Лабораторная работа №1. Введение в программирование на языке ассемблера

Введение

Ассемблерами называют разновидность компиляторов, осуществляющих трансляцию программ со специального машинно-зависимого языка программирования низкого уровня (называемого языком ассемблера) в объектный код. На языке ассемблера в удобной для восприятия человеком форме с использованием специальных обозначений (мнемоник и директив) описывается непосредственно последовательность машинных команд программы и побайтовое распределение памяти, отводимой под данные.

В данном цикле лабораторных работ используется ассемблер Turbo Assembler (TASM) фирмы Borland для процессоров семейства Intel 80x86.

В первой лабораторной работе необходимо ознакомиться с типовым процессом написания программ на языке ассемблера и разработать простейшую программу, использующую одну из возможностей DOS или BIOS по вводу/выводу текстовой информации.

1. Процесс разработки программы

В простейшем случае процесс написания программы на языке ассемблера выглядит следующим образом.

1) В любом текстовом редакторе подготавливается исходный текст программы на языке ассемблера, например, файл MY.ASM. (Это должен быть именно текст, а не, например, документ в формате Word или RTF!)

2) Исходный текст компилируется (ассемблируется) с целью проверки синтаксиса и получения объектного кода. В командной строке следует набрать

```
tasm my.asm
```

на выходе `my.obj` или сообщения об ошибках

Сообщения об ошибках выводятся с указанием номера строки исходного текста, где ошибка имеет место.

3) Из одного или нескольких объектных файлов с использованием компоновщика (TLINK) создается исполнимый файл. На этом этапе также возможно появление сообщений об ошибках, которые могут быть выявлены только на этапе компоновки.

```
tlink my.obj
```

на выходе `my.exe` или сообщения об ошибках

4) Полученные исполнимый файл запускается и тестируется на предмет корректности выполнения заданных действий.

5) В случае необходимости пошаговой отладки программы исполнимый файл загружается в отладчик кодов. Удобно воспользоваться отладчиком Turbo Debugger той же фирмы, так как он при использовании некоторых опций компиляции позволяет производить отладку, отображая исходный код программы, а не только результат ее дизассемблирования.

```
td my.exe
```

2. Шаблон программы

Исходный текст программы в рамках данной и большинства последующих лабораторных работ может быть построен в соответствии со следующим шаблоном. Некоторые теоретические сведения приведены ниже.

```
; комментарий с указанием:
; - фамилии и группы студента
; - варианта
; - краткой формулировки задания
.model small      ; один сегмент кода, данных и стека
.stack 100h       ; отвести под стек 256 байт
.data             ; начало сегмента данных
S     db    'Hello, world!$'
.code             ; начало сегмента кода
      ; Начальная инициализация
      mov   ax,@data
      mov   ds,ax       ; настройка DS на начало сегмента данных

      ;-------------------------------------------------
      ; Здесь – код в соответствии с заданием, например
      ; Вывод строки на экран
      mov   ah,9        ; номер функции DOS
      mov   dx,offset S ; DS:DX <- адрес строки S
                        ; DS уже проинициализирован ранее
      int   21h         ; Вывод строки на экран в текущей позиции курсора
      ;-------------------------------------------------

      ; Стандартное завершение программы
      mov   ax,4C00h    ; ah = N функции, al = код возврата
      int   21h         ; снять программу с выполнения

      end         ; конец текста программы
```

3. Сведения о процессоре

3.1. Сегментированная модель памяти

Процессор 8086 имеет 20-разрядную шину адреса и может адресовать 1 Мб памяти. Однако регистры процессора 16-разрядные. Поэтому полный 20-разрядный адрес всегда формируется из двух 16-разрядных компонент – сегмента и смещения. В литературе их часто записывают через символ двоеточия: A000:0000h.

Для вычисления 20-разрядного адреса сегмент умножается на 16 (сдвигается влево на 4 двоичных разряда, что эквивалентно приписыванию одного нуля справа в шестнадцатеричной записи) и складывается со смещением:

1234:5678h (1234h<<4) + 5678h = 12340h + 5678h = 179B8h

Формально, сегмент – это номер 16-байтового параграфа (их в мегабайте как раз 64к), относительно начала которого путем отсчета указанного смещения получается адрес конкретного байта.

Очевидно, что с использованием одного сегментного адреса в качестве начала отсчета можно получить 64к различных адресов (столько возможно указать смещений). Очевидно также, что один и тот же 20-разрядный адрес можно получить разными сочетаниями сегмента и смещения, например B800:0000h и B000:8000h

При адресации к памяти сегментная часть адреса подразумевается находящейся в одном из специальных сегментных регистров. Если регистр не указывается в команде в явном виде, то

- при адресации к кодам (переходах) сегментная часть адреса находится в CS;
- при адресации к ячейкам памяти с данными – в DS;
- при адресации к стеку – в SS.

3.2. Регистровая модель процессора

Регистры процессора 8086 16-разрядные. Часть из них является регистрами общего назначения (РОН) и может использоваться программистом более-менее свободно, другие регистры являются регистрами специального назначения и "жизненно важны" для правильного функционирования процессора.

К РОНам относятся регистры AX, BX, CX, DX, с некоторой натяжкой – SI и DI. Младший и старший байты регистров AX, BX, CX и DX доступны по отдельности, для их обозначения употребляются имена AL, AH, BL, BH и т.д.

AX – accumulator, аккумулятор. Используется некоторыми командами в качестве жестко заданного источника или приемника данных (в других командах данные могут находиться в произвольных регистрах).

BX – base, база. Может использоваться в качестве адреса при косвенной адресации к памяти.

CX – counter, счетчик. Используется для задания количества повторений в циклических командах.

DX – data, дополнительный регистр данных. Используется как расширение аккумулятора при 32-разрядных арифметических операциях а также в качестве адреса порта при косвенной адресации к портам ввода-вывода.

SI – source index, индекс источника. Используется при косвенной адресации к памяти. В строковых командах пара DS:SI содержит адрес источника данных.

DI – destination index, адрес приемника. Используется при косвенной адресации к памяти. В строковых командах пара ES:DI содержит адрес приемника данных.

Регистры специального назначения:

BP – base pointer, указатель базы в стеке. Используется в качестве смещения при косвенной адресации к стеку. Используется компиляторами с языков высокого уровня для организации стековых кадров процедур. Его "порча" приводит в этом случае к краху системы. В простых программах на языке ассемблера, не использующих передачу параметров в процедуры через стек, может использоваться как РОН.

SP – stack pointer, указатель стека. Пара SS:SP содержит адрес последней занятой ячейки стека. При помещении информации в стек автоматически уменьшается, при извлечении – увеличивается. Порча SP приводит к краху системы.

IP – instruction pointer, указатель текущей команды. Программно недоступен, т.е. не может быть в явном виде прочитан или изменен. Пара CS:IP содержит адрес команды, исполняемой процессором. После выполнения команды IP увеличивается на длину кода команды, т.е. происходит переход к следующей команде. При выполнении переходов изменяется IP или оба регистра (CS:IP). Любое изменение IP означает переход.

Flags – регистр флагов. Не может быть в явном виде прочитан или изменен, однако отдельные флаги могут быть проанализированы в командах условных переходов, а также (некоторые) модифицированы.

Сегментные регистры:

CS – code segment, сегмент кода. Содержит сегментную часть адреса текущей команды (CS:IP). Любое изменение CS означает переход.

SS – stack segment, сегмент стека. Содержит сегментную часть адресов в стеке при косвенной адресации с использованием BP. Вершина стека – SS:SP. Изменение SS означает изменение адреса тека, чревато крахом системы и должно производиться (при необходимости), когда прерывания запрещены.

DS – data segment, сегмент данных. Содержит сегментную часть адреса, используемую по умолчанию при прямом или косвенном обращении к данным. При выполнении строковых команд пара DS:SI содержит адрес источника данных. Может изменяться в соответствии с нуждами программиста.

ES – extended segment, дополнительный сегмент. Может указываться **в явном виде** в качестве сегментной части адреса при обращении к данным. При выполнении строковых команд пара ES:DI содержит адрес приемника данных. Может изменяться в соответствии с нуждами программиста.

3.3. Виды адресации

В системе команд процессора 8086 реализованы следующие виды адресации:

1) Регистровая. Операнд находится в регистре, который указан в команде.

2) Непосредственная. Операнд является числом, непосредственно указанным в команде.

3) Прямая. Операнд находится в памяти, адрес ячейки (смещение) указан в явном виде в команде. Адрес на языке ассемблера пишется в квадратных скобках.

4) Косвенная. Операнд находится в ячейке памяти, адрес ячейки содержится в регистре, указанном в команде. Адрес на языке ассемблера пишется в квадратных скобках.

5) Стековая. Операнд заносится в стек или извлекается из вершины стека. Адрес вершины стека неявно задан в паре регистров SS:SP.

К разновидностям косвенной адресации относятся:

1) Базовая адресация. В регистре указывается база, адрес образуется сложением этой базы и константы.

2) Индексная адресация. Начало структуры данных задается константой, смещение вычисляется как сумма этой константы и индекса, задаваемого в регистре.

3) Базово-индексная адресация. В регистре указывается база (BX или BP), адрес образуется сложением этой базы, индекса, указываемого в другом регистре (SI или DI), и, возможно, константы.

Следует заметить, что индексация является базовой или индексной в голове программиста, а не в системе команд процессора. И индексные (SI, DI), и базовые (BX, BP) регистры могут использоваться как при построении схемы (1), так и (2).

При прямой и косвенной адресации к памяти по умолчанию в качестве сегментной части адреса используется содержимое регистра DS.

При косвенной адресации с использованием BP идет обращение к стеку, и сегментная часть адреса берется из SS.

3.4. Система команд

3.4.1. Команды пересылки данных

Пересылка осуществляется командой

MOV <куда>, <откуда>

Оба операнда могут адресоваться регистрово, непосредственно, прямо или косвенно. Пересылка память-память невозможна (один из операндов должен быть в регистре или задан непосредственно).

Примеры:

```
mov   ax,10 ; занести число 10 в регистр ax
mov   [20],ax; занести содержимое ax в память по адресу ds:20d
```

3.4.2. Арифметические команды (простейшие)

ADD <куда>, <что> - сложение
SUB <откуда>, <что> - вычитание
AND <что>, <с чем> - побитовое И
OR <что>, <с чем> - побитовое ИЛИ
NOT <что> - побитовая инверсия (НЕ)
NEG <что> - смена знака числа
DEC <что> - уменьшение (декремент) числа на единицу
INC <что> - увеличение (инкремент) числа на единицу

Операнды могут быть заданы как в MOV. Операций память-память нет. Результат помещается на место первого операнда и формируются флаги, например – флаг нуля (Z), переноса (C), отрицательности (S).

Для сравнения двух значений используются неразрушающее вычитание и неразрушающее И. Операция вычитания или побитовой конъюнкции производится, но первый операнд не изменяется, а только формируются флаги.

CMP <откуда>, <что> - неразрушающее вычитание

TEST <op1>,<op2> - неразрушающее И

3.4.3. Переходы

Безусловный переход:

JMP <метка>

Условные переходы выполняются на основании проверки значений флагов (одного или нескольких).

JZ (Jump if Zero) и JE (Jump if Equal) производят переход, если флаг нуля взведен, т.е. в результате предыдущей арифметической команды получился нулевой результат. Соответственно, команды JNZ и JNE производят переход, если флаг сброшен.

JZ и JE, JNZ и JNE – разные мнемоники одной и той же команды, введенные для лучшей читаемости программы. Если по логике программы важно, равен ли результат нулю, используются команды JZ или JNZ, если же нулевой результат означает равенство при сравнении (например, путем неразрушающего вычитания или операции И), то запись JE или JNE повышает читаемость программы.

Переходы по флагу переноса – JC и JNC.

Существует множество других команд условного перехода.

3.4.4. Организация циклов

Для организации циклов используется команда LOOP. Команда уменьшает значение CX на единицу и производит переход на указанную метку (зацикливание), если CX больше нуля, иначе выполняется следующая команда.

Цикл организуется так:

```
      mov   cx, <количество повторов>
m1:                                 ; начало цикла
      ;..... тело цикла ...........
      loop  m1
```

С использованием произвольного регистра, например, DX, данная последовательность эквивалентна следующей:

```
      mov   dx, <количество повторов>
m1:                                 ; начало цикла
      ;..... тело цикла ...........
      dec   dx
      jnz   m1    ; переход, если при уменьшении DX получился не ноль
```

4. Сведения о синтаксисе языка ассемблера

Алфавит языка ассемблера включает большие и малые латинские буквы, цифры, специальные знаки. Регистр символов для компилятора безразличен.

Синтаксической единицей языка ассемблера является строка. На одной строке должна размещаться одна машинная команда или директива. Допустимы пустые строки.

Комментарии записываются после символа "точка с запятой" и могут располагаться как на целой строке, так и в конце строки после команды. Например:

```
; Вся строка – комментарий
      xor   ax,ax ;обнуление регистра ax
```

Строки и отдельные символы заключаются в апострофы:

```
'A', 'Hello, world'
```

Если, например, код символа нужно поместить в регистр, можно указать его не в виде числа, а в виде символа:

```
mov   al,'!'  ; эквивалентно mov    al,33
```

Числа могут записываться в различных системах счисления. Наиболее употребительны десятичная, двоичная и шестнадцатеричная система счисления. По умолчанию принимается, что система счисления десятичная. Признаком системы счисления является буква "B", "D" или "H" после числа. Шестнадцатеричные числа, начинающиеся с буквы, должны записываться с ведущим нулем. Например:

100 – число "сто"

100D = 100

100DH = 4109

100h = 256

100B = 4

E8B – идентификатор "E8B", для числа требуется ведущий ноль

0E8B – синтаксически неверная конструкция, т.к. "B" на конце означает двоичный формат

0E8Bh = 3723

4.1. Запись команд

При **записи команд** следует придерживаться формата:

```
[<метка>:] [<мнемоника>     [<параметры>]] [<комментарий>]
```

при этом в качестве разделителя полей удобно использовать символ табуляции. Метка обычно используется, когда на команду, кодируемую в данной строке, предполагается выполнять переход. Обычно метка начинается с буквы или знака подчеркивания и состоит из букв, цифр и знаков подчеркивания. Метка завершается двоеточием.

Адреса данных, если они указываются в команде, записываются в квадратных скобках.

Пример (запись цикла):

```
; Небольшая задержка – цикл из 10 пустых команд
      mov   cx,10 ; cx – счетчик повторений
m1:   nop               ; пустая операция
      loop  m1          ; декремент cx, переход если >0
```

4.2. Объявление данных

Для использования **ячеек памяти** для хранения данных (в языках высокого уровня для этого используются статические переменные) эти ячейки необходимо:

- зарезервировать, указав размер (1, 2, 4 байта и т.д.);
- назначить нужным ячейкам имена (метки данных);
- описать начальные значения, которыми эти ячейки будут проинициализированы при загрузке программы.

Резервирование ячеек производится, например, директивами DB, DW, DD – "define byte", "define word", "define double".

Метки данных записываются без двоеточия и могут отсутствовать.

Пример описания данных:

```
S       db      'Hello,' , 32       ; 32=20h – код пробела
        db      'World'
        db      '$'
N       dw      ?                   ; ? = значение не определено или не важно
D       dd      258
```

Данный код резервирует 19 байт подряд, причем байту по смещению 0 сопоставляется символьное имя S, и он и последующие 12 байт содержат коды символов строки 'Hello, World$', по смещению 13 – символьное имя N, и два байта по этому адресу не требуют инициализации, и 15 – D, где размещается код 00000102h (4 байта). Обратите внимание (в первой строке), что символы, строки и числа можно смешивать в одной директиве.

Вид 18-байтового поля, определяемого в приведенном выше примере, таков:

H'	e'	l'	l'	o'	,'	'	W'	o'	r'	l'	d'	$'	0 0 h	0 0 h	0 2 h	0 1 h	0 0 h	0 0 h
S													N		D			

4.3. Сегментация в программе

Код, данные и стек программы могут располагаться в отдельных блоках памяти, называемых сегментами. Максимально возможный размер сегмента определяется механизмом адресации процессора и для изучаемого процессора не может превышать 64 кб. Распределение кода и данных по сегментам описывается в исходном тексте программы и хранится в заголовке исполнимого файла .EXE. В соответствии с этой информацией загрузчик операционной системы распределяет память во время загрузки программы (перед ее выполнением).

В разрабатываемой в данной работе простейшей программе на языке ассемблера, компилируемой в формат .EXE, проще всего иметь по одному отдельному сегменту для кода (собственно машинных команд), данных и стека. Такая модель памяти имеет стандартное обозначение small.

Начальное содержимое сегмента стека не важно, важен лишь его размер, поэтому стек описывается одной директивой .stack с указанием размера. 256 байт стека вполне достаточно для простейших программ, не использующих стек для хранения локальных переменных процедур и рекурсии:

```
.stack 100h
```

Содержимое сегмента данных описывается после директивы .data. Обычно в сегменте данных располагаются директивы резервирования ячеек (DB, DW и т.д.). Хотя там при желании можно разместить и мнемоники команд.

Содержимое сегмента кодов описывается после директивы .code. Обычно в сегменте кодов располагаются мнемоники машинных команд, но часто там присутствуют также и директивы резервирования ячеек памяти. Если специальным образом не указать точку входа в программу, то выполнение команд начнется по порядку с самой первой команды в сегменте кодов. Для явного указания точки входа необходимо использовать метку и указать ее после директивы END в конце программы, например:

```
.code
      ..................................  ; здесь могут быть команды
mm:   mov   ax,@data
      mov   ds,ax
      ..................................  ; команды
      end   mm          ; mm – точка входа
```

5. Предмет исследования

5.1. Текстовый вывод

Графический адаптер компьютера IBM PC (VGA-совместимый) может функционировать в графическом и текстовом режиме. В первом случае программно доступна каждая точка на экране (пиксель), во втором – можно удобно и быстро вывести текст.

В самом распространенном текстовом режиме 3 на экране имеется 2000 знакомест (ячеек размером 8x16 пикселей), организованных в 25 строк по 80 символов в каждой. Нумерация строк и столбцов идет с нуля.

Изображение на экране строится аппаратурой графического адаптера в соответствии с образом экрана в видеопамяти (видеобуфере). Изменение информации в видеобуфере немедленно отображается на экране. В текстовом режиме образ экрана начинается по адресу B800:0000h и содержит 2000 двухбайтовых ячеек (по количеству знакомест, сначала – строка номер 0, затем – строка номер 1 и т.д.). Младший байт ячейки содержит ASCII-код отображаемого символа, старший – его атрибут (информацию о цвете).

Атрибут задает цвет символа (младшая тетрада) и цвет фона (старшая тетрада), а также атрибут мигания. Биты байта атрибута трактуются следующим образом:

```
K R G B i r g b
| | | | | | | +- синий луч при отображении символа
| | | | | | +--- зеленый луч при отображении символа
| | | | | +----- красный луч при отображении символа
| | | | +------- интенсивность = повышенная яркость символа
| | | +--------- синий луч при отображении фона
| | +----------- зеленый луч при отображении фона
| +------------- красный луч при отображении фона
+--------------- символ мигает
```

Например, атрибут "желтый на синем" означает "яркий+красный+зеленый на синем", и будет выглядеть как 0001 1110 b = 1Eh. **Нулевой атрибут** означает "черный на черном"; текст с таким атрибутом (как и при любом совпадении цвета символа и цвета фона) **невидим**.

В текстовом режиме в видеопамяти зарезервировано место не для одного, а сразу для восьми образов экрана (по 4 килобайта на каждый), называемых видеостраницами. По умолчанию на экране отображается нулевая видеостраница, остальные невидимы. Можно сделать видимой любую другую видеостраницу.

Для вывода текста из программы можно как непосредственно изменять информацию в видеобуфере, так и использовать сервисные функции, предоставляемые DOS и BIOS. Как

правило, функции BIOS напрямую работают с видеобуфером, а функции DOS содержат вызовы функций BIOS.

5.2. Использование "прерываний"

Сама по себе система прерываний процессора предназначена для организации программной реакции на асинхронные внешние события в вычислительной системе (нажатие клавиш на клавиатуре, прием очередной порции данных портом и т.п.). Однако разработчики архитектуры IBM PC и операционной системы MS DOS решили, что механизм прерываний также удобно использовать для организации доступа программ к сервисным функциям операционной системы и BIOS. Соответствующие прерывания генерируются программно, управление передается обработчику, который в свою очередь вызывает нужную сервисную функцию. В частности, прерывание 10h используется для вызова функций графической подсистемы BIOS, а 21h – для вызова большинства функций DOS. При этом в среде программистов общепринятым является оборот речи "воспользоваться функцией такого-то прерывания" или просто "воспользоваться прерыванием DOS". Именно поэтому слово "прерывание" в заголовке взято в кавычки.

Чтобы воспользоваться какой-либо из рассматриваемых сервисных функций, необходимо:

1) Выяснить (по справочнику) точный формат вызова функции. В качестве справочного материала по функциям прерываний в ходе данного цикла лабораторных работ предлагается файл **interrupt.lst** (на английском языке). Существует огромное количество бумажной и электронной литературы аналогичного содержания.

2) Загрузить необходимые значения в регистры процессора (необходимо проинициализировать все упомянутые в справочнике регистры!). Номер функции при этом заносится в регистр AH.

3) Выполнить команду **INT n**, где n – номер нужного прерывания.

Следует помнить, что:

- функции возвращают результат в одном или нескольких регистрах, при этом часто используется регистр флагов;
- функции прерываний DOS могут портить состояние регистров, в том числе тех, в которых передавались параметры; не следует полагаться на то, что после вызова функции состояние регистров такое же, как было до вызова;
- адреса в функцию всегда передаются полностью, и сегмент, и смещение; при этом может потребоваться, например, записать в регистр DS адрес, отличный от адреса начала сегмента данных программы.

Например, имеется следующее описание:

```
-------------------------------------------------------
INT 10 - VIDEO - WRITE STRING (AT,XT286,PS,EGA,VGA)
       AH = 13h
       AL = mode
           bit      0: set in order     to move      cursor after write
           bit      1: set if string contains alternating characters and attributes
       BL = attribute if AL bit 1 clear
       BH = display page number
       DH,DL =      row,column of starting cursor position
       CX = length of string
       ES:BP =      pointer      to start of string
Note: recognizes CR, LF, BS, and bell
-------------------------------------------------------
```

Оно означает, что функция 13h прерывания 10h предназначена для вывода на экран строки. Регистры ES и BP должны содержать сегментный адрес и смещение начала выводимой строки соответственно. В CX должна быть задана длина строки в байтах. Регистры DH и DL должных содержать экранные координаты Y и X начала вывода, BH – номер страницы дисплея, два младших бита AL кодируют дополнительные режимы вывода, а регистр BL должен содержать атрибут (цвет) для выводимых символов, если в AL выбран соответствующий режим.

Бит 0 в регистре AL кодирует, будет ли производиться перемещение курсора при выводе строки в ее конец. Бит 1, если он взведен, означает, что в строке по адресу ES:BP закодированы атрибуты для каждого символа, т.е. байты строки трактуются как символ-атрибут-символ-атрибут... Если бит 1 сброшен, то строка содержит только символы, а атрибут должен быть задан в BL.

В примечании специально указано, что функция интерпретирует как управляющие символы CR (0Dh, возврат каретки - перевод курсора в начало строки), LF (0Ah, перевод строки, перевод курсора на строку вниз), BS (08h, забой – перевод курсора на один символ влево) и bell (07h – звуковой сигнал).

Функция не возвращает результатов.

Тогда следующий код выведет строку 'Hello' посередине экрана:

```
.data
s     db    'Hello'
.code
.........................................
      mov   ax,seg s
      mov   es,ax ; загрузить в ES сегментный адрес S
      mov   ah,13h; номер функции
      mov   al,1  ; обычная строка, переместить курсор в конец
      mov   bh,0  ; 0 страница, отображается по умолчанию
      mov   bl,00000100b; красный на черном
      mov   cx,5  ; длина строки
      mov   dh,25/2     ; Y, вычисляется целое выражение
      mov   dl,80/2-3   ; X
      mov   bp, offset s
      int   10h   ; вывести строку
```

6. Задание

Составить и отладить индивидуальную программу, реализующую взаимодействие с пользователем и содержащую вызов одной из указанных ниже функций DOS/BIOS.

Порядковый номер фамилии студента в журнале группы является номером его индивидуального задания. Если порядковый номер фамилии превышает длину списка заданий, то номер задания выбирается "по кругу", начиная с первого.

Варианты:

1. DOS Int 21h, func. 01h (ввод символа с эхо).
2. DOS Int 21h, func. 02h (вывод символа).
3. DOS Int 21h, func. 06h (ввод/вывод символа).
4. DOS Int 21h, func. 07h (ввод символа без эхо).
5. DOS Int 21h, func. 0Ah (буферизованный ввод строки).

6. DOS Int 21h, func. 08h (ввод символа без эхо).
7. DOS Int 21h, func. 0Bh (проверить состояние ввода).
8. BIOS Int 10h, func.2 и func 3. (позиция курсора).
9. BIOS Int 10h, func.1 (размер курсора).
10. BIOS Int 10h, func.6 или 7 (скроллинг экрана).
11. BIOS Int 10h, func.8 и 9 (чтение/запись символа).
12. BIOS Int 10h, func. 0Ah и 0Eh (запись символа).
13.* Прямой вывод символов в видеопамять.
14.* DOS Int 21h, func.40h (вывод строки символов на экран. Указание: Дескриптор устройства-Дисплей равен 1, т.е. BX=1).
15. BIOS Int 10h, func. 13h (вывод строки на экран. Проверить подфункции AL=1 и какую-либо еще).
16. Функции BIOS для посимвольного вывода строки на экран.
17. Вывод символов на экран (прерывания, которые отображают управляющие символы, и прерывания, которые не отображают их).