(C) Терентьев С.В.

GRAF16.REL

Описание библиотеки GRAF16.REL
12.12.1997

Файл GRAF16.REL представляет собой перемещаемую библиотеку процедур, по функциональному назначению являющуюся аналогом кишиневских "Драйверов устройств". Библиотека состоит из взаимосвязанных модулей, выполняющих графические, символьные, музыкальные функции. Она является средством эффективной работы на ПК "Вектор-06Ц", составляющие ее процедуры оптимизированы по скорости выполнения и занимаемой длине.

Перемещаемый формат библиотеки позволяет пользователю даже низкой квалификации в минимальные сроки создавать программы профессионального уровня. Библиотека может быть расширена функциями пользователя, перекомпонована, переориентирована на другие функции или использована как составная часть другой библиотеки.

Для использования GRAF16.REL необходима следующая минимальная конфигурация вычислительной системы:

- ПК "Вектор-06Ц";
- Электронный диск 256 Кб и/или НГМД.

Для использования библиотеки необходимо знание пользователем языка ассемблера и умение работать со следующими программами:

- макроассемблер MAS (или M80);
- сборщик перемещаемых модулей L80.

Желательно также уметь работать с LIB80 и CREF80. Далее мы будем полагать, что используются только программы MAS и L80.

Создание программ, использующих перемещаемые библиотеки.

Мы опишем здесь вкратце, какими особенностями обладает программа на макроассемблере, использующая перемещаемую библиотеку, и каким образом осуществляется компоновка исходной программы с библиотекой. Пользователи, знакомые с этими вопросами, могут данную часть описания пропустить.

Сразу оговоримся, что мы будем использовать макроассемблер MAS, чтобы не создавать у начинающих пользователей лишних трудностей с освоением системы управления макроассемблером M80. Командная строка управления компиляцией MAS в точности такая же, как и у макроассемблера MAC, с освоения которого, обычно, начинают программирование на макроассемблере в операционной среде. За подробностями можно обратиться к [1].

Основное отличие MAS от MAC в том, что в результате трансляции исходного текста программы получается не HEX-файл, а, так называемый, REL-файл. Мы не будем касаться его внутренней структуры, скажем только об основных особенностях. Но сначала небольшое отступление.

Пусть пользователь наработал некую библиотеку подпрограмм. При написании очередной программы возникает необходимость включения в нее какой-либо функции из этой библиотеки. Есть два пути.

Первый - взять исходный текст необходимой процедуры и включить его в текст программы. Недостатки очевидны: если эта процедура использует другую процедуру из библиотеки, то вам руками необходимо включить в текст программы и ее, и те, на которые ссылается эта процедура и т.д. Кроме того, использованные в библиотеке метки не могут быть использованы в тексте программы. Таким образом нельзя будет воспользоваться чужой библиотекой, тем более если исходных текстов ее у вас нет.

Второй путь - написать "Драйверы устройств" и подключать эту глыбу к каждой своей программке. При этом в программу будет включена вся библиотека, независимо от того, какая часть ее реально используется. Модифицировать такую библиотеку практически невозможно.

Можно первый путь существенно усовершенствовать. Макроассемблер имеет очень удобную возможность ссылаться на библиотеку макросов. Макросом называется текст, который при трансляции программы будет автоматически считан и подставлен в нужном месте в вызвавшую его программу. То есть, свою библиотеку можно сформировать из макросов, включить их все в один файл, описать этот файл в своей программе, и все... Как все это сделать можно прочитать в [1].

Заметим, что определяя макрос, можно скрыть его внутренние метки от внешних, можно вызвать из текущего другой макрос, то есть все, что нам не хватало. Но, ... Все приобретенные достоинства будут уничтожены тем, что макрос будет включен в программу столько раз, сколько раз к нему обратились.

Есть еще существенный недостаток макробиблиотеки - ее каждый раз необходимо транслировать макроассемблером. Имея мощные библиотеки в десятки килобайт, можно превратить работу в сплошное ожидание результатов трансляции.

От всех перечисленных выше недостатков свободен механизм перемещаемых модулей. Поясним, что это такое.

Результатом работы макроассемблера может быть файл, в который транслируемая программа записывается таким образом, что может быть легко модифицирована без дополнительной трансляции. В частности, программу легко переместить на любую область рабочих адресов. Такой файл, имеющий расширение REL, называется перемещаемым (от английского "relocable").

В результате операции сборки (линковки) с помощью L80, мы получим из перемещаемой программы исполняемый машинный код, который будет располагаться и работать по заданному в процессе сборки адресу. Можно таким образом собрать из нескольких модулей программу не прибегая к объединению их исходных текстов на ассемблере. Процедура сборки происходит достаточно быстро, намного быстрее, чем работает любой ассемблер.

Нам еще нужно выяснить, каким образом модули связываются друг с другом, ведь они могут быть написаны в разное время и разными людьми. Программист при написании очередного модуля должен указать так называемые внешние ссылки. Приведем пример: пусть есть программка-модуль, выводящая на экран спрайт с изображением игрового объекта, и есть модуль, содержащий набор изображений этого объекта. (Например различные фазы движений). До момента сборки адрес расположения обоих модулей остается неизвестным, поэтому программа вывода спрайта, казалось бы, не может быть оттранслирована вообще, потому что в ней нельзя указать адрес, который она использует! Однако, механизм перемещаемых модулей позволяет обойти такие препятствия.

Пусть в модуле, выводящем спрайт, адрес изображения (например младший адрес области данных) записан под именем DATA. Причем, не имеет значения, как и сколько раз это имя используется. Оно останется неопределенным после трансляции макроассемблером и получит какое-то значение только в процессе сборки. Чтобы это имя стало известно линковщику L80 (и макроассемблер не выдал сообщения об ошибке), необходимо в тексте программы вывода спрайта записать:

```
EXTRN   DATA
```

Это имя будет описано в перемещаемом модуле и будет внешним для данного модуля (отсутствующим внутри его). В свою очередь в программе, которая состоит из данных изображения объекта, эта метка должна быть описана так:

```
PUBLIC   DATA
```

В перемещаемом модуле этой программы для линковщика будет помещено сообщение о том, что в этом модуле есть метка DATA и сведения о ней, необходимые для сборки.

При сборке программы линковщик модифицирует модуль, выводящий спрайт, таким образом, что все его обращения по метке DATA будут происходить по адресу, на который попадет модуль с изображением объекта.

Такое имя, описанное в одном модуле как PUBLIC (общедоступный), а в другом EXTRN (внешний), называется глобальным именем. Все остальные имена, использованные в текстах программ, называются локальными и безвозвратно теряются после ассемблирования.

Программа не может быть успешно собрана, если хотя бы одна внешняя ссылка между составляющими ее модулями не удовлетворена. То есть, всем EXTRN-именам должны соответствовать PUBLIC-имена. Количество же PUBLIC-имен может быть больше чем требуется.

Перемещаемые модули могут быть собраны в библиотеку с помощью программы LIB80. Такая библиотека будет обладать замечательными свойствами. При сборке программы линковщиком из библиотеки будут выбраны только один раз и только те модули, которые необходимы для работы собираемой программы. Причем эти библиотеки легко модифицируются, дополняются и т.д.

Имея перемещаемую библиотеку, пользователь может очень быстро создавать качественные программы не затрачивая время на разработку или модификацию под конкретный случай стандартных драйверов и т.п., а всего лишь ссылаясь на необходимую ему процедуру из библиотеки. Причем все процедуры, необходимые для этих процедур и т.д. будут автоматически выбраны из библиотеки и подключены в программу в процессе сборки.

Пользователям, которые хотят подробнее разобраться во всех тонкостях данного механизма можем посоветовать литературу [2]. Тем, кто ничего не понял, приводим краткую инструкцию:

1. В исходном тексте программы на ассемблере не ставить псевдооператор ORG (или записать ORG 0000);

2. Описать необходимые вам подпрограммы из библиотеки с помощью псевдооператора EXTRN. Так, если вы в своей программе рисуете линию и закрашенную окружность, то (лучше в начале) в программе следует записать:

```
EXTRN   @LINE,@CIRCF
```

В программе теперь можно использовать эти имена, например:

```
...
CALL   @LINE
...
CALL   @CIRCF
...
```

Здесь ... обозначает какие-то строки текста на ассемблере;

3. Оттранслировать свою программу MASом по образцу:

```
MAS PROGRAM
```

Здесь PROGRAM - имя файла с расширением ASM, содержащего вашу программу;

4. Собрать свою программу с библиотекой с помощью L80 по образцу:

```
L80
PROGRAM
GRAF16/S
PROGRAM/N/E
```

В результате появится файл PROGRAM.COM, который будет содержать вашу программу и используемую ею часть библиотеки.

Общие особенности процедур, входящих в библиотеку GRAF16.REL

Библиотека графических функций GRAF16 довольно хорошо унифицирована в смысле вызова составляющих ее подпрограмм. Вот ее основные черты:

- все процедуры представляют собой подпрограммы и завершаются командой RET, передача параметров происходит только через регистры и ячейки памяти;
- все имена библиотеки начинаются с символа @, поэтому пользователю рекомендуется не применять в своей программе имена внешних ссылок, начинающихся с этого символа;
- во всех процедурах регистры не сохраняются;
- если процедуре в качестве параметра передается слово данных (адрес, координаты точки и т.п.), то его помещают в регистровую пару DE. Причем в D помещается горизонтальная координата, в E - вертикальная;
- если процедуре в качестве параметра передается байт данных, то он, как правило, помещается в аккумулятор;
- процедурами, как правило, возвращаются значения в регистрах HL и аккумуляторе.

Обслуживание с помощью библиотеки прерываний ЦП по кадровому сихроимпульсу.

Этот раздел необходимо обязательно прочесть и разобраться в нем, особенно если вы только осваиваете программирование на ассемблере.

"Вектор-06Ц" устроен так, что каждые 20 миллисекунд, именно в тот момент, когда телевизор (или монитор), погасив электронный луч, переводит его с низа на верх экрана, на центральный процессор (ЦП) подается сигнал запроса на прерывание. Если прерывания при этом разрешены, то происходит следующее. Вместо очередного кода программы из ОЗУ компьютера процессору подают код 0FFH, соответствующий команде RST 7. Получив эту команду, процессор ее выполняет, т.е. выбивает в стек адрес, по которому он должен был извлечь команду, и загружает в программный счетчик адрес 00038H. После этого ЦП будет выполнять ту программу, которая начинается с этого адреса. Таким образом происходит "прерывание" основной программы для выполнения программы, начинающейся с адреса 00038H и обслуживающей данное прерывание. Вернуться в основную программу можно выполнив команду RET. (Естественно, должны быть сохранены все регистры ЦП).

К особенностям "Вектора" относится то, что опрос клавиатуры и установка цветовой палитры должны, вообще говоря, производиться именно в момент, когда луч на экране телевизора погашен, т.е. в момент поступления прерывания на ЦП. Поэтому с адреса 00038H обычно начинается подпрограмма установки палитры и опрос клавиатуры.

Библиотека GRAF16 позволяет пользователю включить в программу обслуживания прерывания до 8 любых драйверов, не считая драйвера установки палитры. Это могут быть и опрос клавиатуры, и просто опрос курсорных клавиш, и драйвер музыкального сопровождения (хоть для КР580ВИ53, хоть для AY-3-8910), и часы, и таймер, и бегущая строка, и т.д. и т.п. Причем, этих драйверов может быть от 0 до 8, и их число может меняться по ходу программы. Естественно, что все они должны завершить работу до поступления следующего прерывания. Поэтому начинающим программистам не рекомендуется браться за самостоятельное написание подобных драйверов.

Процедура библиотеки, вызывающая драйверы обработки прерывания, написана так, чтобы сохранить предельно высокую скорость работы. Так, если изменяется цветовая палитра, то нужный для этого драйвер (находящийся на "особом" положении), будет работать только один раз - в последующие прерывания ему управление не передается. Даже такая "мелочь" экономит более 5% времени ЦП по сравнению с

кишиневскими "Драйверами устройств". Другие драйверы для обработки прерывания можно "подключать" и "отключать" с помощью библиотечных процедур @DRVON и @DRVOF. Если ни один драйвер не задействован, то после установки палитры по адресу 00038H помещается команда RET. В теле самих драйверов сохранения регистров НЕ ТРЕБУЕТСЯ - они уже сохранены в обслуживающей их программе. При первой же возможности ненужные драйверы следует отключать, так как им передается управление довольно часто (50 раз в секунду), и это может замедляет выполнение основной программы.

Отметим пару драйверов, включенных в библиотеку, и работающих по прерываниям. Наиболее важный - это опрос клавиатуры (@KBD). Если вы хотите использовать процедуры @CONIN, @INKEY, @CONST, то прежде необходимо задействовать драйвер @KBD. Также полезен драйвер музыкального сопровождения. Но его подключать следует не напрямую, а вызвав процедуру @MUZON. Их подробное описание, а также о других драйверах читай в следующем разделе.

Далее следует более подробное описание составляющих библиотеку процедур.

Драйверы общего назначения

@INIT подпрограмма инициализации.
Входные: DE - адрес установки стека
Выходные: нет
Действие: процедура "прячет" на одэкранное ОЗУ операционную систему, устанавливает указатель стека на заданный регистровой парой DE адрес, записывает необходимые команды переходов по адресам 0000 и 0038. В данной версии библиотеки на подэкранном ОЗУ сохраняется только МикроДОС версии Т-34 (или другая ДОС, если она занимает в ОЗУ места не больше 16 Кб). Любая другая ДОС уничтожается и возврат в нее через @EXIT невозможен, хотя в остальном программа останется полностью работоспособной (конечно, если она не использует функции ДОС).

@EXIT подпрограмма подготовки выхода в ДОС.
Входные: нет
Выходные: нет
Действие: если ДОС была сохранена на подэкранном ОЗУ (процедурой @INIT), то она разворачивается по рабочим адресам. По адресу 0000, 0005, 0038 восстанавливаются необходимые команды переходов, а также устанавливается цветовая палитра, использованная в ДОС до начала работы библиотеки. После выполнения этой подпрограммы можно передавать управление ДОС командой JMP 0000 или продолжить работу в самой ДОС уже без использования процедур библиотеки.
Если ДОС не сохранялась, данная процедура не производит никаких действий, а команда JMP 0000 приводит к перезапуску программы.

@IDOS подпрограмма подготовки обращения к ДОС.
Входные: нет
Выходные: нет
Действие: если ДОС (напомним, что на сегодня только Т-34) была сохранена на подэкранном ОЗУ процедурой @INIT, то она разворачивается по рабочим адресам, подэкранное ОЗУ включается. После вызова этой процедуры можно обращаться к функциям ДОС (через CALL 0005).

Отличие процедуры @IDOS от @EXIT состоит в том, что опрос клавиатуры (не забудьте, что должна была быть исполнена процедура подключения драйвера @KBD, см. описание), установка палитры и вывод текстовых сообщений на экран во время работы ДОС будут вестись через процедуры библиотеки.

После вызова данной функции обмениваются местами часть свернутой ДОС с подэкранного ОЗУ (A000-BFFF) и данные из экранной области E000-FFFF. Поэтому

нельзя ничего выводить на экран в область A000-FFFF средствами библиотеки после вызова данной процедуры и до вызова процедуры @EDOS. Если пользователь ничего не меняет в области A000-BFFF при работе с ДОС, то после вызова процедуры @EDOS изображение на экране полностью восстанавливается.

@EDOS подпрограмма завершения обращения к ДОС.
Входные: нет
Выходные: нет
Действие: процедура отменяет подмену драйверов ДОС драйверами библиотеки и обменивает часть ДОС, находящуюся в области адресов E000-FFFF, с данными на подэкранном ОЗУ. Затем подэкранное ОЗУ отключается.

Несколько замечаний:

- повторное сохранение ДОС необходимо для нормальной ее работы после очередного "разворачивания", так как могут измениться параметры системы, содержимое дисковых буферов и т.п.

- если во время работы с ДОС не было обращений в область подэкранного ОЗУ с адресами A000-BFFF, изображение на экране, бывшее там до вызова @IDOS, полностью восстанавливается.

@MOVI пересылка блока.
Входные: BC <--(+)-- DE; HL
Выходные: нет
Действие: пересылка блока данных, длиной, заданной в HL. В регистрах BC задан МЛАДШИЙ адрес области, куда производится пересылка. В DE задан МЛАДШИЙ адрес, откуда производится пересылка. Пересылка для увеличения скорости осуществляется с помощью стека, поэтому в начале процедуры происходит запрещение прерываний. После окончания пересылки программист должен сам решать, разрешить ли прерывания. Процедура пересылает данные длиной ТОЛЬКО КРАТНОЙ 8 байтам. Если содержимое HL не кратно этому числу - происходит разрушение всей программы. Процедуру рекомендуется применять только для больших массивов данных.

@MOVD пересылка блока.
Входные: BC <--(-)-- DE; HL
Выходные: нет
Действие: пересылка блока данных длиной, заданной в HL. В регистрах BC задан СТАРШИЙ адрес области, куда производится пересылка. В DE задан СТАРШИЙ адрес, откуда производится пересылка. Пересылка для увеличения скорости осуществляется с помощью стека, поэтому в начале процедуры происходит запрещение прерываний. После окончания пересылки программист должен сам решать, разрешить ли прерывания. Процедура пересылает данные длиной ТОЛЬКО КРАТНОЙ 8 байтам. Если содержимое HL не кратно этому числу - происходит разрушение всей программы. Процедуру рекомендуется применять только для больших массивов данных.

@DRVON включение драйвера в процедуру обработки прерывания
Входные: DE - адрес драйвера, A - номер в очереди
Выходные: нет
Действие: процедура модифицирует программу обработки прерываний, включая в нее вызов драйвера, адрес которого передается в регистрах DE. Теперь при каждом прерывании управление будет передаваться по этому адресу. Сам драйвер должен быть написан как подпрограмма, сохранения регистров ЦП не требуется. С помощью @DRVON можно включить в обработку прерываний до 8 таких драйверов. (При старте их число равно нулю). Они все выполняются за одно прерывание в порядке очереди. Число задействованных драйверов хранится в ячейке @DRVN. При вызове процедуры в аккумуляторе передается номер (начиная с 0), каким по счету среди остальных этот драйвер будет помещен в очередь. Рекомендуется драйверы опроса клавиатуры помещать

как можно раньше, другого типа - в конец списка. Значение аккумулятора большее, чем текущее число задействованных драйверов воспринимается как команда поместить данный драйвер в конец очереди.

@DRVOF исключение драйвера из процедуры обработки прерываний
Входные: DE - адрес драйвера
Выходные: нет
Действие: процедура ищет в списке драйверов, обрабатывающих прерывание, драйвер с адресом, заданным в DE, и исключает его из этого списка. Если такой не найден - никаких изменений в очередь драйверов, обрабатывающих прерывание, не вносится.

@KBD драйвер опроса клавиатуры.
Входные: нет
Выходные: нет
Действие: данная процедура должна быть включена в набор драйверов, обслуживающих прерывание ЦП по кадровому синхроимпульсу. Ее подключение производится при помощи процедуры @DRVON следующим образом (см. также описание процедур @DRVON, @DRVOF):

```
...
XRA    A
LXI    D,@KBD
CALL   @DRVON
...
```

Описанная процедура необходима только если в программе используется ввод с клавиатуры с помощью процедур @INKEY, @CONIN, @CONST. Перечисленные процедуры обеспечивают ввод с клавиатуры символов в коде КОИ-8 с буферированием ввода до 8 символов. Переключение регистров клавиатуры производится как в ДОС Т-34.

@SON включение звука нажатия клавиш
Входные: нет
Выходные: нет
Действие: после выполнения данной процедуры (если драйвер @KBD обслуживает прерывания) нажатие клавиш клавиатуры будет сопровождаться звуковым сигналом.

@SOFF выключение звука нажатия клавиш
Входные: нет
Выходные: нет
Действие: обратное предыдущей процедуре. Опрос клавиш драйвером @KBD будет беззвучным.

@PLTR подпрограмма установки цветовой палитры
Входные: DE - адрес таблицы цветов (16 байт)
Выходные: нет
Действие: процедура изменяет цветовую палитру. В регистрах DE должен быть передан младший адрес таблицы из 16 устанавливаемых цветов. Порядок расположения цветов в таблице следующий: цвет0, цвет1, цвет2, ... , цвет14, цвет15

Процедура запоминает таблицу во внутреннем буфере и подключает в программу обработки прерываний программу занесения этой палитры в цветовое ОЗУ компьютера. Следует помнить, что эта палитра установится в момент обработки первого по счету прерывания, поступившего после возврата из @PLTR (Подпрограмма разрешает прерывания).

По умолчанию установлена следующая палитра:

Номер цвета:	Код (восьмиричный)	Цвет
00	100	темно-синий
01	200	синий
02	020	зеленый
03	320	голубой
04	006	красный
05	206	малиновый
06	026	кирпичный
07	066	желтый
08	000	черный
09	305	фиолетовый
0A	042	ярко-зеленый
0B	300	ярко-синий
0C	002	темно-красный
0D	230	бирюзовый
0E	122	серый
0F	255	белый

@PLTRS подпрограмма плавной смены палитры
Входные: DE - адрес таблицы цветов (16 байт)
Выходные: нет
Действие: процедура изменяет цветовую палитру. В регистрах DE должен быть передан младший адрес таблицы из 16 устанавливаемых цветов. Порядок расположения цветов в таблице следующий: цвет0, цвет1, цвет2, ... , цвет14, цвет15

Процедура в течение 24 прерываний (около 0,5 сек) изменяет палитру плавно переходя от установленной до вызова процедуры до заданной регистрами DE. Возврат из процедуры происходит по истечении этого времени. Передаваемая процедуре таблица цветов не изменяется.

@CONIN подпрограмма ввода символа с клавиатуры
Входные: нет
Выходные: A - код введенного символа
Действие: процедура ожидает ввода какого-либо символа с клавиатуры и возвращает его в аккумуляторе (Драйвер опроса клавиатуры имеет 8-символьный буфер для хранения введенных символов).

@INKEY подпрограмма ввода кода нажатой клавиши
Входные: нет
Выходные: A - код символа, FF - если не нажата
Действие: не ожидая нажатия клавиши, процедура возвращает либо код нажатой клавиши, либо FF.

@CONST подпрограмма опроса статуса клавиатуры
Входные: нет
Выходные: A = 00 - клавиша нажата, A = FF - клавиша не нажата.
Действие: процедура возвращает состояние клавиатуры - нажималась ли какая-либо клавиша.

@SCROL установка указателя скроллинга
Входные: A - значение указателя
Выходные: нет
Действие: указатель скроллинга является номером строки, которая отображается верхней на экране. Если пользователю нужно получить его значение, то можно извлечь байт данных по метке @SCRL.

@BORD установка цвета бордюра (и выбора режима)
Входные: A - значение цвета
Выходные: нет
Действие: устанавливается цвет бордюра, задаваемый младшими 4 битами аккумулятора. Если младший бит старшей тетрады аккумулятора установлен в 1 - включается режим экрана 512x256 точек. Цвет устанавливается немедленно по вызову процедуры, поэтому ее можно использовать для отображения каких-либо "быстрых" процессов, занимающих по времени меньше 20 мсек., т.е. времени смены экрана. Если пользователю нужно получить значение цвета бордюра, то можно извлечь байт данных по метке @BRDC.

Драйверы графического (точечного) отображения

@GCOL установка цвета отображения
Входные: A - номер цвета
Выходные: нет
Действие: устанавливается цвет отображения графической информации. Если в байте 4 старших бита (старшая тетрада), равны 0, то номер цвета задается младшими 4 битами (16 цветов). (Если бит установлен в 1, то в соответствующей экранной плоскости во время вывода графической информации биты, соответствующие устанавливаемым точкам, будут установлены в 1. Если бит цвета установлен в 0 - сброшены в 0.)

Если в старшей тетраде байта цвета какой-то бит установлен в 1, то независимо от состояния бита в младшей тетраде, соответствующего той же экранной плоскости, в этой плоскости будет производиться инверсия изображения. (В режиме инверсии повторный вывод того же объекта в то же место эквивалентен восстановлению исходного состояния экрана до вывода этого объекта).

Комбинируя режимы установки/сброса экранной информации и ее инверсии можно получить огромное количество различных цветовых эффектов.

@MASC установка маски отображения
Входные: DE - адрес массива данных (32 байта)
Выходные: нет
Действие: подпрограмма устанавливает цветную маску вывода графической информации. Маска задается для квадрата 8x8 точек, на которые разбивается весь экран. При установке точки какой-либо графической процедурой, использующей вывод через маску, соответствующие этой точке биты на экранных плоскостях принимают значения, заданные этой маской.

```
    H3 H3 H3 H3 H3 H3 H3 H3          +
     H2 H2 H2 H2 H2 H2 H2 H2          !  Маска верхних 8
      H1 H1 H1 H1 H1 H1 H1 H1         !  точек квадрата
       H0 H0 H0 H0 H0 H0 H0 H0         +

    !  !  !  !  !  !  !  !           - плоскость 3 (8000-9FFF)
     !  !  !  !  !  !  !  !           - плоскость 2 (A000-BFFF)
      !  !  !  !  !  !  !  !           - плоскость 1 (C000-DFFF)
       !  !  !  !  !  !  !  !           - плоскость 0 (E000-FFFF)

    A3 A3 A3 A3 A3 A3 A3 A3          +
     A2 A2 A2 A2 A2 A2 A2 A2          !  Маска нижних 8
      A1 A1 A1 A1 A1 A1 A1 A1         !  точек квадрата
       A0 A0 A0 A0 A0 A0 A0 A0         +
  Номера
  битов: 7  6  5  4  3  2  1  0
```

Так, предположим, что устанавливается точка, соответствующая биту номер 3 нижнего байта квадрата. При этом в соответствующие экранные плоскости будут помещены биты номер 3 байтов A0, A1, A2, A3. Их комбинация и определит цвет данной точки.

При вызове процедуры в регистры DE помещают младший адрес массива из 32 байт, где байты маски расположены в такой последовательности:
A0, A1, A2, A3, B0, B1, ... G2, G3, H0, H1, H2, H3

@SCRN запрещение/разрешение обращения к плоскостям экрана
Входные: A - код обращения к экранным плоскостям
Выходные: нет
Действие: младшие 4 бита аккумулятора задают возможность обращения графических процедур к плоскостям экрана

```
    X X X X X X X X
            ! ! ! +-- плоскость 0 (E000-FFFF)
            ! ! +---- плоскость 1 (C000-DFFF)
            ! +------ плоскость 2 (A000-BFFF)
            +-------- плоскость 3 (8000-9FFF)
```

Причем нулевое значение соответствующего бита запрещает обращение к плоскости, единичное значение - разрешает. По умолчанию установлено обращение ко всем 4 плоскостям.

@CLS подпрограмма очистки экрана
Входные: нет
Выходные: нет
Действие: разрешенные к обращению экранные плоскости заполняются нулями, указатель скроллинга устанавливается равным FF.

@PLOTS установка графического курсора без отображения точки
Входные: DE - гор., вер. координаты курсора
Выходные: нет
Действие: указатель графического курсора устанавливается на точку с заданными координатами. Координаты на экране отсчитываются от левого нижнего угла и принимают значения от 00 до FF.

@PGET определение цвета точки
Входные: DE - гор., вер. координаты
Выходные: A - математический цвет точки
Действие: подпрограмма возвращает цвет точки, координаты которой заданы в регистрах DE. Положение графического курсора при этом не изменяется.

@PLOT установка точки
Входные: DE - гор., вер. координаты
Выходные: нет
Действие: в точку с указанными координатами помещается указатель графического курсора и рисуется точка текущим цветом отображения.

@PLOTM установка точки через маску
Входные: DE - гор., вер. координаты
Выходные: нет
Действие: в точку с указанными координатами помещается указатель графического курсора и рисуется точка цветом, зависящим от цветовой маски вывода (см. описание процедуры @MASC).

@LINE рисование линии
Входные: DE - гор., вер. координаты
Выходные: нет
Действие: из точки, где находился графический курсор, текущим цветом отображения проводится линия в точку, заданную регистрами DE. Указатель графического курсора устанавливается в точку, заданную в DE.

@LINEV рисование вертикальной линии
Входные: E - вертикальная координата
Выходные: нет
Действие: из точки, где находился графический курсор, текущим цветом отображения проводится вертикальная линия до точки с той же горизонтальной координатой и с заданной регистром E вертикальной. Указатель графического курсора устанавливается в последнюю точку.

@LINEH рисование горизонтальной линии
Входные: D - горизонтальная координата
Выходные: нет
Действие: из точки, где находился графический курсор, текущим цветом отображения проводится горизонтальная линия до точки с той же вертикальной координатой и с заданной регистром D горизонтальной. Указатель графического курсора устанавливается в последнюю точку.

@BOX рисование прямоугольника
Входные: DE - гор., вер. координаты
Выходные: нет
Действие: один угол прямоугольника задается положением графического курсора, противоположный - регистрами DE. Графический курсор устанавливается в точку, заданную в DE.

@BOXF рисование закрашенного прямоугольника
Входные: DE - гор., вер. координаты
Выходные: нет

Действие: один угол прямоугольника задается положением графического курсора, противоположный - регистрами DE. Графический курсор устанавливается в точку, заданную в DE.

@BOXFM рисование закрашенного через маску прямоугольника
Входные: DE - гор., вер. координаты
Выходные: нет
Действие: процедура рисует закрашенный через маску прямоугольник, один угол которого задается положением графического курсора, противоположный - регистрами DE. Графический курсор устанавливается в точку, заданную в DE.

@CIRC подпрограмма рисования окружности
Входные: DE - гор., вер. координаты центра, A - радиус
Выходные: нет
Действие: графический курсор устанавливается в точку с координатами DE, рисуется окружность с центром в этой точке и радиусом, заданным в A.

@CIRCF рисование закрашенной окружности
Входные: DE - гор., вер. координаты центра, A - радиус
Выходные: нет
Действие: графический курсор устанавливается в точку с координатами DE, рисуется закрашенная текущим цветом графического отображения окружность с центром в этой точке и радиусом A.

@FILL закраска произвольной области
Входные: DE - гор., вер. координаты начала
Выходные: нет
Действие: графический курсор устанавливается в точку с координатами DE, текущим цветом графического отображения производится закраска области, границы которой определяются точками любого цвета, отличного от того, на который попала начальная точка.

@STEP пересчет относительных координат
Входные: DE - гор., вер. смещение
Выходные: DE - гор., вер. координаты
Действие: подпрограмма вычисляет координаты графического курсора по заданным смещениям по вертикали и горизонтали. Смещение интерпретируется как байт данных со знаком (диапазон значений -128 ... +127). В случае, если в результате смещения курсор должен попасть за пределы экрана, соответствующая координата устанавливается равной координате границы.

@PCUR пересчет координат графического курсора
Входные: DE - гор., вер. координаты
Выходные: HL - адрес байта, A - маска
Действие: подпрограмма получения адреса байта, соответствующего координатам графического курсора, и байта, один бит которого, соответствующий этой точке, установлен в 1. Адрес байта соответствует нулевой экранной плоскости (E000-FFFF).

Символьные драйверы.

@PRINT вывод символа на экран
Входные: C - код символа
Выходные: нет

Действие: выводит на консоль символ, заданный содержимым регистра C. Вывод осуществляется в восьмибитовом коде. Библиотека поставляется вместе с модулем знакогенератора КОИ-8, совпадающим с соответствующей кодировкой ДОС Т-34 (см. [3]). Модуль знакогенератора подключают в процессе сборки программы. Можно подключить любой другой знакогенератор со знакоместом 8x8 точек. Он должен иметь в начале массива данных глобальную метку @ZG.

Управление положением курсора на экране осуществляется с помощью управляющей последовательности:

1BH, положение по вертикали, положение по горизонтали. Где положения по вертикали и горизонтали отсчитываются от левого верхнего угла и принимают значения 00-1F.

Остальные управляющие символы:

08H - курсор влево;
09H - табуляция;
0AH - перевод строки;
0CH - курсор в угол;
0DH - возврат каретки;
18H - курсор вправо;
19H - курсор вверх;
1AH - курсор вниз;
1FH - очистка экрана.

@WRITE вывод на экран сообщения
Входные: DE - адрес сообщения
Выходные: нет
Действие: подпрограмма выводит на экран сообщение, адрес которого передается в регистровой паре DE. Сообщение должно заканчиваться нулевым байтом.

@HEX печать байта в шестнадцатиричном виде
Входные: A - выводимый байт
Выходные: нет
Действие: подпрограмма выводит на экран в шестнадцатиричном виде содержимое аккумулятора. Гашение незначащих нулей не производится.

@DECA печать байта в десятичном виде
Входные: A - выводимый байт
Выходные: нет
Действие: подпрограмма выводит на экран в десятичном виде содержимое аккумулятора. Гашение незначащих нулей не производится.

@DECDE печать слова в десятичном виде
Входные: DE - выводимое слово
Выходные: нет
Действие: подпрограмма выводит на экран в десятичном виде содержимое регистровой пары DE с гашением незначащих нулей.

@SCOL установка цвета символьного отображения
Входные: A - цвет отображения
Выходные: нет
Действие: младшая тетрада байта, помещенного в аккумулятор, определяет цвет символов, старшая - подложки. Если цвета символов и подложки совпадают - устанавливается вывод символов через инверсию. По умолчанию установлен цвет 0FH.

@SCRNS установка обращения к плоскостям экрана симв. процедур
Входные: A - код обращения к экранным плоскостям
Выходные: нет
Действие: аналогично процедуре @SCRN, но @SCRNS влияет на работу только символьных процедур (исключая @BIG, котрорая использует драйвер точечного вывода). Младшие 4 бита аккумулятора, устанавливаемые при вызове @SCRNS задают возможность обращения символьных процедур к плоскостям экрана

```
X X X X X X X X
        ! ! ! +-- плоскость 0 (E000-FFFF)
        ! ! +---- плоскость 1 (C000-DFFF)
        ! +------ плоскость 2 (A000-BFFF)
        +-------- плоскость 3 (8000-9FFF)
```

Причем, нулевое значение соответствующего бита запрещает обращение к плоскости, единичное значение - разрешает. По умолчанию установлено обращение ко всем 4 плоскостям.

Процедура @SCRNS введена в дополнение к @SCRN, чтобы существенно сократить число модулей, подключаемых при сборке программы, если в ней используется процедура @SCRN, но не используются символьные драйверы.

@BIG вывод деформированных символов увеличенного размера
Входные: C - код выводимого символа
Выходные: нет
Действие: подпрограмма выводит символ, увеличенный по вертикали и горизонтали в целое число раз, причем направление горизонтали и вертикали символа на экране может быть произвольным (См. описание процедур @SBIGM, @SBIGH, @SBIGV).

@SBIGM установка масштабов увеличения символов
Входные: DE - масштаб по горизонтали, вертикали
Выходные: нет
Действие: подпрограмма устанавливает масштаб увеличения для символов, выводимых процедурой @BIG. Горизонтальный масштаб передается в регистре D, вертикальный - E. Если содержимое регистра равно нулю, соответствующий масштаб остается таким, каким он был установлен предыдущим вызовом данной процедуры.

@SBIGH установка направления рисования горизонтали символов
Входные: DE - гор., вер. координаты точки направления
Выходные: нет
Действие: подпрограмма устанавливает режим деформации символов, выводимых на экран процедурой @BIG. Устанавливается направление рисования горизонтали символов. Оно задается содержимым регистров DE, в которых процедуре передается горизонтальная и вертикальная координаты некой точки. Направление из точки с координатами (80H, 80H) на эту точку и будет направлением рисования горизонтали символов.

Положение графического и символьного курсоров процедурой не изменяется. Если содержимое регистра D или E равно 80H (неопределенное направление по горизонтали или вертикали), то соответствующая координата точки, задающей направление рисования, останется прежней.

@SBIGV установка направления рисования вертикали символов
Входные: DE - гор., вер. координаты точки направления
Выходные: нет

Действие: подпрограмма устанавливает режим деформации символов, выводимых на экран процедурой @BIG. Устанавливается направление рисования вертикали символов. Оно задается содержимым регистров DE, в которых процедуре передается горизонтальная и вертикальная координаты некой точки. Направление из точки с координатами (80H, 80H) на эту точку и будет направлением рисования вертикали символов.

Положение графического и символьного курсоров процедурой не изменяется. Если содержимое регистра D или E равно 80H (неопределенное направление по горизонтали или вертикали), то соответствующая координата точки, задающей направление рисования, останется прежней.

Другие драйверы.

@RND1,**@RND2**,**@RND3**,**@RND4** датчики случайных чисел
Входные: нет
Выходные: HL, A - случайное число
Действие: подпрограммы вырабатывают псевдослучайные числа. В регистрах HL возвращается число в диапазоне 0000-FFFF. В аккумуляторе возвращается вырабатываемое из него число в диапазоне 00-FF.

@SCUR установка изображения символьного курсора
Входные: DE - адрес блока данных
Выходные: нет
Действие: подпрограмма устанавливает вид изображения символьного курсора, заданного массивом данных, адрес которого передается в регистрах DE. Массив имеет 8 байт длины, установленный в 1 бит в массиве разрешает мигание соответствующей точки в знакоместе, 0 – запрещает. Расположение байтов массива в знакоместе показано на рис:

```
верхний байт:      H7 H6 H5 H4 H3 H2 H1 H0      - старший байт массива

                   .  .  .  .  .  .  .  .              .  .  .

нижний байт:       A7 A6 A5 A4 A3 A2 A1 A0      - младший байт массива

номер бита:        7  6  5  4  3  2  1  0
```

@MUZON включение музыкального сопровождения на AY-3-8910
Входные: DE - адрес массива данных
Выходные: нет
Действие: процедура подготавливает и подключает к драйверам обработки прерывания драйвер музыкального сопровождения. Драйвер обслуживает музыкальный контроллер на м/с AY-3-8910, адресуемый через 14H и 15H порты. В DE передается адрес начала массива данных, записанных в стандарте STM.

@MUZOF выключение музыкального сопровождения на AY-3-8910
Входные: нет
Выходные: нет
Действие: процедура отключает драйвер музыкального сопровождения и настраивает музыкальный контроллер AY-3-8910 на молчание.

@DESPR вывод на экран изображения в формате SPR.
Входные: DE - адрес последнего байта массива данных

Выходные: нет
Действие: процедура распаковывает и выводит на экран изображение, записанное в SPR-формате. Структура SPR-файла следующая:

```
       10H байт - палитра
       2 байта  - 00
       N байт - массив упакованных данных
       M байт - 00
```

Паковка производится по повторяющимся байтам последовательно от младших к старшим адресам экранной области (в результате распаковки должно получаться 32 Кб данных). Пакованные данные состоят из набора записей следующих двух видов:

```
       Описание повторяющихся данных
       1 байт - байт, который нужно повторить
       1 байт - 0 x x x x x x x
                  \-----v-----/
                        число повторений

       Описание неповторяющихся данных
       N байт - данные
       1 байт - 1 x x x x x x x
                  \-----v-----/
                        N (длина строки данных)
```

Поскольку байт, описывающий очередную запись, находится в конце самой записи, данные распаковываются от старших адресов к младшим. Завершение одной записи означает начало другой (между записями никаких дополнительных разделителей нет). Такой способ распаковки от старших адресов к младшим позволяет размещать пакованные данные в начале экранной плоскости без риска их уничтожения при последующей распаковке. Процедура производит распаковку данных без установки палитры. Установка палитры остается за пользователем.

Назначение некоторых ячеек памяти
/ содержимое ячеек не должно изменяться программой пользователя /

@ISDOS	байт	- если 00 - ДОС на подэкранном ОЗУ не сохранялась, если любое другое число - сохранялась;
@SCRL	байт	- указатель скроллинга;
@BRDC	байт	- цвет бордюра;
@SCRW	байт	- код обращения к экранным плоскостям точечных процедур;
@SCRWS	байт	- код обращения к экранным плоскостям симв. процедур;
@GRCOL	байт	- текущий цвет графического отображения;
@GRCUR	слово	- позиция графического курсора;
@SMCOL	байт	- текущий цвет символьного отображения;
@CURV	байт	- вертикальная позиция символьного курсора;
@CURH	байт	- горизонтальная позиция символьного курсора;
@DRVN	байт	- число драйверов, обрабатывающих прерывание

Литература.

1. Сервисные программы МикроДОС ПК "Вектор-06Ц" / Автор-сост. Терентьев С.В., 1993. - 52 с. (Б-чка "Вектор". Вып. 2.)
2. Григорьев В.Л. Программное обеспечение микропроцессорных систем. - М.: Энергоатомиздат, 1983.
3. Фигурнов В.Э. Русские буквы на вашем компьютере. Компьютер N1, стр. 16, (1990)