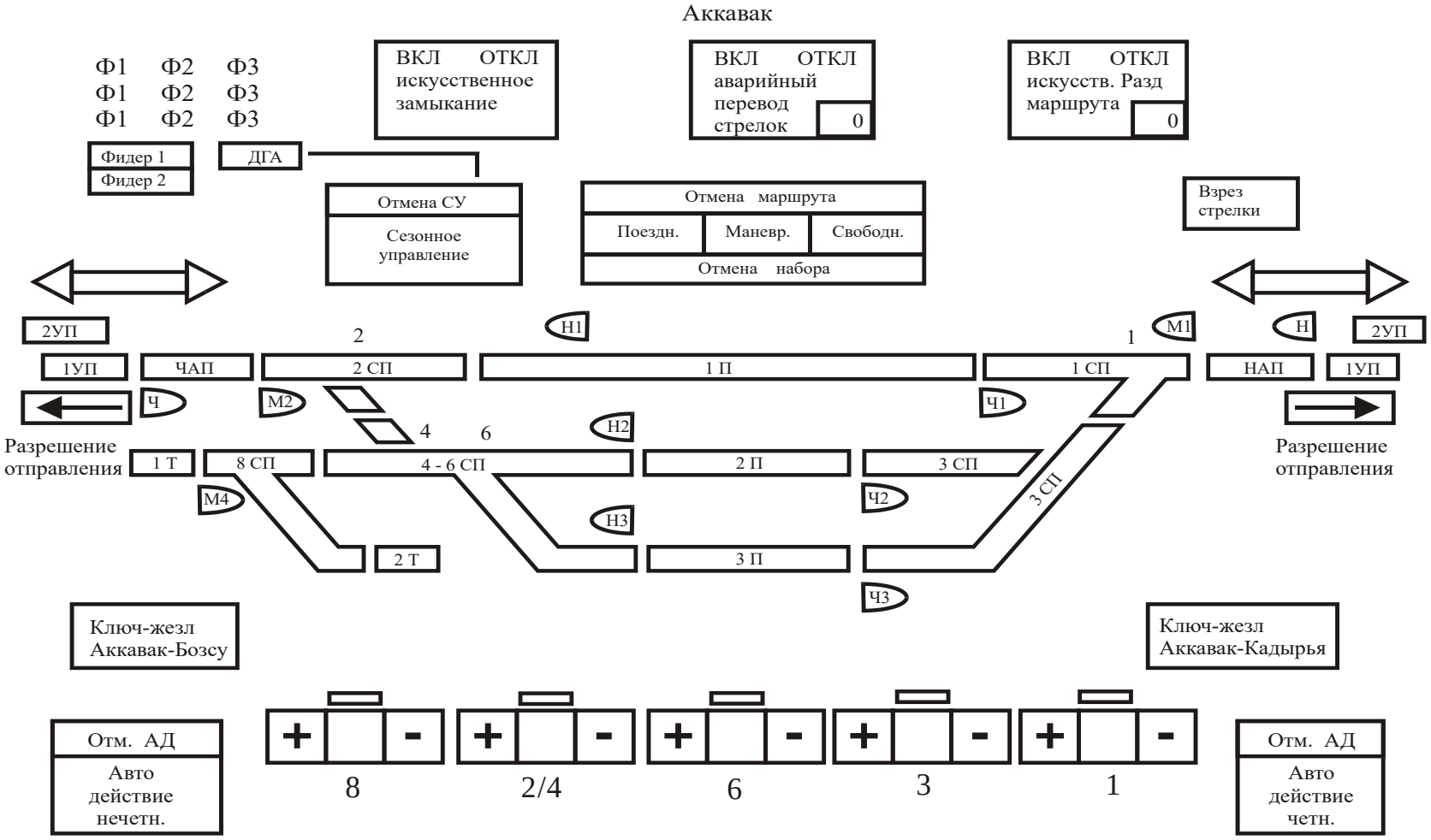

Аккавак
Ф1 Ф2 Ф3
Ф1 Ф2 Ф3
Ф1 Ф2 Ф3
Фидер 1
Фидер 2
ДГА
ВКЛ ОТКЛ
искусственное
замыкание
ВКЛ ОТКЛ
аварийный
перевод
стрелок
0
ВКЛ ОТКЛ
искусств. Разд
маршрута
0
Отмена СУ
Сезонное
управление
Отмена маршрута
Поездн.
Маневр.
Свободн.
Отмена набора
Взрез
стрелки
2УП
1УП
ЧАП
2
2 СП
Н1
1 П
1
1 СП
М1
Н
НАП
2УП
1УП
Ч
М2
Ч1
Разрешение
отправления
4
6
Н2
1 Т
8 СП
4 - 6 СП
2 П
3 СП
Разрешение
отправления
М4
Ч2
3 СП
Н3
2 Т
3 П
Ч3
Ключ-жезл
Аккавак-Бозсу
Ключ-жезл
Аккавак-Кадырья
Отм. АД
Авто
действие
нечетн.
8
2/4
6
3
1
Отм. АД
Авто
действие
четн.

Рис. 19

Для отмены маршрута при мигающем прямоугольнике "Отмена маршрута" следует навести указатель мыши на шильдик сигнала и снова нажать на кнопку. Эти действия ДСП приведут к тому, что сигнал будет перекрыт, мигающий прямоугольник будет индицироваться ровным красным цветом, а рядом с ним один из трех малиновых прямоугольников с надписями поездной, маневровый, свободный будет окрашен в красный цвет. Кроме того, значение счетчика числа отмен маршрутов увеличивается на единицу. Такая индикация сохранится до окончания отмены маршрута, когда упомянутые прямоугольники окрасятся малиновым цветом, а замкнутый маршрут обозначенный белой полосой разомкнется и эта полоса окрасится синим цветом. Для перекрытия сигнала, когда уже идет одна отмена, нужно навести указатель мыши на шильдик открытого сигнала и нажать кнопку. При этом сигнал перекрывается, а процесс размыкания начнется только после отмены предыдущего маршрута.

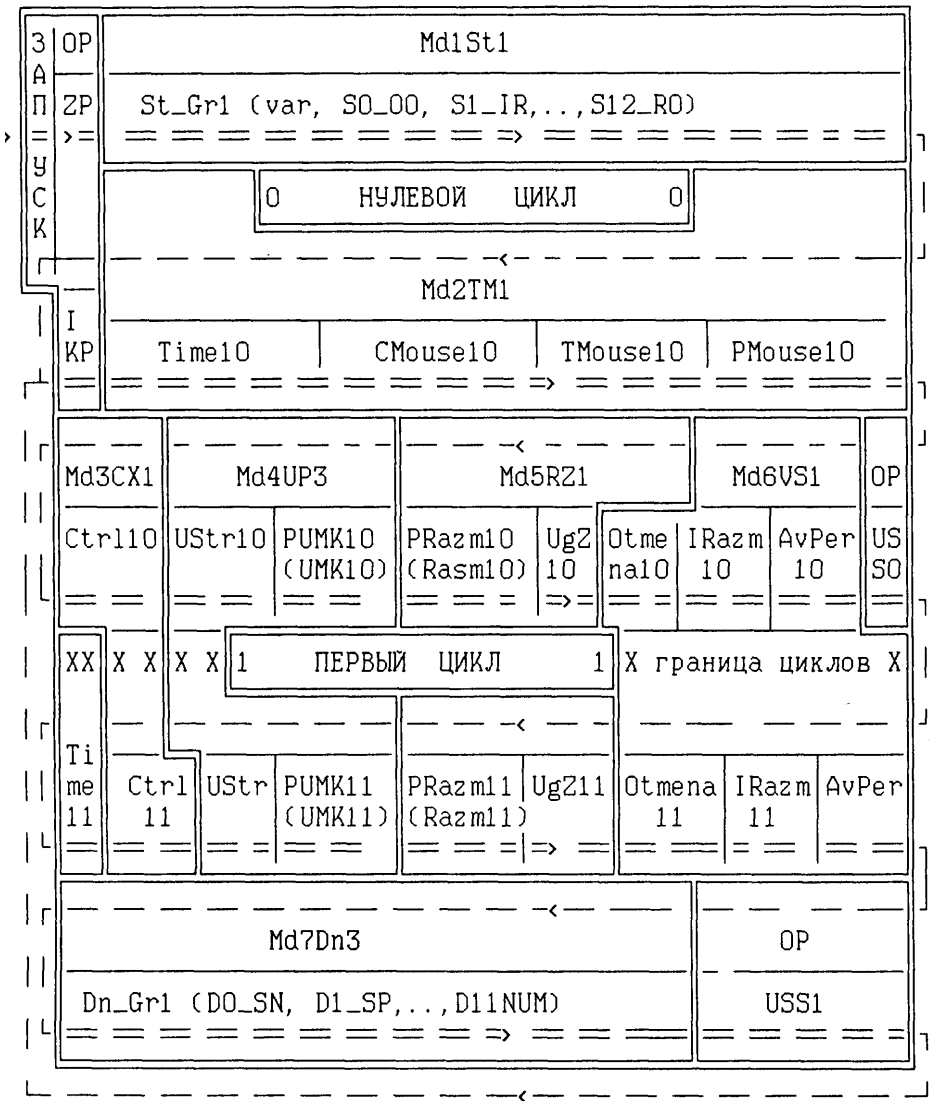
Искусственное размыкание осуществляется за счет наведения указателя мыши на малиновый прямоугольник с надписью "ИСКУССТВ. РАЗДЕЛКА" и нажатия на кнопку. При этом прямоугольник начинает мигать малиново-красным цветом. Далее указатель мыши наводится на обозначение участков путевого развития, а именно в те места, где указаны наименования участков и нажимается кнопка. При этом упомянутые обозначения свободных участков начинают мигать бело-синим, а занятые красно-синим цветом. Затем указатель мыши вновь наводится на прямоугольник "ИСКУССТВ. РАЗДЕЛКА" и нажимается кнопка. С этого момента прямоугольник окрашивается ровным красным цветом и начинается отсчет времени искусственной разделки, значение счетчика увеличивается на единицу. После окончания выдержки времени прямоугольник окрашивается малиновым цветом, а обозначения путевых участков в синий или красный цвет в зависимости от состояния этих участков.

Перевод ложно занятой стрелки осуществляется при наведении указателя мыши на прямоугольник "Вспомогательный перевод" и нажатии кнопки. Как и в предыдущих случаях прямоугольник начинает мигать малиново-красным цветом. Далее дежурный наводит указатель мыши на прямоугольник начинает мигать малиново-красным цветом. Далее дежурный наводит указатель мыши на прямоугольник соответствующий номеру и положению переводимой стрелки и нажимает кнопку. При этом значение

счетчика числа переводов увеличивается на единицу и начинается перевод стрелки. После окончания перевода или по истечению интервала времени предусмотренного для работы на фрикцию стрелка выключается.

В нижней части дисплея предусмотрены управляющие и контрольные стрелочные органы. Они соответствуют одноименным органам управления с использованием рукоятки. Индикатором рукоятки служит черная полоска над контрольным органом. При переводе стрелки имитируется поворот рукоятки за счет наведения указателя в левую или правую часть стрелочного контрольного органа и нажатия кнопки. При этом смещается черная полоса в сторону направления перевода и изменяется ее цвет на белый. Вместе с тем под контрольным стрелочным органом появляется красная полоса так же в направлении намеченного перевода и сигнализирует до окончания перевода или до окончания работы стрелки на фрикцию.

Алгоритм программы приведен на рис. 20 и состоит из алгоритма основной программы и алгоритмов модулей статической графики Md1St6; таймера и мыши Md2TM1 , контроля напольных объектов Md3CX1(Md3Ct1), управлении стрелками и сигналами Md4Up3 , размыкания стрелочно-путевых и бесстрелочных участков Md5Rz1 , вспомогательных операций (отмена, искусственное размыкание, вспомогательный перевод) Md6Vs1 ; динамической графики (вместе с алгоритмом индикации номера поезда) Md7Dn3 . Алгоритм модуля статической графики используется только для изображения конфигурации станции, контрольных и управляющих органов на экране дисплея, и выполняется ПК один раз при вводе программы. Посредством алгоритма динамической графики вводится новая информация на экран дисплея о изменении состояния путевых участков и сигналов, положении стрелок и др. Программа этого алгоритма может выполняться только в первом цикле. Алгоритм модуля таймера и мыши состоит из пяти процедур, первая, третья, четвертая и пятая из которых работает в нулевом цикле, а вторая в первом. Алгоритмы остальных модулей используются в обоих циклах одинаково.



Ри с. 20. Структурная схема программы компьютерной централизации.

Условные обозначения:
двойная пунктирная линия - работа программы;
одинарная пунктирная линия - переходы;
двойная сплошная линия - границы модулей;
одинарная сплошная линия - границы процедур;
IKP-имитация контроля состояния портов;
OP - основная программа;
USSO-управление стрелками и сигналами в нулевом цикле;
USSl-управление стрелками и сигналами в первом цикле;
ZP - запуск программы;
Timell расположена в модуле Md2TMl. 30-00,..,312-RO - процедуры.

Алгоритм основной программы (рис.20) осуществляет вызовы процедур из модулей, а отдельные его части UISK (установка

исходного состояния стрелочных коммутаторов), IVP имитация ввода с портов на момент отладки программы), UPO (управление портами в нулевом цикле), UP1 (управление портами в первом цикле). осуществляют непосредственно ввод и вывод информации.

Алгоритм основной программы начинается с объявления модулей и меток, затем во фрагменте "ЗАПУСК" осуществляется операция настройки для использования оверлеев, определяется драйвер дисплея, инициализируется графика, вводятся ограничения перемещения указателя мыши и осуществляется вызов процедуры статической графики St_Gr1 из модуля Md1ST6. Следующий фрагмент алгоритма описан в основной программе и называется "установка исходного состояния стрелочных коммутаторов" UISK . Логично бы было разместить его в процедуре статической графики St_Gr1, но в этом случае потребовалось бы дополнительная связь между модулем Md1St6 и Md7Dn3, что усложняет программу. Фрагмент основного алгоритма программы "имитация ввода с портов" IVP предназначен для отладки при работе ПК без ее подключения к блоку сопряжения с постовыми устройствами и служит для ввода данных имитирующих информацию с портов. Следующие четыре алгоритма Time10 , CMouse10 , TMouse10 и PMouse10 содержатся в модуле Md2TM1. Первый из этих блоков содержит таймеры предназначенные для исключения влияния дребезга контактов контрольных реле, замедлений при перекрытии сигналов, контроля возбуждения путевых реле при потере шунта, работе алгоритма размыкания маршрута при потере питания , луча питания рельсовых цепей, создания режима мигания цветов изображения сигналов, путевых участков и других обозначений на экране. Второй блок CMouse10 предназначен для фиксации позиции указателя мыши при переводе стрелок, установке и отмене маршрута, искусственной разделке и вспомогательном переводе стрелок и в других случаях. Третий TMouse10 и четвертый PMouse10 осуществляют соответственно трансляцию и прием информации о положении указателя мыши на второй ПК, работающий синхронно с первой. Предложенный алгоритм предусматривает сопряжение только двух ПК, т.к. на этапе разработки компьютерной централизации целесообразно отладить программные и аппаратные средства на малом комплексе. Следующий блок Ctrl10 составляет модуль Md2CX1 и предназначен для контроля состояния путевых участков, положения стрелок и сигналов. В этом блоке предусмотрена защита от влияния дребезга

и последствий потери шунта. Блоки USTR10 и PUMK10 с блоком UMK10 находятся в модуле Md4UP1 и предназначены для ручного управления стрелками, подготовке данных к использованию блока установки маршрута и реализации установки маршрута. Блоки PRAZM10 с блоком RAZM10 и блок UgZ10 содержатся в модуле Md5Rz1 и обеспечивают соответственно подготовку данных для использования в блоке размыкания маршрута, реализацию размыкания и размыкание неиспользованной части маршрута при угловых заездах (на средних и крупных станциях). Два следующих блока Otmena10 и AvPer10 находятся в модуле Md6Vs1 и предназначены соответственно для отмены маршрута и аварийного перевода ложно занятых стрелок. Последний блок нулевого цикла UP0 находится в основной программе и осуществляет воздействие на управляющие реле посредством управляющего порта. Первый цикл содержит блоки аналогичные тем, которые были описаны в нулевом, кроме блоков CMouse, TMouse и PMouse и дополнительно имеет блок Dn_Gr11, который осуществляет все изменения на экране дисплея в процессе работы программы. Размещение блоков первого цикла по модулям следующее: Md2TM1; Ctrl11 - в модуле Md3Ct1; USTP11 ,PUMK11 - в модуле Md4UP1; PRAZM11 RAZM11 и UgZ11 -в модуле Md5Rz1; Otmena 11 , AvPer11 - в модуле MdGVs1; UP1 в основной программе; Dn_Gr11 - в модуле Md7dn3.

6.6. Организация данных: статические и динамические данные, список, очередь

Для разработки программ компьютерной централизации рассмотрим небольшую станцию, показанную на рис. 21.

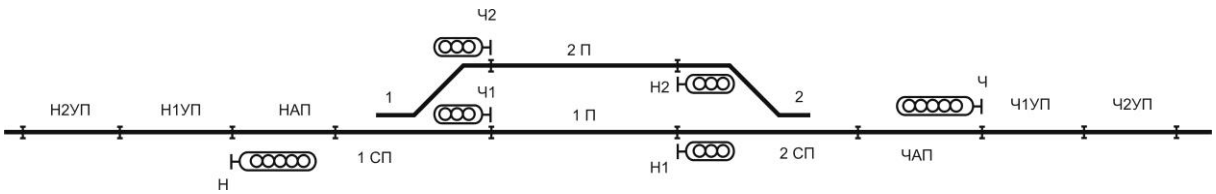


Рис. 21

Для этой станции в графическом режиме Borland-Pascal строим пульт-табло как показано на рис. 22.

Затем определяем переменные, необходимые для управления и контроля станционными объектами.

На каждый станционный участок необходимо иметь по пять переменных:

1. контроля состояния рельсовых цепей
2. контроля состояния стрелочного перевода
3. замыкающего реле
4. двух маршрутных реле

Переменную для контроля состояния рельсовых цепей обозначим через PS – стрелочная рельсовая цепь и P – бесстрелочная рельсовая цепь, например, для бесстрелочной рельсовой цепи участка НАП переменная будет иметь PNA, участка 1 СП – PS1, (1 пути P1, первого участка приближения P N 1 и т.д., а их повторители PNAW, PS1W, PN1W и т.д.

KP1 – контроль плюсового положения первой стрелки

Km1 – контроль минусового положения первой стрелки

SUP1 – стрелочное управляющее реле перевода в плюсовое положение первой стрелки

SUm1 – стрелочное управляющее реле перевода в минусовое положение первой стрелки

ZS1 – замыкающее реле стрелочной секции 1сп.

Sn – сигнальное реле выходного сигнала с первого пути

m1S1 – первое маршрутное реле стрелочного участка 1 сп

m2S1 – второе маршрутное реле стрелочного участка 1 сп

On – отмена набора

Om – отмена маршрута

Кроме вышеперечисленных переменных для организации движения по станции необходимы еще вспомогательные переменные

EE, EN – определение координат объектов управления на пульт-табло;

мр – задания маршрута

But-Status, X-coord, XC, Y – coord, YC - координат мыши

C-KM, C-KMW – изменения цвета светофоров;

C-S6, C-S6W - изменения цвета путевых путевых участков;

C-fxw, C-fxw, X, X1, X2, Y, Y1, y2 - вспомогательные переменные

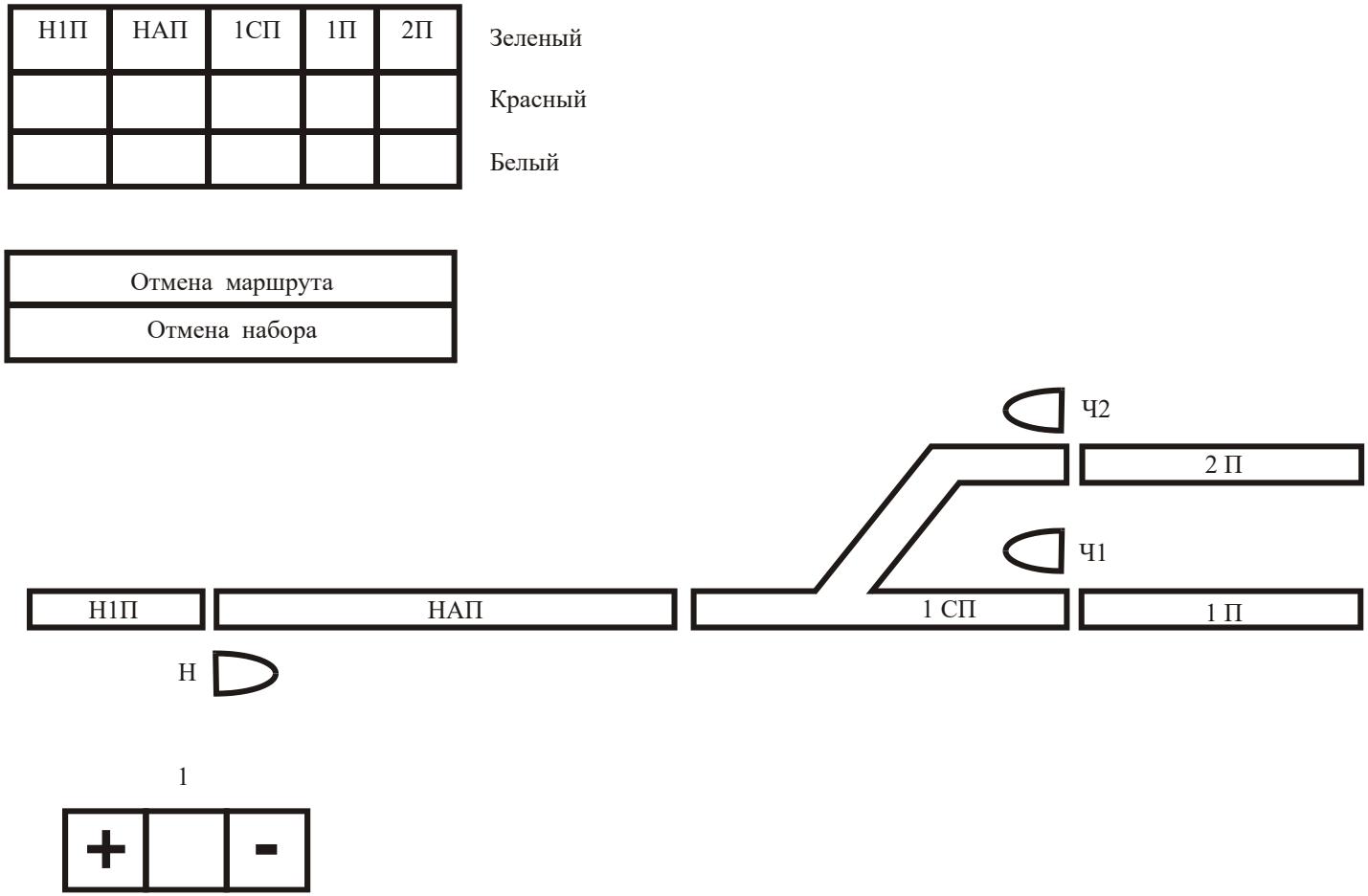

Н1П
НАП
1СП
1П
2П
Зеленый
Красный
Белый
Отмена маршрута
Отмена набора
Ч2
2 П
Ч1
Н1П
НАП
1 СП
1 П
Н
1
+
-

Рис.22

6.7. Программа статической графики. Схема станции

На табло дисплея строится станция со всеми обозначения, светофорами и стрелками.

Обычным режимом для экрана является текстовый. Для того, чтобы перевести экран в графический режим, используется процедура модуля Graph InitGraph : Init Graph (GD, GM, Path) – перевести экран в графический режим, где

GD – номер драйвера, GM – номер режима, Pafh – путь доступа к файлу, содержащему нужный драйвер. Если переменная Pafh содержит просто пустую строку (Pafh = ‘ ‘), то драйвер ищется в текущем каталоге.

GM и GD являются переменными параметрами. Если при запуске InitGraph переменная GD равна нулю, то нужный драйвер и оптимальный графический режим для этого драйвера определяется автоматически.

Для автоматического определения драйвера нужно ввести константу Detect, приравняв ее нулю.

Для закрытия графического режима используется процедура CloseGraph.

Следующая программа инициализирует графический режим дисплея.

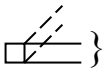
```
{ Инициализация графики}
 Begin
   GD:=Detect;
   InitGraph(GD,GM,'C:\TP7\BGI');
   M_Horiz_Range(0,633);
   M_Vert_Range(0,478);      размеры экрана

        {  Настройка графического экрана }
   SetFillStyle(1,8);       {заполнение всего экрана серым цветом}
   Bar(0,0,GetMaxX,GetMaxY); {рисование прямоугольного экрана}
   SetLineStyle(0,0,1); {уст. Текущий стиль, шаблон, толщина}
```

После инициализации графики можно перейти к статической графики (рисуем схему станции)

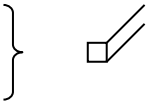
```
      {  СТАТИЧЕСКАЯ ГРАФИКА }
   setcolor(1);
   OutTextXY(500,10,'1  2  3  4  5  6');
```

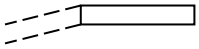
1. SetFillStyle(1,1); { установить текущий стиль и цвет наполнения (непрерывное наполнение синим цветом)}

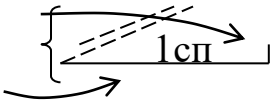
2. bar(0,270,30,280); { участок Н1П}
3. bar(130,270,140,280); {шильдик стрелки  }

4. line(140,270,170,240); {  линия }

5. line(140,280,180,240); {  линия }



6. bar(170,240,220,250); {  }

7. bar(225,240,319,250); { 2п }
8. floodfill(145,270,1);{заливает синим цветом область, указанную в пунктах 4,5}
9. line(145,280,155,270); {стрелка}
10. line(155,270,220,270); {стрелка}
11. line(220,270,220,280);
12. line(220,280,145,280); 

13. floodfill(190,275,1); {
14. bar(225,270,319,280); { участок 1 п }

Установка светофоров

setcolor(0); {рисование черным цветом}
{н1}
ellipse(40,300,240,120,10,5);
line(35,295,35,305);
SetFillStyle(1,4);
floodfill(45,300,0); { закрасить красным цветом }
{ч1}
ellipse(215,230,60,300,10,5);
line(220,225,220,235);
setfillstyle(1,4);
floodfill(215,230,0);
{ч2}
ellipse(215,260,60,300,10,5);
line(220,255,220,265);
setfillstyle(1,4);
floodfill(215,260,0);

Установка надписей

```
setcolor(0); {черный цвет}
   settextstyle(0,0,1);
   outtextxy(24,298,'Н');
   outtextxy(225,227,'Ч2');
   outtextxy(225,256,'Ч1');
   setcolor(8);      {серый цвет}
   outtextxy(2,271,'Н1П');
   outtextxy(63,271,'НАП');
   outtextxy(182,271,'1СП');
   outtextxy(258,241,'2П');
   outtextxy(258,271,'1П');
```

Установка иммитатора продвижения поезда

Для иммитации продвижения поезда по станции установим иммитатор с указанием всех участков по которым будет проходить подвижная единица. В данном случае мы имеем участок приближения Н1П, бесстрелочный участок НАП, стрелочный участок 1сп и два пути 1п и 2 п. Примем условно цвета для свободного участка – зеленый цвет, занято – красный цвет и заданного маршрута – белый цвет.

В верхней части экрана изобразим следующий рисунок.

(10,20)

Н1П	НАП	1СП	1П	2П

- зеленый цвет
- красный цвет
- белый цвет

(260,65)

Программа графики иммитатора будет следующей

```
setcolor(0);
   Rectangle(10,20,260,80); {чертит прямоугольник с левым верхним углом 10,20  и правым нижним 260, 80}

   line(10,35,260,35);
   line(10,50,260,50);
```

```
 line(10,65,260,65);
 SetFillStyle(1,2);
 floodfill(12,21,0);      {закрашивает зеленым цветом}
 SetFillStyle(1,4);
 floodfill(12,36,0);      {закрашивает красным цветом}
 SetFillStyle(1,15);
 floodfill(12,51,0);      {закрашивает белым цветом}
 Rectangle(10,20,60,80);
 Rectangle(10,20,110,80);     { рисует ячейки }
 Rectangle(10,20,160,80);
 Rectangle(10,20,210,80);
 setcolor(0);
 outtextxy(25,25,'Н1П');
 outtextxy(75,25,'НАП');
 outtextxy(125,25,'1СП');     { устанавливает надписи }
 outtextxy(175,25,'1П');
 outtextxy(230,25,'2П');
```

Установка стрелок

```
 Rectangle(115,380,175,400);
 Rectangle(115,380,135,400);
 Rectangle(115,380,155,400);
 SetFillStyle(1,2);
 floodfill(125,390,0);
 settextstyle(0,0,2); {устанавливает шрифт, размеры символов}
 outtextxy(120,365,'+');
 outtextxy(140,365,'1');
 outtextxy(160,365,'-');
 settextstyle(0,0,1);
```

Установка кнопок отмены набора и маршрута {отмены}

```
 setcolor(15);   { цвет белый}
 Rectangle(10,110,260,140); {прямоугольник}
 line(10,125,260,125);
{маршрута}
 SetFillStyle(1,5); {заполнение малиновым цветом}
 floodfill(12,111,15);
 outtextxy(70,114,'ОТМЕНА МАРШРУТА');
{набора}
```

```
SetFillStyle(1,7);
floodfill(12,130,15);
outtextxy(77,129,'ОТМЕНА НАБОРА');
```

6.8. Условия установки и размыкания маршрута Инициализация данных

Прежде, чем перейти к основной программе еще раз рассмотрим условия установки и размыкания маршрутов.

При установке маршрута необходимо выполнить следующие основные условия:

1. Контроль свободности маршрутных участков
2. Контроль положения стрелок в соответствии с маршрутом
3. Контроль незамкнутого состояния стрелочно-путевых и бесстрелочных участком с последующим контролем замыкания
4. Контроль отсутствия враждебных маршрутов совпадающих по положению стрелок с задаваемыми
5. Контроль отсутствия лобовых маршрутов
6. Контроль целостности нити лампы разрешающего огня (начинается через 3 сек. После возбуждения сигнального реле).
7. Контроль положения охраняемых стрелок
8. Контроль состояния негабаритных участков
9. Контроль направления движения на перегоне (в маршрутах отправления)
10. Контроль наличия ключа жезла (в маршруте отправления)
11. Контроль отсутствия отмены маршрута
12. Контроль отсутствия искусственной разделки маршрута

Итак для рассматриваемой станции, например, при приеме поезда на 1-й путь необходимо выполнить следующие условия:

1. Участки НАП ↑ 1СП ↑1П ↑- свободны (рядом стрелка указывает на то, что путевое реле под током

2. Стрелка № 1 находится в + положении (реле КР1 ↑)

3. Замыкающие и маршрутные реле этих участков под током – ZNA↑, ZS1 ↑, Z1 ↑, M1NA↑, M2NA ↑, M1S1 ↑ , M2S1↑, m11+, M21 ↑

4. Так как станция небольшая, то здесь отсутствуют враждебные маршруты.

5. Реле NI1↑(исключающее реле лобовых маршрутов первого пути).

6. Реле ORO ↑ (огневое реле разрешающего огня).

7. Охранных стрелок на данной станции нет.
8. Негабаритных участков нет.
9. Задается маршрут приема.
10. Задается маршрут приема.
11. Реле ОМ ↑ отмены маршрута – маршрут не был задан.
12. Реле IR ↑- искусственной разделки – отсутствует.

Если теперь рассмотреть маршрут приема на 2 путь, то для выполнения этих условий необходимы те же реле, но с другими индексами.

Кроме этого для открытия светофоров потребуются и сигнальные реле SN, SC1, SC2 и их повторители SNW, SC1W, SC1V, SC2W, SC2V, а также управляющие реле перевода стрелок UM1 – управляющее реле перевода стрелки в минусовое положение;
UP1 – управляющее реле перевода стрелки в плюсовое положение и контрольные реле контроля положения стрелки KP1 (плюсовое) KM1 (минусовое).

Если реле под током ↑ , то переменная соответствующая данному реле будет иметь значение TRUE – истинно, и наоборот FALSE – ложно. Для чего воспользуемся еще двумя переменными, которые будут принимать разное значение в зависимости от состояния переменной.

6.8.1. Условия размыкания секций при использовании поездного маршрута

1. При освобождении участка приближения:

- контроль занятия рассматриваемой секции и размыкания предыдущей (для первой секции освобождения предыдущей;
- контроль занятия последующей секции, освобождения рассматриваемой при выполнении вышеперечисленных условий.

2. При неосвобождении участка приближения:

- контроль поочередного занятия всех секций маршрута с последующим контролем освобождением всех секций.

6.8.2. Условия размыкания секций при использовании маневрового маршрута

В отличие от условий размыкания поездного маршрута в маневровом маршруте не контролируется освобождение предмаршрутного участка, а также занятия последующего участка и

освобождения рассматриваемого для бесстрелочного участка (участок НАП)

6.8.3. Фрагмент программы объявления переменных

```
VAR
    Lest_ Button, Lest_ Button W,
    Right_ Button.
    F,t.
    TF, tfw
    KP1, KP1W, SUP1, Km1, Km1W, Sum1,
    pn1, pn1W, pna, pnaw, PS1, PS1W, P1, P1W, P2, P2W, P1V, P2V,
    pn1V, Zn1, Zn1W, Zna, Znaw, ZS1, ZS1W, Z1, Z1W, Z2, Z2W,
    Sn, SnW, SC1, SC1W, SC2, SC2W, SnV, SC1V, SC2V,
    onv, onw, omv, omvw,
    m1na, m1S1, m11, m12,
    m2na, m2S1, m21, m22        : Boolean;

    U1, U1W, V2, V2W, EE, EN, mp: Byte;
    But_Status;
    X_Coord, XC;
    Y_Coord, YC;
    GD, GM, : : INTEGER;
    C_KM, C_KMW, C_SG, C_SGW, C_fx, C_fxw, X, X1, X2, Y, Y1,
Y2,
    Hour, Minute, Second, Sec100:Word;
    XS, YS:string;
```

Переменные объявлены, теперь их необходимо привести в исходное состояние, то есть станция свободна и на всех светофорах горят запрещающие огни.

Инициализация исходных данных

```
f:=false;       p2:=t;      sc2:=f;      m21:=t;
t:=true;        zn:=t;      m1na:=t;     m22:=t;
kp:=t;          zna:=t;     m2na:=t;
pn1:=t;         zs1:=t;     m1s1:=t;
pna:=t;         z1:=t;      m2s1:=t;
ps1:=t;         z2:=t;      m11:=t;
p1:=t;          sn:=f;      m12:=t;
sc1:=f;
```

6.9. Установка и размыкание маршрута

Теперь рассмотрим программу установки и размыкания маршрута

```
M1:
 {   НАЧАЛО  ПРОГРАММЫ  НУЛЕВОГО  ЦИКЛА   }
{3 Опрос мыши}
 Left_ButtonW:=Left_Button ;  xc:=0;ee:=0;yc:=0;
 M_Check(But_Status,X_Coord,Y_Coord);
 Analiz(But_Status,Left_Button,
       Middle_Button,Right_Button);

{\ Индикация коорд. курсора}

  If Left_ButtonW>Left_Button THEN
    Begin
      XC:=X_Coord;YC:=Y_Coord;
      M_Hide;
      SetFillStyle(1,0); Bar(0,0,80,8);
      IF XC>0 THEN STR(XC,XS);
      IF XC>0 THEN STR(YC,YS);
      SetColor(15);OutTextXY(0,0,XS);OutTextXY(30,0,YS);
    End;
{ TIME\ }
    GetTime( Hour,Minute,Second,Sec100);

 {отмена маршрута}
    if (tfw<>tf)and(om)AND((SNV)or(sc1v)or(sc2v))then
       begin tfw:=tf;i:=i+1; end;
 {св,отмены}
    IF SEC100>50 THEN TF:=T ELSE TF:=F;
    IF TF THEN
       begin
         C_KM:=5;
       end
    ELSE
       begin
         C_KM:=4;
       end;
```

```
 {1п миг}
   IF SEC100>50 THEN TF:=T ELSE TF:=F;
   IF TF THEN C_SG:=1 ELSE C_SG:=9;

{ TIME/ }

{ Определени E\ }
 {стрелка}
   IF( yC>380) and(yc<400)THEN
     BEGIN
       IF( XC>115)and(xc<135)and zs1 and ps1 THEN
         BEGIN kp1:=t;km1:=f;END;
       IF( XC>155)and(xc<175)and zs1 and ps1 THEN
         BEGIN km1:=t;kp1:=f;END;
       IF( XC>135)and(xc<155) THEN
         BEGIN km1:=f;kp1:=f;END;
     END;
{участки}
  if(yc>20)and (yc<35)then
     begin
       if(xc>10)and(xc<60) then  begin pn1:=t;end;
       if(xc>60)and(xc<110) then begin pna:=t;end;
       if(xc>110)and(xc<160) then begin ps1:=t;end;
       if(xc>160)and(xc<210) then begin p1:=t;end;
       if(xc>210)and(xc<260) then begin p2:=t;end;
     end;
  if(yc>35)and (yc<50)then
     begin
       if(xc>10)and(xc<60) then  begin pn1:=f;end;
       if(xc>60)and(xc<110) then begin pna:=f;end;
       if(xc>110)and(xc<160) then begin ps1:=f;end;
       if(xc>160)and(xc<210) then begin p1:=f;end;
       if(xc>210)and(xc<260) then begin p2:=f;end;
     end;
  if(yc>50)and (yc<65)then
     begin
       if(xc>60)and(xc<110) then  begin zna:=f;end;
       if(xc>110)and(xc<160) then begin zs1:=f;end;
       if(xc>160)and(xc<210) then begin z1:=f;end;
       if(xc>210)and(xc<260) then begin z2:=f;end;
```

```
    end;
 if(yc>65)and (yc<80)then
    begin
      if(xc>60)and(xc<110) then  begin zna:=t;end;
      if(xc>110)and(xc<160) then begin zs1:=t;end;
      if(xc>160)and(xc<210) then begin z1:=t;end;
      if(xc>210)and(xc<260) then begin z2:=t;end;
    end;

{ светофор}
 {неч}
 if(yc>295)and(yc<305)then
    begin if(xc>35)and(xc<50) and((not sn)or(om))
          and((not sc1)or(not sc2))then
                begin snv:=t;end;
    end;
 {ч1}
 if(yc>255)and(yc<365)then
    begin if(xc>200)and(xc<220)and((not sc1)or(om))
                then begin sc1v:=t;end;
    end;
 {ч2}
 if(yc>225)and(yc<235)then
    begin if(xc>200)and(xc<220) and((not sc2)or(om))
                then begin sc2v:=t;end;
    end;

{1п,2п миг}

 if(xc>225)and(xc<319) then
    begin
      if(yc>240)and(yc<250)then begin p2v:=t;yc:=0;end;
      if(yc>270)and(yc<280)then begin p1v:=t;yc:=0;end;
    end;
{н1п,миг}
 if(yc>270)and (yc<280)then
    begin if(xc>1)and(xc<30)then
          begin pn1v:=t;yc:=0;end;
    end;
{отмены}
```

```
  if(xc>10)and(xc<260) then
     begin if(yc>110)and (yc<125)then begin om:=t;end;
          if(yc>125)and (yc<140)then begin on:=t;end;
     end;

{установка маршрута}
if snv then begin ee:=1 ;end;
if sc1v then begin ee:=2 ;end;
if sc2v then begin ee:=3 ;end;
if p1v then begin ee:=4 ;end;
if p2v then begin ee:=5 ;end;
if pn1v then begin ee:=6 ;end;

if (ee<4) then en:=ee;
if (en=1)and(ee=4) then mp:=1;
if (en=1)and(ee=5) then mp:=2;
if (en=2)and(ee=6) then mp:=3;
if (en=3)and(ee=6) then mp:=4;

{размыкание маршрута}

   {НЕЧ}
 if mp<3 then
  begin
{НАП}
    if not M2NA and  not PNA  and PN1 then M1NA:=t;
    if  M1NA and  PNA  and not PS1 then M2NA:=t;
    if M1NA and M2NA then begin ZNA:=t; end;
{1СП}
    if not M2S1 and  not PS1  and ZNA  then M1S1:=t;
    if M1S1 and PS1 and
     ((not P1 AND  KM1)OR(not P2 AND  KP1))then M2S1:=t;
    if M1S1 and M2S1 then begin ZS1:=t; end;
{1П}
    if not M21 and(not P1)and ZS1 then
       BEGIN M11:=t;MP:=0;END;
    if M11 and P1 then M21:=t;
    if M11 and M21 then begin Z1:=t; end;
{2П}
    if not M22 and(  not P2)  and ZS1 then
```

```
      BEGIN M12:=t;MP:=0;END;
    if M12 and P2 then M22:=t;
    if M12 and M22 then    begin Z2:=t; end;
  end;

   {ЧЕТ}
 if mp>2 then
  begin
{1СП}
   if not M2S1 and(not PS1)and( p1 OR p2)then M1S1:=t;
   if M1S1 and (P1 OR P2) then M2S1:=t;
   if M1S1 and M2S1 then    begin ZS1:=t; end;
{НАП}
   if not M2NA and(not PNA)and ZS1 then M1NA:=t;
   if M1NA and PS1 then M2NA:=t;
   if M1NA and M2NA then begin ZNA:=t;mp:=0;M1NA:=f;end;
  end;

{Определени E/}

        { UMK }
{установка маршрута}
 if (mp=1) then
  begin
    if(pna)and(ps1)and(p1)and (zna)
      and (zs1) and (z1) then um1:=t;
    if um1 then begin kp1:=f;km1:=t;p1v:=f;end;
    if (pna)and(ps1)and(p1)and (zna)
      and (zs1) and (z1)and  um1 and km1 then
         begin zna:=f;zs1:=f;z1:=f;sn:=t;
              sc1v:=f;sc2v:=f;m1na:=f;m1s1:=f;m11:=f;
              m2na:=f;m2s1:=f;m21:=f;
         end;
  end;
 if (mp=2) then
  begin
    if(pna)and(ps1)and(p2)and (zna)
      and (zs1) and (z2) then up1:=t;
     if up1 then begin km1:=f;kp1:=t;p2v:=f;end;
     if (pna)and(ps1)and(p2)and (zna)
```

```
        and (zs1) and (z2)and  up1 and kp1 then
           begin zna:=f;zs1:=f;z2:=f;SN:=T;sc1v:=f;
                 sc2v:=f;m1na:=f;m1s1:=f;m12:=f;m2na:=f;
                 m2s1:=f;m22:=f;
           end;
  end;
if (mp=3) then
  begin
    if(pn1)and(pna)and(ps1)and (zna)
       and (zs1) and (z1) then um1:=t;
    if um1 then begin kp1:=f;km1:=t;pn1v:=f;end;
    if(pn1)and (pna)and(ps1)and (zna)
      and (zs1) and (z1)and  um1 and km1 then
         begin zna:=f;zs1:=f;sc1:=t;snv:=f;sc2v:=f;
                m11:=f;m1s1:=f;m1na:=f;m2na:=f;
                m2s1:=f;m21:=f;
        end;
  end;
if (mp=4) then
  begin
    if(pn1)and(pna)and(ps1)and (zna)
       and (zs1) and (z2) then up1:=t;
    if up1 then begin km1:=f;kp1:=t;pn1v:=f;end;
    if(pn1)and (pna)and(ps1)and (zna)
       and (zs1) and (z2)and  up1 and kp1 then
          begin zna:=f;zs1:=f;sc2:=t;snv:=f;sc1v:=f;
                m12:=f;m1s1:=f;m1na:=f;m2na:=f;
                m2s1:=f;m22:=f;
          end;
  end;

{отмена маршрута}
if (snv)and(pn1)and(om)and(i=12)and((mp=1)or(mp=2))then
 begin  mp:=0; SNW:=f;  i:=0;
       om:=f;snv:=f;sc2v:=f;sc1v:=f;
       sn:=f;zna:=t;zs1:=t;z1:=t;z2:=t;
       setfillstyle(1,4);M_HIDE;floodfill(40,300,0);
       setfillstyle(1,5);M_HIDE;floodfill(200,120,15);
 end;
if (snv)and(not pn1)and(om)and(i=24)and((mp=1)or(mp=2))
```

```
 then
  begin  mp:=0; SNW:=f;  i:=0;
        om:=f;snv:=f;sc2v:=f;sc1v:=f;sn:=f;
        zna:=t;zs1:=t;z1:=t;z2:=t;
        setfillstyle(1,4);M_HIDE;floodfill(40,300,0);
        setfillstyle(1,5);M_HIDE;floodfill(200,120,15);
  end;
if (pna=f) then sn:=f;
if(sc1v)and(p1)and(om)and(i=12)and(mp=3)then
  begin  mp:=0; Sc1w:=f;i:=0;
        om:=f;snv:=f;sc2v:=f;sc1v:=f;sc1:=f;
        zna:=t;zs1:=t;z1:=t;z2:=t;
        setfillstyle(1,4);M_HIDE;floodfill(211,260,0);
        setfillstyle(1,5);M_HIDE;floodfill(200,120,15);
  end;
if(sc1v)and(not p1)and(om)and(i=24)and(mp=3)then
  begin  mp:=0; Sc1w:=f;i:=0;
        om:=f;snv:=f;sc2v:=f;sc1v:=f;sc1:=f;
        zna:=t;zs1:=t;z1:=t;z2:=t;
        setfillstyle(1,4);M_HIDE;floodfill(211,260,0);
        setfillstyle(1,5);M_HIDE;floodfill(200,120,15);
  end;  if(sc1v)and(not p1)and(om)and(i=24)and(mp=3)then
  begin  mp:=0; Sc1w:=f;i:=0;
        om:=f;snv:=f;sc2v:=f;sc1v:=f;sc1:=f;
        zna:=t;zs1:=t;z1:=t;z2:=t;
        setfillstyle(1,4);M_HIDE;floodfill(211,260,0);
        setfillstyle(1,5);M_HIDE;floodfill(200,120,15);
  end;
if (ps1=f) then sc1:=f;

if(sc2v)and(p2)and(om)and(i=12)and(mp=4)then
  begin  mp:=0; Sc2w:=f;i:=0;
        om:=f;snv:=f;sc2v:=f;sc1v:=f;sc2:=f;
        zna:=t;zs1:=t;z1:=t;z2:=t;
        setfillstyle(1,4);M_HIDE;floodfill(211,230,0);
        setfillstyle(1,5);M_HIDE;floodfill(200,120,15);
  end;
if(sc2v)and(not p2)and(om)and(i=24)and(mp=4)then
  begin  mp:=0; Sc2w:=f;i:=0;
        om:=f;snv:=f;sc2v:=f;sc1v:=f;sc2:=f;
```

```
        zna:=t;zs1:=t;z1:=t;z2:=t;
        setfillstyle(1,4);M_HIDE;floodfill(211,230,0);
        setfillstyle(1,5);M_HIDE;floodfill(200,120,15);
  end;
 if (ps1=f) then sc2:=f;


{отмена набора}
if(onw<on) then
  begin
    if on and(z1 or z2)and
      ((not sn)xor(not sc1)xor(not sc2))then
          begin om:=f;snv:=f;sc2v:=f;sc1v:=f;
               p1v:=f;p2v:=f;pn1v:=f;
               sn:=f;sc1:=f;sc2:=f;
          end;
  end;

  {   Динамическая графика начало  }
{стрелки}
 {+}
    IF kp1>kp1w THEN
      BEGIN kp1w:=kp1;km1:=f;km1w:=f;
           setfillstyle(1,2);floodfill(125,385,0);
           setfillstyle(1,8);floodfill(165,385,0);
           setfillstyle(1,8);floodfill(140,385,0);
           SetColor(8);D_GN;
           SetColor(1);D_GG;
       end;
 {-}
    IF (km1>km1w) THEN
      BEGIN km1w:=km1;kp1:=f;kp1w:=f;
           setfillstyle(1,14);floodfill(165,385,0);
           setfillstyle(1,8); floodfill(125,385,0);
           setfillstyle(1,8);floodfill(140,385,0);
           SetColor(8);D_GG;
           SetColor(1);D_GN;
      end;
 {в}
    IF (km1=f)and(kp1=f)and((km1w=t)or(kp1w=t)) THEN
```

```
        BEGIN km1w:=f;kp1w:=f;
              setfillstyle(1,4); floodfill(140,385,0);
              setfillstyle(1,8); floodfill(160,385,0);
              setfillstyle(1,8); floodfill(120,385,0);
              SetColor(1);D_GG;
              SetColor(1);D_GN;
       END;
{н1п}
 IF pn1>pn1w THEN
    BEGIN pn1w:=pn1;
         setfillstyle(1,1); floodfill(5,275,8);
    end;
 IF pn1<pn1w THEN
    BEGIN pn1w:=pn1;
         setfillstyle(1,4); floodfill(5,275,8);
    end;
{нап}
 IF( pna>pnaw)and(zna) THEN
    BEGIN pnaw:=pna;
         setfillstyle(1,1); floodfill(95,275,8);
    end;
 IF( pna>pnaw)and(not zna) THEN
    BEGIN pnaw:=pna;
         setfillstyle(1,15); floodfill(95,275,8);
    end;
 IF pna<pnaw THEN
    BEGIN pnaw:=pna;
         setfillstyle(1,4);
         floodfill(95,275,8);floodfill(74,274,8);
    end;
{1сп}
 IF (ps1>ps1w)and(zs1)THEN
    BEGIN ps1w:=ps1;
         setfillstyle(1,1);
         floodfill(132,274,8);floodfill(180,245,8);
    end;
 IF (ps1>ps1w)and(not zs1) THEN
    BEGIN ps1w:=ps1;
         setfillstyle(1,15);floodfill(132,274,8);
    end;
```

```
 IF ps1<ps1w THEN
   BEGIN ps1w:=ps1;kp1:=f;km1:=f;km1w:=f;kp1w:=f;
        setfillstyle(1,4); floodfill(132,274,8);
   end;
{1п}
 IF (p1>p1w)and(z1) THEN
   BEGIN p1w:=p1;
        setfillstyle(1,1); floodfill(265,275,8);
   end;
 IF (p1>p1w)and(not z1) THEN
   BEGIN p1w:=p1;
        setfillstyle(1,15); floodfill(265,275,8);
   end;
 IF p1<p1w THEN
   BEGIN p1w:=p1;
        setfillstyle(1,4); floodfill(265,275,8);
   end;
{2п}
 IF (p2>p2w)and(z2) THEN
   BEGIN p2w:=p2;
        setfillstyle(1,1); floodfill(275,245,8);
   end;
 IF (p2>p2w)and(not z2) THEN
   BEGIN p2w:=p2;
        setfillstyle(1,15); floodfill(275,245,8);
   end;
 IF p2<p2w THEN
   BEGIN p2w:=p2;
        setfillstyle(1,4); floodfill(275,245,8);
   end;

{нап}
 IF (zna<>znaw) and(pna) THEN
   begin
    znaw:=zna;
    if zna then
     BEGIN setfillstyle(1,1);floodfill(95,275,8);end
    else
     BEGIN setfillstyle(1,15);floodfill(95,275,8);end;
   end;
```

```
{1сп}
 IF (zs1<>zs1w) and(ps1) THEN
   begin
    zs1w:=zs1;
    if zs1 then
     BEGIN setfillstyle(1,1); floodfill(132,274,8);end
    else
     BEGIN setfillstyle(1,15);floodfill(132,274,8);end;
   end;
{1п}
 IF (z1<>z1w) and(p1) THEN
   begin
    z1w:=z1;
    if z1 then
     BEGIN setfillstyle(1,1); floodfill(265,275,8);end
    else
     BEGIN setfillstyle(1,15);
          m_hide;floodfill(265,275,8);
     end;
   end;
{2п}
 IF (z2<>z2w) and(p2) THEN
   begin
     z2w:=z2;
     if z2 then
      BEGIN setfillstyle(1,1); floodfill(275,245,8);end
     else
      BEGIN setfillstyle(1,15);
           m_hide;floodfill(275,245,8);
      end;
   end;

{светофоры}
 {неч}
IF (C_KMW<>C_KM)AND (SNV)and
   ((not sn)or(not sc1)or(not sc2)) THEN
  BEGIN setfillstyle(1,C_KM);
       M_HIDE; floodfill(40,300,0);
  end;
IF SNW<SN THEN
```

```
  BEGIN SNW:=SN;SNV:=F;SC1V:=f;SC2V:=f;
        setfillstyle(1,2); M_HIDE; floodfill(40,300,0);
  end;
IF SNW>SN THEN
  BEGIN SNW:=SN;SNV:=F;SC1V:=f;SC2V:=f;
        setfillstyle(1,4); M_HIDE; floodfill(40,300,0);
  end;

{ч1}
IF (C_KMW<>C_KM)AND (SC1v)and
   ((not sn)or(not sc2)or(not sc1)) THEN
  BEGIN setfillstyle(1,C_KM);
        M_HIDE; floodfill(215,260,0);
  end;
IF SC1W<SC1 THEN
  BEGIN SC1W:=SC1;Sc1V:=F;SNV:=F;SC2V:=f;
        setfillstyle(1,2); M_HIDE; floodfill(215,260,0);
  end;
IF SC1W>SC1 THEN
  BEGIN SC1W:=SC1;Sc1V:=F;SNV:=F;SC2V:=f;
        setfillstyle(1,4); M_HIDE; floodfill(215,260,0);
  end;
{ч2}
IF (C_KMW<>C_KM)AND (SC2V)and
   ((not sn)or(not sc1)or(not sc2)) THEN
  BEGIN setfillstyle(1,C_KM);
        M_HIDE; floodfill(215,230,0);
  end;
IF SC2W<SC2 THEN
  BEGIN SC2W:=SC2;Sc2V:=F;SNV:=F;SC1V:=f;
        setfillstyle(1,2); M_HIDE; floodfill(215,230,0);
  end;
IF SC2W>SC2 THEN
  BEGIN SC2W:=SC2;Sc2V:=F;SNV:=F;SC1V:=f;
        setfillstyle(1,4); M_HIDE; floodfill(215,230,0);
  end;
{2п миг}
IF (C_SGW<>C_SG)AND (P2V)and p2 THEN
  BEGIN setfillstyle(1,C_SG);
        M_HIDE; floodfill(230,245,8);
```

```
  end;
 {1п миг}
 IF (C_SGW<>C_SG)AND (P1V)and p1 THEN
  BEGIN setfillstyle(1,C_SG);
       M_HIDE; floodfill(230,275,8);
  end;
 {н1п миг}
 IF (C_SGW<>C_SG)AND (PN1V)and pn1 THEN
   BEGIN setfillstyle(1,C_SG);
        M_HIDE; floodfill(1,275,8);
   end;
{отмена набора}
 IF (onw<on)and not(z1 xor z2)then
   BEGIN setfillstyle(1,1);M_HIDE;floodfill(1,275,8);
                           floodfill(230,275,8);
                           floodfill(230,245,8);
        setfillstyle(1,4);M_HIDE;floodfill(40,300,0);
                           floodfill(215,260,0);
                           floodfill(215,230,0);
        setfillstyle(1,5);M_HIDE;floodfill(15,115,15);
   end;
{отмены}
  IF (C_KMW<>C_KM)AND (om) THEN
   BEGIN setfillstyle(1,C_KM);
        M_HIDE; floodfill(15,115,15);
   end;

  onw:=f;on:=f;
  C_KMW:=C_KM;
  C_SGW:=C_SG;
{ Динамическая графика конец }

  M_Show;
  if (Right_Button) then GOTO M2;
  GOTO M1;
  M2:
    CloseGraph;
END.
```

ЛИТЕРАТУРА

1. Казаков А.А., Бубнов В.Д., Казаков Е.А. Станционные устройства автоматики и телемеханики. -М.: Транспорт, 1990.
2. Файсман А. Профессиональное программирование на Турбо Паскале. –Т: 1992.
3. Советов Б.Я. Информационная технология. –М.: Высшая школа, 1994.
4. Гостев В.М. Среда программирования Турбо Паскаль. –К.: 1995.
5. Алиев М.М., Хайдаров А.Х. Дастурли таъминот ва компьютерда масалалар ечиш. –Т.: 2003.
6. Delphi 4, Delphi 5, Delphi 6. Инструкция пользователя. 1998, 1999, 2000 гг.
7. Епанешников А.М., Епанешников В.А. Программирование в среде Турбо Паскаль. –М.: «Диалог-Мифы», 1993.
8. Гостев В.М. Система программирования Турбо Паскаль. – Казань, 1995.

СОДЕРЖАНИЕ