

МИНИСТЕРСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

КАФЕДРА : ИНФОРМАТИКА

Программирование на Ассемблере  
для персональных компьютеров

Учебно-методическое пособие

РОСТОВ-НА-ДОНУ 2001г.

Составители : доц., к.т.н. Садовой Н.Н., доц., к.т.н. Жмайлов Б.Б.

УДК 681.3

Программирование на Ассемблере для персональных компьютеров:  
Учебно-методическое пособие / ДГТУ / г. Ростов-на-Дону, 2001 г. ... с.

Учебно-методическое пособие содержит теоретический материал к лабораторному практику по разделу "Программирование на языке Ассемблера" курса "Организация и функционирование ЭВМ ". В учебно-методическом пособии "Программирование на Ассемблере для персональных компьютеров" предлагается методика последовательного изложения, которая облегчает понимание материала студентами. Кроме того, пособие разработано с учетом возможности самостоятельной работы студентов заочного факультета. Пособие предназначено для студентов специальностей 220400, 351500, 075200, 6547 и 3514 дневного отделения ДГТУ и филиала в г. Азове.

Печатается по решению методической комиссии факультета 'Автоматизации и информатики '.

Научный редактор профессор, кандидат технических наук

Землянухин В.Н.

- Донской государственный технический университет

## 1. Знакомство с программой-отладчиком Turbo Debugger. Выполнение простейших команд микропроцессора в среде Turbo Debugger.

### 1.1. Понятие отладки. Назначение программ-отладчиков

**Отладка** (debugging) — один из важнейших этапов разработки программного обеспечения (английский термин bug означает "ошибка в программе"). В процессе отладки путем детального анализа в компьютерных программах выявляются и устраняются возможные логические ошибки, которые не обнаруживаются на стадии компиляции.

**Отладчики** (debugger) — это вспомогательные программы (утилиты), включаемые в набор инструментальных средств программиста для выполнения отладки других программ. Отладчики предоставляют программисту возможность выполнять программу по шагам, следить за изменениями данных и проверять выполнение условий. В зависимости от уровня языка, которым оперирует отладчик, можно выделить два их типа.

*Отладчики исходного кода* дают программисту возможность видеть текст программы на языке высокого уровня (например, Си), проверять значения отдельных переменных и агрегатов данных (таких, как массивы), используя их имена.

*Отладчики машинного уровня* отслеживают реально выполняемые машинные команды, отображаемые в виде команд ассемблера. Они позволяют также просматривать содержимое ячеек памяти и регистров микропроцессора.

Отладчики, интегрированные в среду разработки пакетов программ, например, Borland C++, Borland Pascal, относятся к первому типу.

Рассматриваемый здесь Turbo Debugger относится к отладчикам второго типа.

## 1.2. Основные режимы работы отладчика Turbo Debugger

Запуск программы осуществляется файлом `td.exe`, расположенный в директории BIN каталога BP или BC.

### 1.2.1. Структура экрана программы Turbo Debugger

При запуске Turbo Debugger на экране появляется его основное меню и рабочее окно рис.1.

Рабочее окно состоит из следующих четырёх окон:

1. окно команд – **CPU**;
2. окно регистров и флагов – **Registers**;
3. окно данных - **Dump**;
4. окно стека.

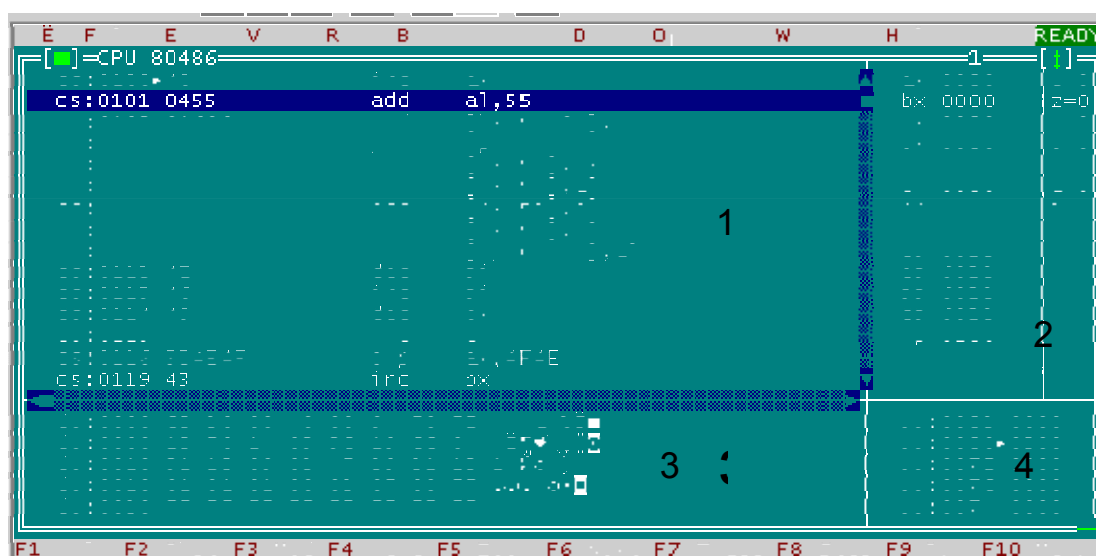


Рисунок 1

В свою очередь окно **Registers** поделено на две части. В левой его части указано содержимое всех регистров микропроцессора (ax,bx,cx,dx...), а в правой части показаны биты регистра флагов.

### 1.2.2. Команды работы с Turbo Debugger

Команды	Горячие клавиши	Описание
Переключение между окнами	Tab Shift+Tab	По часовой стрелке Против часовой стрелки
Размер окна	F5	Увеличение/уменьшение размеров окна
Закреть окно	Alt+F3	
Загрузка программы	-	Меню File/Open...
Запуск программы	F9	Программа выполняется до конца или до точки останова
Установка точки останова	Alt+F2	Установить маркер на требуемую команду, а затем Alt+F2. Команда выделяется красным цветом.
Снятие точки останова	F2	Установить маркер на требуемую команду, а затем F2. Снимается выделение красным цветом.
Пошаговое выполнение	F7	
Запуск после точки останова	F9	
Выбор исполняемой команды	Ctrl+N	Позволяет сделать очередной любую команду

Изменение данных	Ctrl+C	Установить маркер на редактируемую ячейку или регистр, а затем Ctrl+C
Установка типа изменяемых данных	Ctrl+D	При редактировании ячейки памяти можно установить: byte, word и т.д.
Просмотр памяти	Ctrl+G	В диалоговом окне ввести адрес начала области просмотра
Поиск	Ctrl+S	Поиск необходимой инструкции или содержимого ячейки памяти

Порядок выполнения работы. Загрузить в отладчик небольшой по объему файл с расширением `.exe` и выполнить все команды приведенные в таблице.

### **1.3. Вопросы по теме**

1. Каковы задача и содержание этапа отладки программ?
2. Типы программ-отладчиков и особенности их работы.
3. Основные команды для работы с ячейками памяти и регистрами отладчика Turbo Debugger ?
4. Основные команды для отладки загруженной программы ?
5. Основные команды для работы с окнами отладчика Turbo Debugger ?

## 2. Начальные сведения о языке Ассемблер

### 2.1. Аппаратная поддержка языка

При выполнении программы, микропроцессор взаимодействует с оперативной памятью, где хранятся исполняемая программа и данные, а также с периферийными устройствами.

Программируемая структура процессора

Для организации вычислений микропроцессор i8086 имеет в своём составе 14 шестнадцатиразрядных регистров, которые обеспечивают выполнение программы:

Регистры общего назначения			Сегмент- ные регистры		Специ- альные регистры	
AH	AL	AX	CS		SP	Указатель стека
BH	BL	BX	DS		BP	Указатель базы стека
CH	CL	CX	ES		IP	Указатель инструкций
DH	DL	DX	SS		FLAGS	Регистр флагов
SI						
DI						

Регистры общего назначения:

AX(AH, AL), BX(BH, BL), CX(CH, CL), DX(DH, DL) делятся программно на пары однобайтных регистров и могут использоваться для хранения данных. Разбиение на однобайтные регистры позволяет увеличить общее число регистров;

SP, BP – указатель и база стека, соответственно, обеспечивают доступ к данным в стеке, могут использоваться для хранения данных, но делать

это не рекомендуется, так как при этом возможно нарушение адресации в стеке, особенно при использовании SP.

SI, DI – шестнадцатиразрядные регистры для хранения данных.

CS, DS, ES, SS – хранят адреса сегментов в памяти, не могут использоваться для хранения данных.

IP – регистр инструкций – хранит адрес(смещение) следующей исполняемой команды.

FLAGS – регистр флагов содержит набор битовых флагов, определяющий текущее состояние процессора и результат выполнения предыдущей команды (таблица 2.1).

Таблица 2.1

Регистр флагов процессора		
Флаг	Название	Назначение
O	Переполнение	Переполнение при выполнении арифметических операций
D	Направление	Направление пересылки данных при выполнении строковых команд
I	Прерывание	Разрешает/Запрещает внешние прерывания
T	Пошаговый режим	Останов после выполнения каждой команды(используется отладчиками)
S	Знак	Знак результата выполненной команды(0 – плюс, 1 – минус)
Z	Ноль	Значение результата выполненной команды(0 – ненулевой, 1 – нулевой)
A	Внешний перенос	Используется для специальных арифметических операций



Р	Контроль чётности	Число единиц в операнде(0 – нечётное, 1 – нечётное)
С	Перенос	Содержит перенос из старшего бита при выполнении арифметических операциях

### 2.1.1. Структура памяти

Память, с которой взаимодействует процессор при обработке программ, называется Оперативным Запоминающим Устройством(ОЗУ) или Random Access Memory(RAM). Она состоит из набора однобайтных ячеек, обращение к которым происходит по их номерам(физическим адресам). Число ячеек зависит от ширины шины адреса и составляет для процессора i8086(ширина шины адреса равна 20)  $2^{20}$  – ячеек(1Мбайт). Для современных процессоров с шириной шины адреса 32 объём ОЗУ может достигать до 4 Гбайт.

Данные можно читать или сохранять в ОЗУ байтами, указывая номер требуемой ячейки или словами(2 байта), указывая адрес младшей ячейки памяти и вводя специальный префикс.

### 2.1.2. Сегментация памяти

Для обращения к памяти процессор предварительно помещает адрес ячейки в один из своих регистров, но для процессора i8086, очевидно нельзя в шестнадцатиразрядном регистре хранить двадцатиразрядный адрес. Поэтому применяют так называемую сегментацию памяти, которая заключается в том, что истинный, физический адрес ячейки хранится в двух регистрах.

Один из них – сегментный, он хранит адрес начала блока памяти, который и называется сегментом. Если к шестнадцати разрядам сегмента мысленно справа дописать четыре двоичных нуля( $16+4=20$ ), то получим

физический адрес начала сегмента в ОЗУ. Второй регистр хранит величину смещения адреса требуемой ячейки от начала сегмента. Адрес ячейки памяти записывается в виде двойного слова(4 байта): <сегмент>:<смещение>.

Сегмент всегда начинается с ячейки, номер которой заканчивается на 4 двоичных(или один шестнадцатеричный) нуля. Минимальная длина сегмента 16 байтов(параграф). Максимальная длина определяется длиной регистра, хранящего смещение и равна  $2^{16}$ (64 Кбайта).

Пара регистров CS:IP(<сегмент>:<смещение>) определяют адрес следующей команды программы.

Для адресации данных используются сегментные регистры DS и ES, а в качестве регистров, хранящих смещение, используются регистры общего назначения BX, SI, DI. Для работы с сегментом стека используют сегментный регистр SS и регистр BP.

## ***2.2. Структура программы на языке Ассемблер***

Программа на языке ассемблера представляет собой текст разбитый на строки. Каждая строка либо соответствует машинной команде, либо является директивой ассемблера или макрокомандой. Команды и директивы можно набирать как большими, так и малыми латинскими буквами. Русские буквы можно использовать только в комментариях.

<имя сегмента> **segment**

команды или директивы

<имя сегмента> **ends**

[

<имя сегмента> **segment**

команды или директивы

<имя сегмента> **ends** ]

**end** <метка входа в программу>

Директива **end** < метка входа в программу> отмечает конец текста программы и указывает ассемблеру, где завершить трансляцию. Поэтому директива **end** должна присутствовать в каждой программе.

< метка точки входа > указывает инструкцию с которой должно начинаться выполнение программы.

Каждая программа содержит сегменты данных и команд, но минимально должна содержать сегмент команд.

Строка программы, в общем случае, состоит из четырех полей:

Поле метки	Поле операции	Поле операндов	Поле комментария
------------	---------------	----------------	------------------

M1:	Add	AX, BX	; сложение
-----	-----	--------	------------

Имена данных, процедур, сегментов или метки команд могут состоять не более чем из 31 латинских букв и цифр, причем первым символом должна быть обязательно буква. Большие и маленькие буквы не различаются.

### 2.2.1. Директивы ассемблера

Директивой называется команда транслятору для выполнения определённых данной директивой действий, сама директива в текст транслированной программы не включается.

#### 1. Директива задания исходных данных:

[<имя>] **d**<тип> <константа>[,<константа>, <константа>, . . .]

- <имя> - имя массива данных, по которому к ним можно обратиться из команды;
- **d** (define) – определяет начало массива данных;
- <тип> - размер констант, входящих в массив:

b – байт,

w – слово(два байта),

d – двойное слово,

q – учетверённое слово,

t – десять байтов;

- <константа> - числовой или символьный элемент массива данных.

В ассемблере используется несколько типов констант:

*десятичные* – последовательность цифр от 0 до 9;

*шестнадцатеричные* – последовательность шестнадцатеричных цифр от 0 до 9 и от A или a до F или f завершающаяся буквой H или h, первой должна быть десятичная цифра или 0;

*восьмеричные* – последовательность цифр от 0 до 7, завершающаяся буквами Q или q;

*двоичные* – последовательность цифр от 0 до 1, завершающаяся буквой B или b;

*символьные* – символ или группа символов, заключённые в кавычки;

*знак ?* – используется для резервирования места для данных.

Например,

data1 db 123, 0a2h, 75q, 110011b, 'a', 'пример', ?, ?

Для заполнения больших массивов используется директива `dup` (`duplicate`):

`<число повторений> dup(<образец>)`

`<число повторений>` - задаёт количество размещаемых в памяти данных, определяемых образцом;

`<образец>` - любая допустимая группа констант.

Например,

```
data2 db 23 dup(1, 2, 'x')
```

выделяет в памяти  $23 \cdot 3 = 69$  байтов и заносит в них образец 1, 2, 'x', 1, 2, 'x', ... .

2. Директива использования сегментных регистров по умолчанию:

```
assume <имя сегментного регистра>:<имя сегмента или  
nothing>[, <имя сегментного регистра>:<имя сегмента или nothing>,  
...]
```

Как отмечалось выше, для задания адреса в памяти требуется два регистра, один из них всегда сегментный, поэтому в команде при обращении к памяти приходится набирать имя сегментного регистра, часто одного и того же. Директива **assume** позволяет избежать этого. Транслятор сопоставляет имя массива данных и автоматически подставляет сегментный регистр, заданный для сегмента, в котором расположен данный массив. Слово **nothing** показывает, что данный сегментный регистр не адресуется по умолчанию. Директива **assume** может использоваться в программе при каждом изменении сегмента для данного сегментного регистра, но обязательно в начале сегмента, где она задаёт по умолчанию сегментный регистр для сегмента кодов.

Например,

```
assume cs:code, ds:data1, es:nothing
```

Здесь code и data1 – имена сегментов кодов и данных, соответственно.

### 2.2.2.Режимы адресации

1. Регистровая прямая - операнд находится в регистре.

Обозначение - <регистр> ,

< регистр > - AX, BX, CX, DX, SI, DI, BP, SP, AL, BL, CL, DL, AH, BH, CH, DH.

Пример:

**mov** AX,SI ; переслать содержимое регистра SI в регистр AX.

2. Непосредственная - непосредственный операнд (константа) присутствует в команде.

Обозначение - < константное выражение > .

Пример:

**mov** AX, 093Ah ; занести константу 093Ah в регистр AX.

3. Прямая - исполнительный адрес операнда присутствует в команде.

Обозначение - < переменная > +/-< константное выражение > .

Пример:

**mov** AX, WW ; переслать в AX слово памяти с именем WW

**mov** BX, WW+2 ; переслать в BX слово памяти отстоящее от переменной с именем WW на 2 байта.

4. Регистровая косвенная - регистр содержит адрес операнда.

Обозначение - [< регистр >],

< регистр > - BX, BP, SI, DI.

Пример:

**mov** [ BX ], CL ; переслать содержимое регистра CL по адресу, находящемуся в регистре BX.

5. Регистровая относительная - адрес операнда вычисляется как сумма содержимого регистра и смещения.

Обозначение - < переменная >[< регистр >] или [< регистр >]< константное выражение > ,

< регистр > - SI или DI индексная адресация, BX или BP - базовая адресация.

Пример:

**mov** AX, WW[SI] ; переслать в AX слово из памяти, адрес которого вычисляется как сумма содержимого регистра SI и смещения WW.

6. Индексно - базовая - адрес операнда вычисляется как сумма содержимых базового и индексного регистров и смещения.

Обозначение - [< базов. регистр>][< индексн. регистр>] или <переменная>[<базов. регистр >][< индекс. регистр >] или [<базов. регистр >][< индекс. регистр >]< константное выражение ,

где < индекс. регистр > - SI или DI, < базов. Регистр > - BX или BP.

Пример:

**mov** [BX+ SI+ 2], CL; переслать содержимое регистра CL по адресу, вычисляемому как сумма содержимого регистров BX, SI и константы 2.

### ***2.3. Инструкции пересылки данных и двоичной арифметики***

Команды данной группы приведены в таблице 2.1. Код определяет выполняемое командой действие, операнды показывают адреса ячеек, хранящих исходные данные, необходимые для выполнения команды и адрес ячейки результата. Процессор i8086 и более поздние версии относятся к двухадресным машинам. Это значит, что его команда может содержать не более двух операндов. Если для выполнения команды необходимо иметь два источника данных, например, сложение, то сохранение результата вы-

полнения команды производиться по адресу одного из источников данных. Чтобы показать, какой из операндов будет хранить результат, его обозначают при описании команды как `dst`(`destination` - назначение), операнд, который используется только как адрес исходных данных, обозначается как `src`(`source` – источник). В двухоперандных командах операнд `dst` указывает, перед выполнением команды, адрес исходного данного, а после выполнения - адрес результата.



Таблица 2.2

## Команды пересылки и двоичной арифметики

Мнемокод		Флаги						Действие
Код	Операнды	O	S	Z	A	P	C	
mov	dst, src.	-	-	-	-	-	-	пересылка
xchg	dst, src	-	-	-	-	-	-	обмен
add	dst, src	x	x	x	x	x	x	сложение
adc	dst, src	x	x	x	x	x	x	сложение с переносом
inc	dst	x	x	x	x	x	-	увеличить на единицу
sub	dst, src	x	x	x	x	x	x	вычитание
sbb	dst, src	x	x	x	x	x	x	вычитание с заемом
dec	dst	x	x	x	x	x	-	уменьшение на единицу
neg	dst	x	x	x	x	x	x	изменение знака
rcl	dst, счетчик	x	-	-	-	-	x	циклический сдвиг влево
rcr	dst, счетчик	x	-	-	-	-	x	циклический сдвиг вправо
rol	dst, счетчик	x	-	-	-	-	x	циклический сдвиг влево
ror	dst, счетчик	x	-	-	-	-	x	циклический сдвиг вправо
sal	dst, счетчик	x	x	x	u	x	x	арифметический сдвиг влево
sar	dst, счетчик	x	x	x	u	x	x	арифметический сдвиг вправо
shl	dst, счетчик	x	x	x	u	x	x	логический сдвиг влево
shr	dst, счетчик	x	x	x	u	x	x	логический сдвиг вправо
push	src	-	-	-	-	-	-	сохранение слова в стеке
pop	dst	-	-	-	-	-	-	восстановление слова из стека
xlat	таблица	-	-	-	-	-	-	трансляция байтов из таблицы
lea	dst, src	-	-	-	-	-	-	загрузка исполнительного адреса
lds	dst, src	-	-	-	-	-	-	загрузка указателя с DS
les	dst, src	-	-	-	-	-	-	загрузка указателя с ES
lahf		-	-	-	-	-	-	загрузка флагов в AH
sahf		-	r	r	r	r	r	установка флагов из AH
pushf		x	-	-	-	-	x	сохранение флагов в стеке
popf		r	r	r	r	r	r	восстановление флагов из стека

Примечание:

- Флажок не модифицируется
- x Устанавливается или сбрасывается в соответствии с результатом;
- u Не определен;
- r Восстанавливается прежнее запомненное значение.

#### ***2.4. Запись программ на языке ассемблера***

Ниже приведена типичная структура простой программы на ассемблере.

```
data segment                ;директива начала сегмента данных
d1 dw 34h
d2 db 10100110b
d3 dd 3 dup (?)
data ends                  ; директива конца сегмента данных
code segment                ; директива начала сегмента кодов
assume cs: code, ds: data
start: mov ax,data           ; Загрузить адрес
      mov ds,ax             ; сегмента данных
      .
      .                     ; текст программы
quit:  mov ax,4c00h          ; Код завершения 0
      int 21h               ; Выход в DOS
code ends
end start
```

Загрузка адреса сегмента данных состоит из двух команд, так как непосредственные данные нельзя заносить прямо в сегментный регистр.

Для завершения программы и выхода в DOS имеется несколько возможностей, рекомендуется использовать две команды, начинающиеся с метки quit.

### **2.5. Обработка программ в MS-DOS**

Обработка программ на языке ассемблера в MS-DOS состоит из следующих этапов:

- Создать с помощью текстового редактора файл с текстом программы на языке ассемблера.
- Транслировать программу с помощью ассемблера TASM (или MASM);
- Скомпоновать программу с помощью компоновщика (редактора связей) TLINK(или LINK).
- Запустить программу на выполнение.

Файл исходного текста программы должен иметь расширение asm.

Запуск транслятора осуществляется командой

**tasm** <исходный файл >[, [< объектный файл >] [, [< файл листинга >] [, [< файл перекрестных ссылок >]]] [:]

Все создаваемые транслятором файлы будут иметь разные расширения имени, поэтому им можно оставить имя исходного файла:

**tasm** <исходный файл >, , , , ;

Точка с запятой показывает, какие файлы должен создать транслятор, например, конструкция

**tasm** <исходный файл >;

создаст только объектный файл.

Расширение объектного файла по умолчанию obj; расширение файла листинга по умолчанию lst; расширение файла перекрестных ссылок по умолчанию crf.

Компоновщик использует, как исходный, объектный файл и создаёт исполняемый файл с расширением по умолчанию exe.

Запуск компоновщика осуществляется командой:

tlink < объектный файл >[,< исполняемый файл >]

В случае сохранения имени исходного файла команда имеет вид:

tlink < объектный файл>

Для запуска под отладчиком необходимо запустить отладчик и загрузить исполняемый файл.

## **2.6. Пример выполнения работы**

Вычислить  $X = 3A + (B + 5) / 2 - C - 1$ ,

где A, B, C, X- целые знаковые числа занимающие слово, написать программу реализующую данную формулу.

Распишем формулу по отдельным операциям:

$AX \leftarrow A$  ; значение A в регистре AX

$AX \leftarrow 2 * (AX)$  ; 2A в AX

$AX \leftarrow (AX) + A$  ; 3A в AX

$BX \leftarrow B$  ; B в BX

$BX \leftarrow 5 + (BX)$  ; B+5 в BX

$BX \leftarrow (BX) / 2$  ; (B+5) / 2 в BX

$AX \leftarrow (BX) + (AX)$  ; 3A+(B+5) / 2 в AX

$AX \leftarrow (AX) - C$  ; 3A+(B+5) / 2 - C в AX

$AX \leftarrow (AX) - 1$  ; 3A+(B+5)/2 - C - 1 в AX

$X \leftarrow (AX)$  ; 3A+(B+5)/2 - C - 1 в X

Ниже приведена типичная структура простой программы на ассемблере.

#### 2.6.1.1 Текст программы:

```
data segment
a dw 10
b dw 20
c dw 5
x dw ?
data ends
code segment
    assume cs: code, ds: data
    start: mov ax, data
           mov dx, ax      ; загрузить адрес
           mov ax, a       ; сегмента данных
sal ax, 1
add ax, a
mov bx, b
add bx, 5
sar bx, 1
add ax, bx
sub ax, c
dec ax
mov x, ax      ; запись результата в память
quit:
    mov ax, 4c00h  ; код завершения 0
    int 21         ; выход в dos
code ends
```

end start

### **2.7. Варианты заданий**

Разработать программу реализующую указанную формулу, исполнить программу с несколькими ( три - четыре) наборами исходных данных, проверить правильность результатов.

- |                                      |                                       |
|--------------------------------------|---------------------------------------|
| 1. $X = A - 5(B - 2C) + 2$           | 14. $X = -(- (C + 2A) * 4B + 38)$     |
| 2. $X = -4A + (B + C) / 4 + 2$       | 15. $X = A - 3(A + B) + C \bmod 4$    |
| 3. $X = 7A - 2B - 100 + C$           | 16. $X = 3(A - 2B) + 50 - C / 2$      |
| 4. $X = -A / 2 + 4(B + 1) + 3C$      | 17. $X = (3A + 2B) - C / 4 + 217$     |
| 5. $X = 5(A - B) - 2C + 5$           | 18. $X = 3(C - 2A) + (B - C + 1) / 2$ |
| 6. $X = (A / 2 + B) / 4 + C - 1$     | 19. $X = (2A + B) / 4 - C / 2 + 168$  |
| 7. $X = -(C + 2A + 4B + B)$          | 20. $X = 6(A - 2B + C / 4) + 10$      |
| 8. $X = 6C + (B - C + 1) / 2$        | 21. $X = 3(A - 4B) + C / 4$           |
| 9. $X = 2 - B(A + B) + C / 4$        | 22. $X = -(- (C + 2A) * 5B - 27)$     |
| 10. $X = 2B - 1 + 4(A - 3C)$         | 23. $X = A / 2 - 3(A + B) + C * 4$    |
| 11. $X = (2A + B) / 4 - C / 2 + 168$ | 24. $X = 3(A - 2B) + 50 - C / 2$      |
| 12. $X = 6(A - 2B + C / 4) + 10$     | 25. $X = 5A + 2B - B / 4 + 131$       |
| 13. $X = 5(A - B) + C \bmod 4$       |                                       |

### **2.8. Вопросы по теме**

1. Назначение директив SEGMENT и ENDS.
2. Назначение директивы ASSUME.
3. Назначение директив DB, DW.
4. Назначение оператора DUP в директивах DB, DW.
5. Назначение директивы END.
6. Из каких полей состоит строка программы на ассемблере?
7. Какие поля обязательны, а какие можно опустить?

8. В чем различие между командами :

`mov AX, BX` , `mov AX, [BX]` и `mov [AX], BX` ?

9. В чем разница между командой `mov A, 1` и директивой `A dw 1` ?

### 3. Циклические и разветвляющиеся программы

Команда передачи, управления служит для передачи управления инструкции, не следующей непосредственно за данной. Управление может передаваться как внутри текущего сегмента кода (внутрисегментная передача управления), так и за его пределы (межсегментная передача управления). Тип передачи управления может быть задан ассемблеру предшествующим адресу перехода ключевым словом NEAR (внутрисегментная) или FAR (межсегментная ).

#### 3.1. Безусловные переходы

Инструкция безусловного перехода передаёт управление команде, адрес которой указан в инструкции. Команда безусловного перехода имеет вид

**jmp** [< тип > **ptr** ] операнд.

<тип> - тип перехода **short** (короткий) – смещение 127 байтов вперёд или 128 байтов назад, **near** (близкий) – смещение в пределах сегмента (64 Кбайта), **far** (дальний) – в любой сегмент с любым смещением.

**ptr** – приставка, которую можно перевести как *указанный в*.

Если тип не задан, по умолчанию принимается **near**.

Всего можно выделить пять типов безусловных переходов (таблица 3.1).

Таблица 3.1

Типы команд безусловного перехода		
Название	Мнемоника	Описание
внутрисегментный прямой короткий	jmp short <операнд>	IP ← (IP) + 8-битное смещение, опреде-



		ляемое операндом
внутрисегментный прямой близкий пе- реход	jmp near ptr <операнд>	IP $\leftarrow$ (IP)+16-битное смещение, опреде- ляемое операндом
внутрисегментный косвенный переход	jmp <адрес операнда>	IP $\leftarrow$ 16-битный адрес перехода
Межсегментный прямой далекий пе- реход	jmp far ptr <операнд>	IP $\leftarrow$ смещение опе- ранда в сегменте CS $\leftarrow$ адрес сегмен- та, содержащего опе- ранд
Межсегментный косвенный далёкий переход	jmp far ptr <адрес операнда>	IP $\leftarrow$ операнд CS $\leftarrow$ адрес операн- да +2

### 3.2. Условный переход

Команда условного перехода организует передачу управления при выполнении определённого в команде условия, в противном случае переход осуществляется на команду, следующую за инструкцией условного перехода. Условия определяются текущим состоянием флагов процессора. Каждая из 30 команд условных переходов проверяет определённую комбинацию флагов.

Все условные переходы являются короткими, т.е. адрес перехода должен отстоять не далее, чем на - 128 или +127 байтов от первого байта следующей команды.

Команды условной передачи управления и проверяемые при их выполнении условия приведены в таблице 3.2.

Таблица 3.2.

Инструкции условной передачи управления		
Мнемокод	условие перехода	
	Флаги	Смысл
<b>ja/jnbe</b>	CF or ZF=0	выше /не ниже и не равно
<b>jae/jnb</b>	CF=0	выше или равно/не ниже
<b>jb/jnae</b>	CF=1	ниже/не выше и не равно
<b>jbe/jna</b>	CF or ZF=1	ниже или равно/не выше
<b>je/jz</b>	ZF=1	равно/нуль
<b>jne/jnz</b>	ZF=0	не равно/не нуль
<b>jg/jnle</b>	(SF xor OF) or ZF=0	больше/не меньше и не равно
<b>jge/jnl</b>	SF xor OF=0	больше или равно/не меньше
<b>jl/jnge</b>	(SF xor OF)=1	меньше/не больше и не равно
<b>jle/jng</b>	((SF xor OF) or ZF)=1	меньше или равно/не больше
<b>jp/jpe</b>	PF=1	есть паритет/паритет четный
<b>jnp/jpo</b>	PF=0	нет паритета/паритет нечетный
<b>jc</b>	CF=1	перенос
<b>jnc</b>	CF=0	нет переноса
<b>jo</b>	OF=1	переполнение
<b>jno</b>	OF=0	нет переполнения
<b>jns</b>	SF=0	знак +
<b>js</b>	SF=1	знак -

Примечания: 1. термины “выше” и “ниже” применимы для сравнения без-  
знаковых величин (адресов);  
2. термины “больше” и “меньше” используются при учете  
знака числа;

3. слова **хог** и **ог** обозначают соответствующие логические операции.

### **3.3. Циклы**

Инструкция, организующая программный цикл имеет вид:

**loop**[<условие повторения цикла>] <метка короткого перехода>

Инструкция **loop** использует содержимое регистра CX как счетчик повторений цикла. Команда **loop** уменьшает содержимое регистра CX на 1 и передает управление по адресу, определяемому меткой перехода, если содержимое CX  $\neq 0$ , в противном случае выполняется следующая за LOOP инструкция. Подобно условным переходам инструкции этой группы могут осуществлять только короткие передачи управления, т.е. в пределах от -128 до +127.

Добавление к инструкции **loop** <условие повторения цикла> позволяет ввести дополнительные логические условия на повторение цикла:

**loope/loopz** – повторять, пока ноль;

**loopne/loopnz** – повторять, пока не ноль.

Проверка флага ZF осуществляется командой **loop**. Цикл повторяется, если содержимое CX  $\neq 0$  и выполняется соответствующее условие, в противном случае выполняется следующая за **loop** инструкция.

### **3.4. Пример выполнения работы**

Дан массив из десяти слов, содержащих целые числа. Требуется найти максимальное значение в массиве.

Текст программы:

```
data segment
max dw ?
mass dw 10,24,76,479,-347,281,-24,70,124,97
```

```

data ends
code segment
assume cs: code, ds: data
start: mov ax, data
        mov ds, ax      ; Загрузить сегментный адрес данных
        lea bx, mass     ; Загрузить адрес смещения массива
        mov cx, 10       ; Установить счетчик повторений цикла
        mov ax, [bx]     ; Первый элемент массива в Аккумулятор
beg:    cmp [bx], ax      ; Сравнить текущий элемент
; массива с максимальным
        jl no            ; он меньше
        mov ax, [bx]     ; он больше или равен
no:     inc bx            ; Следующий элемент
        inc bx            ; массива
        loop beg
        mov max, ax
quit:   mov ax, 4C00h     ; Код завершения 0
        int 21h          ; Выход в DOS

```

1. code ends
2. end start

### **3.5. Варианты заданий**

Дан массив из десяти знаковых чисел (слов или байт). Требуется:

1. Найти количество отрицательных чисел. Массив байт.
2. Найти сумму всех положительных и отрицательных чисел. Массив слов.
3. Найти сумму абсолютных величин. массив байт.
4. Найти количество положительных чисел. Массив байт.

5. Поменять местами пары соседних чисел. Массив слов.
6. Переставить числа в обратном порядке. Массив байт.
7. Заменить все отрицательные числа нулями. Массив байт.
8. Найти среднее арифметическое чисел. Массив слов.
9. Найти количество чисел больших  $10h$ . Массив слов.
10. Найти наименьшее по абсолютной величине числа. Массив байт.
11. Найти наибольшее отрицательное число. Массив байт.
12. Найти произведение положительных элементов последовательности. Массив слов.
13. Найти среднее арифметическое квадратов ненулевых элементов последовательности. Массив слов.
14. Найти полусумму наибольшего и наименьшего чисел. Массив байт.
15. Найти среднее арифметическое отрицательных элементов последовательности. Массив слов.
16. Найти сколько в массиве чисел больше  $12h$  и меньше  $0Afh$ . Массив байт.
17. Найти есть ли в массиве два нуля, идущих подряд. Массив слов.
18. Найти сумму абсолютных величин, меньших 6. Массив байт.
19. Найти среднее арифметическое чисел больших 10. Массив слов.
20. Найти сколько чисел равно  $12h$ . Массив байт.
21. Заменить все отрицательные числа их модулями. Массив байт.
22. Найти среднее арифметическое положительных чисел. Массив слов.
23. Найти количество чисел меньших  $10h$ . Массив байт.
24. Найти наименьшее среди положительных чисел. Массив слов.
25. Найти наибольшее отрицательное число. Массив байт.

### **3.6. Вопросы по теме**

1. Для чего нужен префикс ptr ?

2. В чем отличие команд `mov ax, offset mass` и `lea ax, mass`?
3. В чем отличие команд `mov ax, bx` и `mov ax, [bx]`?
4. В чем отличие команд `mov ax, [bp]` и `mov ax, [bx]`?
5. В чем отличие команд `mov ax, [bx+2]` и `mov ax [bx] + 2`?
6. В чем отличие команд `mov ax, [bx][si]` и `mov ax, [si][bx]`?
7. Какие существуют разновидности инструкции `jmp`?
8. Как организовать межсегментную передачу управления?
9. Напишите фрагмент программы условного перехода к метке, лежащей от самого перехода на расстоянии 257 байт.
10. Для организации каких вычислений служат команды `loop`, `loope`, `loopne`?
11. Модифицирует ли какие-нибудь регистры команда `loop`?
12. Можно ли организовать цикл по счетчику, не используя команды `loop`?
13. Можно ли организовать цикл `while` с помощью одной из команд `loop`?

## 4. Применение логических инструкций

Логические команды служат для сброса или установки отдельных бит в байте или слове. Они включают булевы операторы НЕ, И, ИЛИ, исключающее ИЛИ и операцию тестирования, которая устанавливает флаги, но не изменяет значения своих операндов.

### 4.1. Логические инструкции

**not** dst

Инструкция **not** инвертирует все биты байта или слова.

**and** dst, src

Инструкция **and** выполняет операции логическое И двух операндов (байтов или слов) и возвращает результат в операнд-приемник. Бит результата устанавливается в 1, если установлены в 1 оба соответствующих ему бита операндов, и устанавливаются в 0 противном случае.

**or** dst, src

Инструкция **or** выполняет операции логическое ИЛИ двух операторов (байтов или слов) и помещает результат на место операнда-приемника. Бит результата устанавливается в 1, если равен 1 хотя бы один из двух соответствующих ему битов операндов и устанавливается в 0 в противном случае.

**xor** dst, src

Инструкция **xor** выполняет операцию логическое исключающее ИЛИ двух операндов и помещает результат на место операнда-приемника. Бит результата устанавливается в 1, если соответствующие ему биты операндов имеют противоположные значения, и устанавливается в 0 в противном случае.

**test** dst, src

Инструкция **test** выполняет логическое И двух операндов (байтов или слов), модифицирует флаги, но результат не возвращает, т.е. операнды не изменяются.

В таблице 4.1. приведены значения регистра флагов, устанавливаемые логическими командами.

Таблица 4.1

Логические инструкции								
Мнемокод		Флаги						Действие
Код	Операнды	O	S	Z	A	P	C	
and	dst, src	0	x	x	u	x	0	логическое И
or	dst, src	0	x	x	u	x	0	логическое ИЛИ
xor	dst, src	0	x	x	u	x	0	логическое исключающее ИЛИ
not	Dst	-	-	-	-	-	-	логическое НЕТ
test	dst, src	0	x	x	u	x	0	логическое И без изменения dst

Примечание:

- флажок не модифицируется;
- x Устанавливается или сбрасывается в соответствии с результатом;
- u не определен;
- 0 Сбрасывается в 0.

#### 4.2. Примеры использования логических команд

1. Установить 3 и 0 биты в регистре **al**, остальные биты не изменять.  
or **al**, 00001001b
2. Сбросить 4 и 6 битвы в регистре **al**, остальные биты не изменять.  
and **al**, 10101111b
3. Инвертировать 2 и 4 биты в регистре **al**, остальные биты не изменять.



```
xor al, 00010100b
```

4. Перейти на метку LAB, если установлен 4 бит регистра al, в противном случае продолжить выполнение программы.

```
test al, 00010000b
```

```
jnz LAB
```

```
продолжаем
```

```
...
```

LAB:

5. Посчитать число единиц в регистре al, рассматривая байт, как набор бит.

```
mov cx, b          ; число сдвигов
```

```
xor bl, bl         ; обнуление BL
```

```
LL: shl al, 1      ; сдвиг влево на один разряд
```

```
jnc NO            ; переход, если нет переноса
```

```
inc bl            ; иначе увеличить BL
```

```
NO: loop LL       ; возврат, если cx ≠ 0
```

#### ***4.3. Пример выполнения работы***

Дан массив из 10 байт. Все байты имеют нулевые старшие биты. Необходимо каждый байт содержащий единицу в нулевом бите дополнить до четного числа единиц установкой седьмого бита.

Текст программы:

```
data segment
```

```
NB db 04h, 07h, 14h, 23h, 04h, 38h, 3Fh, 2Ah, 0Dh, 34h
```

```
data ends
```

```
code segment
```

```
assume cs: code, ds: data
```

```
START: mov ax, data
```

```

        mov ds, ax      ; Загрузить сегментный адрес данных
        lea bx, NB      ; bx-текущий адрес массива NB
        mov cx, 10      ; cx-счетчик числа итераций
BEG:    mov al, [bx]     ; считать очередной байт массива
        test al, 1b     ; установлен ли бит 0?
        jz BITOCLR      ; нет, бит 0 сброшен
        ; бит 0 установлен
        test al, 0ffh   ; четное число единиц?
        jp OK          ; да, больше ничего делать не надо
        or al, 80h      ; нечетное дополнить до четного?
        jmp short OK
        ; бит 0 сброшен
BITOCLR: test al, 0ffh   ; четное число единиц?
        jnp OK          ; нет, больше ничего делать не нужно
        or al, 80h      ; нечетное, дополнить до нечетного

OK:     mov [bx], al     ; записать измененный байт массива
        loop BEG
QUIT:   mov ax, 4c00h    ; Код завершения 0
        int 21h         ; Выход в DOS

code ends
end START

```

#### ***4.4. Варианты заданий***

1. Дан массив из 10 байт. Посчитать количество байт, в которых сброшены 6 и 4 бита.
2. Дан массив из 8 байт. Рассматривая его, как массив из 64 бит, посчитать количество единиц.

3. Дан массив из 8 байт. Рассматривая его как массив логических значений  $x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7$  (true-есть ненулевые биты в байте, false-все биты нулевые), вычислить логическую формулу  
$$f=(x_7 \& x_6 \& x_1) \vee (x_6 \& x_4 \& x_2 \& x_1 \& x_0) \vee (x_7 \& x_6 \& x_3 \& x_1).$$
4. Дан массив из 10 байт. Посчитать количество байт с числом единиц в байте равным три.
5. Рассматривая байт как набор логических значений  $x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0$  (true -1, false - 0), вычислить логическую формулу  
$$f=(x_7 \& x_6 \& x_3) \vee (x_6 \& x_4 \& x_2 \& x_1) \vee (x_7 \& x_6 \& x_2 \& x_0)$$
6. Дан массив из 8 байт. Рассматривая его, как массив из 64 бит посчитать длину самой длинной последовательности единиц.
7. Дан массив из 10 байт. Посчитать количество единиц во всех разрядах, кратных трём: 3, 6, 9, ..., 75, 78.
8. Дан массив из 5 байт. Рассматривая его как массив из 8 пятиразрядных слов, найти “исключающее или” всех 8 слов для выражения “10101”.
9. Дан массив из 6 байт. Рассматривая его, как массив из 48 бит, посчитать в нём количество нулей.
10. Дан массив из 8 байт. Рассматривая его, как массив из 64 бит, посчитать количество пар единиц в окружении нулей. Конец последовательности рассматривать как ноль.
11. Дан массив из 7 байт. Рассматривая его, как массив из восьми семибитных слов, посчитать количество слов с нечетным числом нулей в слове.
12. Дан массив из 9 байт. Рассматривая его как массив из 72 бит, посчитать число переходов между нулями и единицами.
13. Дан массив из 3 байт. Рассматривая его, как массив из 24 бит, посчитать количество одиночных единиц в окружении нулей. Конец последовательности рассматривать как ноль.

14. Дан массив из 6 байт. Посчитать количество байт число единиц, в которых не превышает 3.
15. Дан массив из 11 байт. Посчитать количество байт, в которых нет единиц, стоящих рядом.
16. Дан массив из 4 байт. Рассматривая его, как массив из 32 бит посчитать длину самой длинной последовательности нулей.
17. Дан массив из 6 байт. Посчитать количество единиц во всех разрядах, кратных пяти: 5, 10, ..., 45.
18. Дан массив из 3 байт. Рассматривая его как массив из 8 трёхразрядных слов, найти “исключающее или” всех 8 слов для выражения “101”.
19. Дан массив из 7 байт. Рассматривая его, как массив из 56 бит, посчитать в нём количество нулей, стоящих после единицы. Конец последовательности рассматривать как ноль.
20. Дан массив из 8 байт. Рассматривая его, как массив из 64 бит, посчитать количество пар единиц в окружении нулей. Конец последовательности рассматривать как ноль.
21. Дан массив из 5 байт. Рассматривая его, как массив из восьми пятибитных слов, посчитать количество слов с чётным числом единиц в слове.
22. Дан массив из 6 байт. Рассматривая его, как массив из 48 бит, посчитать число двух единиц, стоящих между нулями. Конец и начало последовательности рассматривать как нули.
23. Дан массив из 3 байт. Рассматривая его, как массив из 24 бит, посчитать количество одиночных единиц в окружении нулей. Конец последовательности рассматривать как ноль.
24. Дан массив из 6 байт. Посчитать количество байт, число единиц в которых не превышает 3.
25. Дан массив из 11 байт. Посчитать количество байт, в которых нет единиц, стоящих рядом.

#### **4.5. Вопросы по теме**

1. В чем отличие команд test и and?
2. Как сбросить 5-й бит переменной байта ВВ?
3. Как установить 5-й бит переменной байта ВВ?
4. Как инвертировать 5-й бит переменной байта ВВ?
5. Как проверить установлен ли 5-й бит переменной байта ВВ?
6. Как проверить четным или нечетным является количество установленных бит в байте?
7. Какие флаги условий модифицируются после выполнения команд and, or, xor ?
8. В чем основное отличие команд логических и арифметических сдвигов?
9. Укажите максимальное число двоичных разрядов, на которые можно сдвинуть операнд с помощью одной команды сдвига?

## 5. Обработка символьной информации с помощью функций DOS

Прерыванием (interrupt), подробнее см. раздел 7, называется способ общения центрального процессора с периферийными устройствами. Периферийное устройство (клавиатура, дисковод и др.) посылает запрос на установление сеанса передачи, процессор прерывает выполнение основной программы и переходит на выполнение программы обработки запроса от периферийного устройства. Эта программа, *драйвер устройства*, предварительно загружена в память (резидентная программа) и её адрес известен процессору. Такая обработка запросов называется аппаратными прерываниями.

### 5.1. Программные прерывания и системные вызовы

Операционная система MS-DOS, как известно, является одноканальной операционной системой, т.е. одновременно может исполнять только одну задачу. Вместе с тем имеется необходимость во время выполнения основной задачи производить некоторые вспомогательные действия. Подход, основанный на предварительной загрузке резидентных программ, использованный в аппаратных прерываниях, которые можно вызывать в момент выполнения основной программы, например, переключение раскладки клавиатуры или обращение к дисководу, оказался очень продуктивным, так как делает MS-DOS псевдомногоканальной системой.

Вызовы резидентных программ, адрес которых заранее известен процессору, стали называть, по аналогии с аппаратными прерываниями, программными прерываниями, хотя никаких

прерываний на самом деле не происходит. В систему команд процессора ввели команду вызова программного прерывания, которая вызывает соответствующую резидентную программу.

Команда вызова программного прерывания имеет вид

**int** <номер прерывания>

<номер прерывания> - число, обычно в шестнадцатеричное, в диапазоне 00h – 0FFh, определяет адрес вызываемой резидентной программы.

Некоторые резидентные программы, выполняющие низкоуровневое общение с периферийными устройствами записаны в ROM BIOS (Read Only Memory Base Input/Output System) и поставляются вместе с системной платой, например, учёт системного времени, форматирование секторов на дорожке диска и т.д., и не зависят от применяемой операционной системы.

Резидентные программы, использующие низкоуровневую систему резидентов BIOS и выполняющие более сложные задачи, например, файловые операции с диском, подгружаются в память при загрузке операционной системы. Их принято называть функциями операционной системы или системными вызовами.

Наибольшее число различных системных функций в MS-DOS сосредоточено в резидентной программе с номером прерывания 21h – диспетчер функций MS-DOS. В зависимости от значения, содержащегося при вызове прерывания в регистре ah, MS-DOS выполняет одну из нескольких десятков функций MS-DOS.

Все функции BIOS и DOS описаны в специальных справочниках с указанием для каждой функции набора входных и выходных параметров, передаваемых через регистры, а также перечнем возможных ошибок. В данной главе будут описаны функции прерывания 21h относящиеся к работе с клавиатурой и экраном ПЭВМ.

## **5.2. Описание функций работы с клавиатурой и дисплеем диспетчера функций MS-DOS**

Для вызова функции прерывания DOS 21h необходимо проделать следующие действия:

- выбрать функцию, выполняющую требуемые действия;
- занести номер функции в регистр ah;
- подготовить другие регистры (если это необходимо);
- написать команду `int 21h`;

прочитать результаты или состояние из регистров, указанных в описании данной функции.

Ниже следует описание некоторых функций 21H.

### **Функции 01H**

Выполняет ввод с клавиатуры одного символа и отображает его на экране.

- Вызов:  
`ah = 01h`
- Возвращаемое значение:  
`al` = код ASCII введенного символа
- Примечание. Введенный символ отображается на экране (выполняется эхо-отображение). Комбинация клавиш `Ctrl/C` (или `Ctrl/Break`) прекращает выполнение программ пользователя.

### **Функции 02H**

Выполняет отображение символа на стандартный вывод (дисплей).

- Вызов:  
`ah = 02h`  
`dl` = отображаемый символ



- Возвращаемое значение:

нет

- Примечание. Символ отображается на стандартный вывод. Комбинация клавиш Ctrl/C (или Ctrl/Break) прекращает выполнение программ пользователя.

### **Функция 05H**

Выполняет отображение символа на принтер.

- Вызов:

ah = 02h

dl = символ для принтера

- Возвращаемое значение:

нет

- Примечание. Символ отображается на принтер. Комбинация клавиш Ctrl/C (или Ctrl/Break) прекращает выполнение программ пользователя. Эта функция не возвращает ошибки состояния принтера.

### **Функция 07H**

Выполняет ввод с клавиатуры одного символа.

- Вызов:

ah=07h

- Возвращаемое значение:

al = код ASCII введенного символа

- Примечание. Введенный символ не отображается на экране (не выполняется эхо-отображения). Комбинация клавиш Ctrl/C (или Ctrl/Break) прекращает выполнение программы пользователя.

### **Функция 08H**

Выполняет ввод с клавиатуры одного символа.

- Вызов:  
ah=08h
- Возвращаемое значение:  
al = код ASCII введенного символа
- Примечание. Введенный символ не отображается на экране (не выполняется эхо-отображение). Комбинация клавиш Ctrl/C ( или Ctrl/Break) прекращает выполнение программы пользователя.

### **Функция 09H**

Выполняет отображение строки на стандартный вывод.

- Вызов:  
ah = 09H  
ds: dx=указатель на отображаемую строку  
Возвращаемое значение:  
нет
- Примечание: Строка отображается на стандартный вывод. \$ признак конца строки, \$ не отображается, dx содержит смещение строки, ds - сегментный адрес. Ниже приведены код управления курсором:  
0dh (13) - перевод курсора в начало текущей строки;  
0ah (10) - перевод курсора вниз на 1 строку;  
08h (8) - перевод влево на 1 позицию;  
07h (7) - звонок.

Пример.

Чтобы вывести на экран с новой строки текст: “Функция 09H для выдачи текста на экран” и затем перевести курсор в следующую строку, следует в сегменте данных описать строку:

beg db 0dh, 0ah, “Функция 09H для выдачи текста наэкран”, 0dh,0ah, “\$”

а в программном сегменте записать команды:

```
lea dx,beg      ; адрес строки в dx
mov ah,09h      ; номер функции в ah
int 21h         ; вызов функции
```

## Функция ОАН

Выполняет ввод с клавиатуры в буфер строки символов.

- **Вызов:**

ah= 0ah

ds: dx = адрес буфера ввода

- **Возвращаемое значение:**

Строка символов по указанному адресу

- **Примечание.** Читается со стандартного ввода. dx содержит смещение буфер вывода, DS - сегментный адрес. Буфер вывода имеет следующую структуру: 0-й байт содержит максимальное количество символов в буфере; 1-й байт содержит количество реально введенных символов; начиная со 2-го размещён буфер для ввода размером не менее указанного в 1-м байте. Выполняется эхо-отображение. Комбинация клавиш Ctrl/C (или Ctrl/Break) прекращает выполнение программы пользователя. Символы вводятся один за другим, до тех пор, пока не будет введен код ODh (код клавиш "Enter"), завершающий строку. В ходе ввода строки пользователь может редактировать строку, и, в частности, использовать "забой".

Пример.

Пусть требуется ввести строку длиной не более 10 символов. При этом в сегменте данных можно описать буфер, например, таким образом:

```
buffer      db 11      ; Нулевой байт буфера
```

```
entered      db (?)           ; Число введенных символов
string       db 11 dup (?)    ; Введенные символы
```

Сам ввод выполняется командами:

```
lea dx, buffer      ; Адрес буфера в dx
mov ah, 0ah          ; Номер функции в ah
int 21h              ; Вызов функции
```

### **Функция 0Bh**

Выполняет опрос состояния буфера клавиатуры.

- Вызов:  
ah = 0Bh
- Возвращаемое значение:  
al = 00h, если нет символа в буфере клавиатуры;  
al = ffh, если есть символ в буфере клавиатуры.
- Примечание. Устанавливает значение AL в зависимости от наличия символов в буфере клавиатуры. Часто используются в задачах, действующих при нажатии определенных клавиш. Комбинация клавиш Ctrl/ (или Ctrl/Break) прекращает выполнение программы пользователя.

### **5.3. Пример выполнения работы**

Ввести строку с клавиатуры, посчитать, сколько и каких десятичных цифр имеется во введенной строке, посчитанные значения вывести на терминал.

Текст программы:

```
data segment
COUNT db 10 dup (0)          ; счетчик количества цифр
CIFR db '0123456789ABCDEF'    ; таблица преобразования цифр
```

```

IN_STR db 80, ?, 82 dup (?)      ; буфер ввода
OUT_STR db 0Dh, 0Ah, ?, '-', ?, ?, '$' ; буфер вывода
data ends

code segment
    assume cs:code, ds:data
START:  mov ax, data
        mov ds, ax      ; Загрузить сегментный адрес данных
        ; Ввод строки
    lea dx, IN_STR
    mov ah, 0ah
    int 21h
        ; Обработка
        xor ah, ah      ; обнуление старшего байта AX
        lea bx, IN_STR+2 ; адрес начала введенной строки
        xor cx, cx
        mov cl, IN_STR+1 ; количество введенных символов
BB:     mov al, [BX]      ; очередной символ строки
        cmp al, '0'      ; код символа меньше чем код нуля?
        jb NC            ; да, не цифра
        cmp al, '9'      ; код символа больше чем код девяти
        ja NC            ; да, не цифра
        ; символ - десятичная цифра
        sub al, '0'      ; преобразуем ASCII код в число
        mov si, ax       ; индекс в массиве счетчиков COUNT
        inc COUNT[si]    ; увеличиваем счетчик цифр
NC:     inc bx           ; получить очередной символ строки
        loop BB
        ; Вывод результатов

```

```

mov cx, 10
    lea bx, CIFR      ; адрес таблицы преобразования цифр в ASCII
    xor si, si        ; номер выводимой цифры
OUT:   mov al, '0'
    add ax, si        ; ASCII код очередной цифры с номером в si
    mov OUT_STR+2, al ; в буфер вывода
    mov dl, COUNT[si] ; читать количество цифр с номером в si
mov al, dl
    push cx           ; временное сохранение cx
    mov cl, 4         ; сдвиг на четыре
    shr al, cl        ; выделить старшую цифру
    xlat              ; ASCII старшей цифры в al
    pop cx            ; восстановление cx
    mov OUT_STR+4, al ; в буфер вывода
    mov al, dl        ; восстановить количество цифр в al
    and al, 00001111b ; выделить младшую цифру
    xlat              ; ASCII младшей цифры в al
mov OUT_STR+5, al ; в буфер вывода
    lea dx, OUT_STR   ; подготовка к выводу строки
    mov ah, 09h       ; номер функции
    int 21h           ; вывод строки
    inc si            ; Счетчик очередной цифры
    loop OUT
QUIT:  mov ax, 4C00h   ; Код завершения 0
    int 21h           ; Выход в DOS
code ends
end START

```

#### **5.4. Варианты заданий**

1. Ввести с клавиатуры строку. Сжать строку, т.е. удалить пробелы и табуляции. Вывести результаты на экран.
2. Ввести с клавиатуры строку. Преобразовать все малые буквы в большие. Вывести результаты на экран.
3. Ввести с клавиатуры строку. Посчитать количество слов в строке. Определить, что является разделителем слов. Вывести результаты на экран.
4. Ввести с клавиатуры строку. Ввести с клавиатуры коротенькую строку - шаблон. Найти шаблон во введенной строке. Вывести на экран "ДА", если шаблон есть и "НЕТ", если нет.
5. Ввести с клавиатуры две строки. Сравнить их. Вывести на экран "ДА", если они равны и "НЕТ", если нет.
6. Ввести с клавиатуры строку. Если она длиннее некоторой заданной величины, то обрезать, если короче растянуть, вставив нужное число пробелов между словами. Вывести результаты на экран.
7. Ввести с клавиатуры строку, состоящую из нескольких слов. Вывести каждое слово на экран в отдельной строке, т.е. выдать слова в столбик.
8. Ввести с клавиатуры строку. Переставить в ней символы, поменяв местами первый символ с последним, второй с предпоследним и т.д. Вывести результаты на экран.
9. Ввести с клавиатуры две строки. Сравнить их. Вывести на экран номер начала второй строки в первой.
10. Ввести с клавиатуры строку, содержащую несколько точек. Преобразовать строку, чтобы после каждой точки был пробел, и следующая буква после точки была заглавная. Вывести результаты на экран.
11. Ввести с клавиатуры строку, содержащую несколько слов, разделенных пробелом. Переставить в ней слова, поменяв местами первое слово с последним, второе с предпоследним и т.д. Вывести результаты на экран.

12. Ввести с клавиатуры строку, состоящую из нескольких слов. Вывести каждое слово на экран в отдельной строке лесенкой, т.е. выдать каждое слово в столбик, с фиксированным сдвигом относительно начала предыдущего.
13. Ввести с клавиатуры строку. Преобразовать все буквы в числа. Построить криптограмму (вместо букв вывести на экран соответствующие им числа).
14. Ввести с клавиатуры строку, состоящую из нескольких букв. Заменить каждую букву в строке на другую букву, следующую за данной буквой по алфавиту. Вывести результаты на экран.
15. Ввести с клавиатуры строку и строку из двух чисел. Первое число указывает начало подстроки для ввода на экран, второе количество символов из первой строки, которое необходимо вывести на экран. Ввести с клавиатуры две строки. Сравнить их. Вывести на экран номер начала первой строки во второй. Ввести с клавиатуры строку. Ввести с клавиатуры коротенькую строку - шаблон. Найти шаблон во введенной строке. Вывести на экран "ДА", если шаблон есть и "НЕТ", если нет.
16. Ввести с клавиатуры две строки. Сравнить их. Вывести на экран какая из строк больше и насколько.
17. Ввести с клавиатуры строку и некоторое число. Если строка длиннее заданного числа, то обрезать, если короче растянуть, вставив нужное число пробелов между словами. Вывести результаты на экран.
18. Ввести с клавиатуры строку, состоящую из нескольких слов. Вывести каждое слово на экран в отдельной строке, со смещением влево на одно знакоместо по отношению к предыдущей строке.
19. Ввести с клавиатуры строку. Посчитать в ней количество запятых. Вывести результаты на экран.



20. Ввести с клавиатуры две строки. Вывести на экран все символы, которые содержатся в обеих строках.
21. Ввести с клавиатуры строку, содержащую несколько точек. Преобразовать строку, чтобы после каждой точки был пробел, и следующая буква после точки была заглавная. Вывести результаты на экран.
22. Ввести с клавиатуры строку, содержащую несколько слов, разделенных пробелом. Переставить в ней слова, поменяв местами первое слово с последним, второе с предпоследним и т.д. Вывести результаты на экран.
23. Ввести с клавиатуры строку, состоящую из нескольких слов. Вывести каждое слово на экран в отдельной строке лесенкой, т.е. выдать каждое слово в столбик, с заданным сдвигом относительно начала предыдущего.
24. Ввести с клавиатуры строку. Преобразовать все буквы в числа. Построить криптограмму (вместо букв вывести на экран соответствующие им числа).
25. Ввести с клавиатуры строку, состоящую из нескольких букв. Заменить каждую букву в строке на другую букву, следующую за данной буквой через заданное число символов по алфавиту. Вывести результаты на экран.

#### **5.5. Вопросы по теме**

1. Что такое программное прерывание?
2. Какие возможности работы с клавиатурой имеются у программиста?
3. Чем отличаются друг от друга различные функции DOS? выполняющие ввод с клавиатуры?
4. Как работает команда `xlat`?
5. Можно ли выдать на экран текст '\$1.00=25,00 rub/', используя функции DOS 09h?
6. Какие режимы адресации удобно использовать при работе с одномерными массивами?

7. Что означает выражения в поле операндов в строках примера

`lea bx, IN_STR+2`

`mov OUT_STR+3, al?`

8. Как выделить младшую тетраду байта?

9. Как выделить старшую тетраду байта?

10. В чем отличие команд

`lea BX, STR`

`mov BX, offset STR?`

## 6. Подпрограммы

Подпрограммы позволяют сократить объём текста программы, применять модульный принцип построения программ, использовать одни и те же подпрограммы в различных программах, что значительно сокращает время создания программ и уменьшает время отладки.

### 6.1. Структура подпрограммы

Описание подпрограммы в языке ассемблер имеет следующую структуру:

<имя процедуры> **proc** <тип процедуры>

...

операторы тела подпрограммы

...

**ret** [<выражение>]

<имя процедуры> **endp**

<тип процедуры> - определяет тип перехода: **near** (близкий), **far** (дальний). Если тип не задан, по умолчанию принимается **near**. Тип перехода **near** показывает, что тело процедуры описано в том же сегменте, что и её вызов. Тип перехода **far** обеспечивает вызов процедуры из других сегментов, с другим значением регистра CS. Такие процедуры обычно используются как отдельные объектные модули или в составе библиотек.

**ret** [<выражение>] - выполняет возврат из процедуры в вызывающую программу. В зависимости от типа процедуры, эта команда восстанавливает из стека значение IP (близкий вызов) или CS:IP (дальний вызов). Эта команда не обязана быть последней по тексту процедуры, но является последней по порядку выполнения. Значение <выражение> указывает размер стека в байтах, восстанавливаемого при возврате из процедуры. Вос-

становление стека необходимо производить при передаче параметров процедуры через стек. Так как работа со стеком выполняется словами, значение <выражение> всегда должно быть кратным двум.

Допускается вложение описания подпрограммы внутрь описания другой подпрограммы.

## **6.2. Вызов подпрограммы**

Вызов подпрограммы выполняется командой

**call** [<тип вызова> **ptr**] <адрес процедуры>

< тип вызова > - **near (word)** или **far (dword)**. Если тип не задан, по умолчанию принимается **near**.

< адрес процедуры > - имя или адрес процедуры. При ближнем вызове в стеке запоминается текущее значение регистра IP. При дальнем вызове в стеке запоминаются значения CS:IP.

Если тип вызова не указан явно, он определяется типом, на который указывает <адрес процедуры>, аналогично команде безусловного перехода **jmp**.

Пример.

Пусть в сегменте данных описаны переменные:

FADDR      dd    ?

NADDR      dw    ?

в сегменте кода описаны подпрограммы:

FPROC      **proc far**

..      .

FPROC      **endp**

NPROC      **proc**

. . .

**NPROC      endp**

Рассмотрим различные примеры команд вызова:

<b>call FPROC</b>	;дальний вызов п/п FPROC
<b>call FPROC</b>	;дальний вызов п/п FPROC
<b>call FADDR</b>	;дальний вызов п/п, чей адрес в FADDR
<b>call NADDR</b>	;ближний вызов п/п, чей адрес в NADDR
<b>call dx</b>	; ближний вызов п/п, чей адрес в DX
<b>call word ptr [BX]</b>	; косвенный ближний вызов п/п
<b>call dword ptr [BX]</b>	; косвенный дальний вызов п/п

### **6.3. Передача параметров**

Для передачи входных параметров в подпрограмму и выходных в программу существует несколько способов. Чаше всего передача параметров осуществляется через регистры или через стек.

При передаче через регистры перед вызовом подпрограммы параметры заносятся в регистры процессора, а после возврата вызывающая программа забирает из регистров значения результатов.

При передаче через стек, параметры перед вызовом подпрограммы заносятся в стек командой

**push    src**

Для обращения к параметрам, хранящимся в стеке, обычно используется регистр bp:

**mov bp, sp**

Необходимо помнить, что поверх параметров, передаваемых в подпрограмму, в стек записываются командой **call** одно или два слова адреса

возврата. Каждая процедура «знает» свой тип вызова (одно или два слова) и отступив от верхушки стека на +2 или на +4 читает параметры.

**mov** ax, bp+2 ; первый параметр при ближнем вызове

**mov** ax, bp+4 ; первый параметр при дальнем вызове

### **Сохранение регистров**

Подпрограмма во время выполнения использует регистры процессора. Значения, которые в них хранились, возможно, ещё понадобятся основной программе. По этой причине каждая подпрограмма обязана сохранить значения регистров перед началом их использования, а после завершения работы перед возвратом восстановить их прежние значения. Для сохранения регистров используется стек. В процессорах, начиная с 386, введены команды

pusha

popa

сохраняющие в стеке, а после выполнения подпрограммы, восстанавливающие значения всех регистров.

#### **6.4. Пример выполнения работы**

Разработать подпрограмму, которая удаляет, начиная с заданной позиции строки, указанного числа символов. Написать программу, которая вводит с клавиатуры строку, позицию и длину удаляемой части строки, удаляет указанную часть и выводит строку.

Текст программы:

data segment

MESS1 db 0dh,0ah, "Введите строку:", 0dh,0ah, "\$"

MESS2 db 0dh,0ah, "Введите позицию:", 0dh,0ah, "\$"

```

MESS3      db 0dh,0ah, "Введите число удаляемых символов:", 0dh, 0ah, "$"
MESS4      db 0dh,0ah, "Строка после удаления:", 0dh,0ah, "$"
S_BUFLLEN  db 80          ; Максимальная длина строки
S_FACTLEN  db  ?          ; Фактическая длина строки
S_INPBUF   db 80 dup(?)   ; Введённая строка
N_BUFLLEN  db 3           ; Максимальная длина числа при вводе
N_FACTLEN  db  ?          ; Фактическая длина числа
N_INPBUF   db 3 dup(?)    ; Введённое число
POSDEL     dw  ?          ; Позиция начала удаления
LENDEL     dw  ?          ; Число удаляемых символов
data       ends
code       segment
assume cs:code, ds:data

START:     mov ax, data
           mov ds,ax

           ; Ввод строки
LOP:       lea  DX, MESS1
           mov  ah, 09h
           int  21h        ;приглашение к вводу строки
           lea  DX, S_BUFLLEN
           mov  ah, 0Ah
           int  21h        ; ввод строки
           mov  AL, S_FACTLEN
           cmp  al, 0       ; строка пустая?
           ja   LLL0        ; нет продолжать
           jmp  quit        ; закончить работу
LLL0:      lea  bx, S_INPBUF ; получить адрес начала строки
           cbw              ; получить длину в слове

```

```

    add  bx, ax                ; адрес конца строки
    mov  byte ptr [bx], "$"   ; записать признак конца строки
; Ввод позиции удаления
LLL1:   lea  dx, MESS2
        mov  ah, 09h
        int  21h              ;Приглашение к вводу позиции удаления
        lea  dx, N_BUFLLEN
        mov  ah, 0Ah
        int  21h              ; ввод позиции удаления
        call VAL              ; вызов процедуры перевода в число
        jc   LLL1             ; ошибка, повторить ввод
        cmp  al, 0            ; ноль?
        jz   LLL1             ; ошибка, повторить ввод
        cmp  AL, S_FACTLEN ; превышает длину строки?
        jg   LLL1             ; ошибка, повторить ввод
        cbw                   ; расширить до слова
        mov  POSDEL, AX ; запомнить позицию удаления
; Ввод длины удаляемой части
LLL2:   lea  dx, MESS3
        mov  AH, 09h         ; приглашение к вводу числа
        int  21h              ; удаляемых символов
        lea  dx, N_BUFLLEN
        mov  ah, 0ah
        int  21h              ; ввод числа удаляемых символов
        call VAL              ; вызов процедуры перевода в число
        jc   LLL2             ; ошибка, повторить ввод
        cbw                   ; расширить до слова
        mov  LENDEL, AX ; запомнить число удаляемых символов

```



;Занести в стек параметры и вызвать подпрограмму удаления

```
lea    bx, S_INPBUF
mov    al, S_FACTLEN ; дополняем до слова
cbw
push    ax            ; 4-й параметр длина строки
push    LENDEL        ; 3й параметр число удаляемых симв.
push    POSDEL        ; 2-й параметр позиция удаления
push    bx            ; 1й параметр адрес строки
call    DELSUB        ; вызов подпрограммы
```

; Вывод результата

```
lea    dx, MESS4
mov    ah, 09h
int     21h           ; вывод заголовка вывода
lea    bx, S_FACTLEN
xor     cx, cx
mov     cl, S_FACTLEN
LLL3:   inc     bx
        cmp    byte ptr [bx], 0
        loopne LLL3 ; повторять до конца строки или первого нуля
LLL4:   mov     byte ptr [bx], "$"
        lea    DX, S_INPBUF
        mov    ah, 09h
        int     21h
        jmp    LOP
QUIT:   mov     ax, 4c00h
        int     21h
```

; Функция получения числа из его строкового представления

; Схема преобразования десятичного числа a2a1a0 в 16-ричную СС  
по ;схеме Горнера -  $N_{16} = (a_2 * A + a_1) * A + a_0$  , где А десятичное осно-  
вание

```

VAL    proc      near
        push bx          ; сохранение
        push cx          ; регистров
        push dx          ; в стеке
        lea  bx, N_INPBUF ; адрес начала числа
        mov  cl, N_FACTLEN ; фактическая длина числа
        xor  ch, ch       ; расширить до cx
        xor  ax, ax
        mov  dl, 10       ; основание системы счисления
VAL1:   imul  dl           ; умножаем на основание
        mov  dh, [bx]
        sub  dh, "0"      ; преобразуем его в цифру
        add  al, DH        ; добавляем к результату
        inc  bx           ; на следующий символ
        loop VAL1
        cmp  ax, 255
        clc              ; сброс флага CF
        jle  VAL2
        stc              ; если результат больше 255 установить флаг CF=1
VAL2:   pop  dx           ; восстановить
        pop  cx           ; регистры
        pop  bx           ; из стека
        ret
VAL    endp
; Подпрограмма удаления подстроки

```

; Параметры:

;адрес строки BP+2, позиция удаления BP+4, число удаляемых символов ; BP+6, длина строки BP+8

DELSUB     proc         near

    push bp

    mov bp, SP

    push es

    push ax

    push si

    push di

    push cx

    mov ax, ds

    mov es, ax

    mov di, [bp+4]             ; адрес начала строки

    add di, [bp+6]             ; адрес позиции удаления +1

    dec di                     ; адрес позиции удаления

    mov si, di

    add si, [bp+8]             ; адрес остающейся части строки

    mov cx, [bp+4]             ; адрес начала строки

    add cx, [bp+10]            ; адрес конца строки + "\$"

sub	cx, si	; число перемещаемых символов - 1
-----	--------	-----------------------------------

    inc cx                     ; число перемещаемых символов

    cld                         ; продвигаться от начала к концу

    rep movsb                 ; переслать (cx) символов

    pop bp

    pop cx                     ; восстановить

    pop di                     ; регистры

    pop si                     ; из

```

        pop     ax             ;стека
        pop     es
        ret     8              ; вернуться с очищением стека
DELSUB   endp
code     ends
end      START

```

### 6.5. Варианты заданий

В приведённых ниже вариантах заданий способ передачи параметров в процедуру выбирать произвольно. Зациклить программу по вводу строки, а признаком окончания работы считать ввод пустой строки.

1. Разработать подпрограмму, которая определяет, содержится ли одна заданная строка в другой заданной строке, и если да, то начиная с какой позиции. Разработать программу, которая вводит с клавиатуры две строки и сообщает содержится ли одна в другой и сколько раз.
2. Разработать две подпрограммы, одна из которых преобразует любую заданную букву в заглавную (в том числе для русских букв), а другая преобразует букву в строчную. Разработать программу, которая вводит с клавиатуры строку и замещает первые буквы всех слов заглавными, а остальные строчными буквами.
3. Разработать две подпрограммы, одна из которых соединяет две строки в одну, а другая обрезает строки до заданной длины (или дополняет пробелами, если длина строки меньше заданной). Разработать программу, которая вводит с клавиатуры число N, затем вводит несколько строк (конец ввода пустая строка) и формирует новую строку, состоящую из первых N символов каждой введённой строки.

4. Разработать две подпрограммы, одна из которых сравнивает две строки по лексикографическому порядку, а другая обменивает значения двух строк. Разработать программу, которая вводит с клавиатуры несколько строк (конец ввода пустая строка) и сортирует их в лексикографическом порядке.

5. Разработать подпрограмму, которая разбивает заданную строку на две части: первое слово (до первого пробела) и остальная часть строки (пробелы в начале строки убираются). Разработать программу, которая вводит с клавиатуры строку и выводит каждое слово с новой строки.

6. Разработать подпрограмму, которая переставляет символы заданной строки в обратном порядке. Разработать программу, которая вводит с клавиатуры строку и переставляет в обратном порядке символы в каждом слове.

7. Разработать подпрограмму, которая определяет, содержится ли одна заданная строка в другой заданной строке и, если да, то, начиная с какой позиции. Разработать программу, которая вводит с клавиатуры две строки и сообщает содержится ли одна в другой и сколько раз.

8. Разработать подпрограмму, которая подсчитывает, сколько раз заданный символ встречается в строке. Разработать программу, которая вводит с клавиатуры строку и число  $N$  и выдаёт список символов, которые встречаются в строке не менее  $N$  раз.

9. Разработать подпрограмму, которая преобразует заданное десятичное число в двоичную систему. Разработать программу, которая вводит с клавиатуры строку десятичных цифр и выводит на экран её эквивалент в двоичной системе. Если строка не является числом, то сообщает об этом.

10. Разработать подпрограмму, которая преобразует заданное шестнадцатеричное число в десятичную систему. Разработать программу, которая вводит с клавиатуры строку шестнадцатеричных цифр и выводит на

экран её эквивалент в десятичной системе. Если строка не является числом, то сообщает об этом.

11. Разработать подпрограмму, которая определяет, сколько раз заданная подстрока встречается в строке. Разработать программу, которая вводит с клавиатуры две строки и определяет сколько раз вторая строка встречается в первой.

12. Разработать подпрограмму, которая разбивает заданную строку на слова (слово считается от пробела до пробела, но в состав слова не входят). Разработать программу, которая вводит с клавиатуры строку и выводит каждое слово с новой строки.

13. Разработать две подпрограммы, одна из которых преобразует любую заданную букву в заглавную (в том числе для русских букв), а другая преобразует букву в строчную. Разработать программу, которая вводит с клавиатуры строку и заменяет первые буквы после точки на заглавные, а остальные на прописные.

14. Разработать подпрограмму, которая определяет, содержится ли одна строка в другой и если да то с какой позиции. Разработать программу, которая вводит с клавиатуры две строки и определяет содержится ли вторая строка в первой и с какой позиции.

15. Разработать подпрограмму, которая каждой букве ставит в соответствие число равное её порядковому номеру в алфавите. Разработать программу, которая вводит с клавиатуры строку и выводит через пробел числа, кодирующие введённую строку символов.

16. Разработать подпрограмму, которая определяет, содержится ли одна заданная строка в другой заданной строке, и если да, то начиная с какой позиции. Разработать программу, которая вводит с клавиатуры две строки и сообщает содержится ли одна в другой и сколько раз.

17. Разработать две подпрограммы, одна из которых преобразует любую заданную букву в заглавную (в том числе для русских букв), а другая преобразует букву в строчную. Разработать программу, которая вводит с клавиатуры строку и замещает первые буквы всех слов заглавными, а остальные строчными буквами.
18. Разработать две подпрограммы, одна из которых соединяет две строки в одну, а другая обрезает строки до заданной длины (или дополняет пробелами, если длина строки меньше заданной). Разработать программу, которая вводит с клавиатуры число N, затем вводит несколько строк (конец ввода пустая строка) и формирует новую строку, состоящую из первых N символов каждой введенной строки.
19. Разработать две подпрограммы, одна из которых сравнивает две строки по лексикографическому порядку, а другая обменивает значения двух строк. Разработать программу, которая вводит с клавиатуры несколько строк (конец ввода пустая строка) и сортирует их в лексикографическом порядке.
20. Разработать подпрограмму, которая разбивает заданную строку на две части: первое слов (до первого пробела) и остальная часть строки (пробелы в начале строки убираются). Разработать программу, которая вводит с клавиатуры строку и выводит каждое слово с новой строки.
21. Разработать подпрограмму, которая переставляет символы заданной строки в обратном порядке. Разработать программу, которая вводит с клавиатуры строку и переставляет в обратном порядке символы в каждом слове.
22. Разработать подпрограмму, которая определяет, содержится ли одна заданная строка в другой заданной строке и, если да, то, начиная с какой позиции. Разработать программу, которая вводит с клавиатуры две строки и сообщает содержится ли одна в другой и сколько раз.

23. Разработать подпрограмму, которая подсчитывает, сколько раз заданный символ встречается в строке. Разработать программу, которая вводит с клавиатуры строку и число N и выдаёт список символов, которые встречаются в строке не менее N раз.
24. Разработать подпрограмму, которая преобразует заданное десятичное число в двоичную систему. Разработать программу, которая вводит с клавиатуры строку десятичных цифр и выводит на экран её эквивалент в двоичной системе. Если строка не является числом, то сообщает об этом.
25. Разработать подпрограмму, которая преобразует заданное шестнадцатеричное число в десятичную систему. Разработать программу, которая вводит с клавиатуры строку шестнадцатеричных цифр и выводит на экран её эквивалент в десятичной системе. Если строка не является числом, то сообщает об этом.

#### **6.6. Вопросы по теме**

1. Что такое “ближние” и “дальние” подпрограммы?
2. Как определить “ближний” или “дальний” вариант команды **call** использован в программе?
3. Какие способы используются для передачи параметров в подпрограммы?
4. Может ли массив быть параметром подпрограммы?
5. Можно ли использовать для чтения из стека параметров регистр **sp** вместо **bp**?
6. Что означает операнд команды **ret**?
7. Какой последовательностью команд можно заменить команду “**ret 8**”?



## **7. Обработка прерываний**

В MS-DOS различают аппаратные и программные прерывания. Первые возникают по запросу периферийных устройств. Вторые позволяют использовать предоставляемые системой MS-DOS и BIOS большой набор подпрограмм, выполняющих различные полезные действия и оформленные как программные прерывания. В некоторых случаях стандартные программы обработки как аппаратных, так и программных прерываний могут не удовлетворять программиста. Поэтому появляется необходимость заменить существующую программу обработки прерывания иной или внести новые функции в уже имеющуюся программу.

### ***7.1. Обработка прерывания***

Рассмотрим подробно действия, которые называются обработкой прерывания. Эти действия выполняются независимо от того вызвано ли прерывание аппаратно или программно. При получении сигнала на прерывание (при аппаратном прерывании от программируемого контроллера прерываний, при программном командой процессора **int**) процессор содержимое CS, IP и регистра флагов сохраняет в стеке. В CS и IP помещается адрес подпрограммы обработки прерывания, которая и выполняется, после чего восстанавливаются из стека содержимое CS, IP и регистра флагов, и процессор продолжает выполнение программы.

Двойное слово, в котором хранится адрес подпрограммы обработки прерывания, называется вектором прерывания. Всего допустимо иметь 256 различных векторов прерываний. Для хранения векторов прерываний в DOS выделен первый килобайт памяти.

Адрес вектора прерывания с номером N вычисляется как  $N \cdot 4$ . В младшем слове хранится значение IP, а в старшем CS.

Пример. Определить адрес прерывания 21h. Получим  $21h \cdot 4h = 84h$ . Просмотр содержимого четырёх байтов, начиная с 0084h в сегменте 0000h

Показывает, что там храниться число

0000:0084 0DDE:048B,

где 0DDE – адрес сегмент, а 048B – адрес смещения подпрограммы обработки прерывания с номером 21h.

Если написать свою подпрограмму обработки прерывания и с помощью функций, описанных ниже, записать её адрес (сегмент и смещение) в вектор прерывания, то при вызове данного прерывания его обработка будет проходить по новой подпрограмме. Старый вектор прерывания, обычно, сохраняется и перед завершением программы восстанавливается.

Каждая подпрограмма обработки прерывания, в отличие от обычно подпрограммы, завершается командой **iret**, которая похожа на команду **ret**, но восстанавливает из стека кроме CS и IP ещё и регистр флагов.

## ***7.2. Изменение вектора прерывания***

MS-DOS предоставляет две функции 35h и 25h прерывания 21h для чтения и установки вектора прерывания.

### **Функция 35h**

Выполняет чтение адреса подпрограммы обработки прерывания.

- Вызов:  
ah = 35h  
al = номер прерывания
- Возвращаемое значение:

es:bx – указатель на подпрограмму прерывания

- Примечание. Функция получает адрес, указанного AL прерывания из таблицы векторов прерываний. BX содержит смещение, а ES сегмент адреса подпрограммы.

Пример.

```
mov ah, 35h    ; номер функции
mov al, 21h    ; номер прерывания
int 21h
```

В результате: (BX) = 048B, (ES) = 0DDE.

### **Функция 25h**

Выполняет занесение нового вектора прерывания.

- Вызов:

ah = 25h

al = номер прерывания

ds:dx – указатель на подпрограмму обработки прерывания

- Возвращаемое значение:

Нет

- Примечание. Функция устанавливает адрес прерывания, указанного в al в таблицу векторов прерываний. dx содержит смещение, а ds сегмент устанавливаемой подпрограммы.

## **7.3.Дополнительные сведения о структуре DOS и BIOS**

### **7.3.1.Прямое обращение к видеопамяти**

Видеопамять компьютера любой конфигурации расположена в адресном пространстве оперативного запоминающего устройства (ОЗУ). Это позволяет напрямую адресовать видеопамять одним из косвенных способов адресации памяти. Видеопамять занимает адреса с A000h по BFFFh, что со-

ставляет 128 Кбайт. Для увеличения объемов видеопамати (до 64 Мбайт), она делится на слои, так что по одному адресу находиться несколько ячеек, которые расположены в разных слоях. Обращение к видеопамати зависит от видеорежима, который определяет количество точек по горизонтали и вертикали, а так же количество битов, отводимую для хранения кода цвета каждой точки. Графическими режимами управляет видеоадаптер.

Более простым для программирования, допускающим простой доступ к видеопамяти, является символьный режим, который мы и рассмотрим подробнее. Для работы в символьном режиме отводится 16 Кбайт памяти, начиная с адреса B800h. Экран делится на 80 столбцов и 25 строк. Общее количество знакомест  $80 \times 25 = 2000$ . Для каждого знакоместа в видеопамати отводится два байта: чётный байт – ASCII код символа, нечётный – байт атрибутов. Счёт строк и колонок идёт из верхнего левого угла экрана, в байте b800h:0000h хранится символ выводимый в нулевой строке и нулевой колонке, в байте b800h:0001h хранится атрибут этого символа. В байте b800h:0002h хранится символ выводимый в нулевой строке и первой колонке, в байте b800h:0003h хранится атрибут этого символа и т.д.

Байт атрибутов имеет следующую структуру:

	Фон				Символ			
Атрибут	BL	R	G	B	I	R	G	B
Номер бита	7	6	5	4	3	2	1	0

BL – признак мерцания;	R – красный цвет;
G – зелёный цвет;	B – синий цвет;
I – Интенсивность свече-	
ния.	

Для доступа к видеопамяти в текстовом режиме можно использовать непосредственно один из сегментных регистров, например, ES:

```

mov ax, 0b800h      ; записать в регистр
mov es, ax           ; es адрес начала видеопамати
xor bx, bx           ; смещение символа от начала видеопамати
mov dh, 00010100b   ; атрибуты: на голубом фоне красный символ
mov dl, 65h          ; ASCII код символа
mov word ptr es:[bx], dx ; запись в видеопамать символа и атрибута
inc bx              ; смещение для
inc bx              ; следующего символа

```

Другой способ доступа – размещение сегмента данных фиксировано, в области видеопамати директивой AT.

```

video segment AT 0b800h
CHAR_ATTRIB db 4000 dup(?)
video ends
code segment
assume cs: code, ds:video
START: mov ax, video
      mov ds, ax
      xor si, si
      mov dl, byte ptr CHAR_ATTRIB[si] ; чтение символа
      inc si
      mov dh, byte ptr CHAR_ATTRIB[si] ; чтение атрибута
      . . .

```

### 7.3.2. Буфер клавиатуры

В ОЗУ, начиная с адреса 0040h, находится область данных BIOS, в которой хранится различная системная информация: адреса портов уст-

ройств, количество тиков таймера, начиная с полуночи и т.д. В частности, в этой области организован буфер клавиатуры, который способен принять до 15 символов от клавиатуры. Исполняемая программа должна опрашивать этот буфер и извлекать из него символы, иначе буфер переполнится, о чём свидетельствует писк встроенного динамика при нажатии очередной клавиши.

Буфер организован как круговой массив, т.е. при достижении конца буфера, он, если возможно начинает заполняться сначала. Два указателя: "голова" и "хвост" определяют состояние буфера. "Голова" показывает на первую свободную ячейку буфера, куда записывается очередной символ, поступивший от клавиатуры. "Хвост" показывает символ, который будет прочитан исполняемой программой при очередном обращении к буферу. Если буфер пуст, оба указателя показывают на одну и ту же ячейку буфера. Адрес "головы" буфера находится по адресу word ptr 0040h:001ah, а адрес "хвоста" по адресу word ptr 0040h:001ch. Если содержимое обоих указателей совпадает, значит клавиши нажаты не были. Эту проверку можно использовать для вызова в программе функции, читающей символ из буфера клавиатуры.

#### ***7.4. Пример выполнения работы***

Написать программу на ассемблере, выводящую в текущее положение курсора символ @. Следующий символ @ выводить в позицию выше, ниже, левее или правее текущего символа, в зависимости от нажатия клавиш "8", "2", "4", "6" на цифровой клавиатуре. Вывод осуществлять непрерывно с некоторой задержкой. Нажатие клавиши "0" завершает выполнение программы.

- Примечание. В программе необходимо вести отсчёт времени для задержки вывода символа @. Для этого необходимо изменить подпро-

грамму обработки прерывания от таймера 08h. Так как эта подпрограмма выполняет важные операции по управлению компьютером, для получения временного интервала используется прерывание 1Ch. Это прерывание вызывается из подпрограммы обработки прерывания 08h и содержит только команду **iret**. Предназначено оно специально для пользовательских программ, которым необходимо следить за интервалами отсчёта таймера.

#### Текст программы

```
data    segment
DIRECT  db    1      ; направление перемещения
EXIT    db    0      ; признак завершения программы (не 0)
SYM      db    "@"    ; символ, выводимый на экран
ATRIBUT1 db    14    ; атрибут символа (жёлтый)
ATRIBUT2 db    10    ; атрибут символа (зелёный)
POS      dw    3840   ; позиция начального вывода символа
OLD_CS   dw    ?     ; адрес сегмента старого вектора 1Ch
OLD_IP   dw    ?     ; адрес смещения старого вектора 1Ch
data    ends

code    segment
        assume cs:code, ds:data
        ; Подпрограмма обработки прерывания 1Ch
NEW_1C  proc      far
        push     ax      ; сохранить все регистры
        push     bx
        push     cx
        push     dx
        push     ds
```

```

push     es
mov  ax, DATA      ; установить ds на сегмент данных
mov  ds, ax         ; основной программы
mov  ax, 40h        ; установить es на
mov  es, ax         ; сегмент данных bios
mov  ax, es:[1ch]
mov  bx, es:[1ah]
cmp  bx , ax
jne  m5
jmp  back
m5:   mov  al, es:[bx]
      mov  es:[1ch], bx
      cmp  al, 30h
      jnz  m1
      mov  EXIT, 1
      jmp  back
m1:   cmp  al, 35h
      jne  m6
      mov  dl, ATRIBUT1
      mov  dh, ATRIBUT2
      mov  ATRIBUT1, dh
      mov  ATRIBUT2, dl
      jmp  back
m6:   cmp  al, 38h      ; стрелка вверх
      jz   m2
      cmp  al, 32h      ; стрелка вниз
      jz   m3
      cmp  al, 34h      ; стрелка влево

```



```

        jz     m4
        cmp    al, 36h          ; стрелка вправо
        jnz    back            ; неиспользуемая клавиша
        mov    DIRECT, 3
        jmp    back
m2:      mov    DIRECT, 1
        jmp    back
m3:      mov    DIRECT, 4
        jmp    back
m4:      mov    DIRECT, 2
back:    pop    es
        pop    ds
        pop    dx
        pop    cx
        pop    bx
        pop    ax
        iret
NEW_1C   endp
        ; Подпрограмма очистки экрана
CLS      proc      near
        push    cx
        push    ax
        push    si
        xor     si, si
        mov     ah, 7
        mov     dl, ''
        mov     cx, 2000
CL1:     mov     es:[si], ax

```

```

    inc    si
    inc    si
    loop   CL1
    pop    si
    pop    ax
    pop    cx
    ret

```

CLS endp

; Подпрограмма задержки

DELAY proc near

```

    push cx
    mov cx, 100

```

d12: push cx

```

    xor cx, cx

```

d11: nop

```

    loop d11

```

```

    pop cx

```

```

    loop d12

```

```

    pop cx

```

```

    ret

```

DELAY endp

; Подпрограмма вывода символа с заданным атрибутом

OUT\_SYMBOL proc near

```

    push ax

```

```

    push bx

```

```

    mov al, SYM

```

```

    mov ah, ATRIBUT1

```

```

    mov bx, POS

```

```

        call    DELAY
        mov     es:[bx], ax
        pop     bx
        pop     ax
        ret
OUT_SYMBOL endp
        ; Основная программа
START:   mov     ax, DATA
        mov     ds, ax
        ; чтение вектора прерывания
        mov     ah, 35h
        mov     al, 1Ch
        int     21h
        mov     OLD_IP, bx
        mov     OLD_CS, es
        ; установка вектора прерывания
        push    ds
        mov     dx, offset NEW_1C
        mov     ax, seg NEW_1C
        mov     ds, ax
        mov     ah, 25h
        mov     al, 1Ch
        int     21h
        pop     ds
        mov     ax, 0B800h
        mov     es, ax
        call    CLS
        call    DELAY

```

```

p1:      cmp  EXIT, 0
        jne  quit
        cmp  DIRECT, 1
        jz   p2
        cmp  DIRECT, 2
        jz   p3
        cmp  DIRECT, 3
        jz   p4
        mov  ax, POS
        add  ax, 160
        cmp  ax, 3999
        jg   p1
        mov  POS, ax
        call OUT_SYMBOL
        jmp  p1
p2:      mov  ax, POS
        sub  ax, 160
        jl   p1
        mov  POS, ax
        call OUT_SYMBOL
        jmp  p1
p3:      mov  ax, POS
        sub  ax, 2
        jl   p1
        mov  POS, ax
        call OUT_SYMBOL
        jmp  p1
p4:      mov  ax, POS

```

```

        add  ax, 2
        jg   p1
        mov  POS, ax
        call OUT_SYMBOL
        jmp  p1
quit:    call CLS
        mov  dx, OLD_IP
        mov  ax, OLD_CS
        mov  ds, ax
        mov  ah, 25h
        mov  al, 1Ch
        int  21h
        mov  ax, 4c00h
        int  21h
CODE     ends
        end  START

```

### 7.5. Варианты заданий

Во всех вариантах задания завершение программы осуществляется при вводе цифры 0.

1. Выводить последовательно цифры от 0 до 9 в одно место экрана. При вводе с клавиатуры какой-либо цифры менять темп вывода. Значение задержки между выводом очередного символа определять следующим способом: введённую цифру умножить на  $2^9$ , это и будет число повторений цикла задержки. Для анализа нажатия клавиши использовать вектор 1Ch.
2. Выводить в одно место экрана поочерёдно код пробела и код какого-нибудь символа. Задержка между выводом каждого символа определяется нажатием цифровой клавиши следующим способом: введённую цифру ум-

ножить на  $2^9$ , это и будет число повторений цикла задержки. Для анализа нажатия клавиши использовать вектор 1Ch.

3. Выводить в одно место экрана введенный символ до тех пор пока не будет введен другой символ. Менять при выводе атрибут символа циклически от 1 до 15. Для анализа нажатия клавиши использовать вектор 1Ch.

4. Выводить в текущее положение курсора символ #. Следующий символ # выводить в позицию выше, ниже, левее или правее текущего символа, в зависимости от нажатия клавиш "8", "2", "4", "6" на цифровой клавиатуре. Вывод осуществлять непрерывно с некоторой задержкой. Задержка между выводом каждого символа определяется нажатием цифровой клавиши, следующим способом: введенную цифру умножить на  $2^9$ , это и будет число повторений цикла задержки. Для анализа нажатия клавиши использовать вектор 1Ch.

5. Выводить в текущее положение курсора символ, введенный с клавиатуры. Этот же символ выводить в позицию выше, ниже, левее или правее текущего символа, в зависимости от нажатия клавиш "8", "2", "4", "6" на цифровой клавиатуре. С клавиатуры можно ввести любую латинскую букву, при этом, выводимый символ изменяется на введенный символ. Вывод осуществлять непрерывно с некоторой задержкой. Задержка между выводом каждого символа определяется нажатием цифровой клавиши, следующим способом: введенную цифру умножить на  $2^9$ , это и будет число повторений цикла задержки. Для анализа нажатия клавиши использовать вектор 1Ch.

6. В программе имеются два циклических счётчика, считающих от 0 до 23 и от 0 до 79. Их значение определяет соответственно строку и столбец для вывода символа на экран. При нажатии какойлибо клавиши на экран выводится символ % в положение, определяемое состоянием счётчиков на момент вывода. Для анализа нажатия клавиши использовать вектор 1Ch.

7. В программе имеется циклический счётчик, считающий от 1 до 6. При нажатии любой клавиши содержимое счётчика преобразуется в ASCII код и выводится в определённое место экрана, после чего счётчик продолжает считать. Для анализа нажатия клавиши использовать вектор 1Ch.
8. Посчитать за какое время процессор выполнить 1 000 000 команд `mov DI, SI; add DI, SI; mul SI`. Для подсчёта времени использовать вектор 1Ch. Выводить на экран преобразованные в ASCII коды число тиков таймера, затраченное на операцию.
9. Очистить экран. Вывести несколько строк произвольного текста (атрибут 14). Перехватив прерывание печати экрана Print Screen (Int 5h), менять атрибуты всех строк экрана циклически от 1 до 15. Каждое нажатие клавиши Print Screen вызывает изменение атрибута.
10. Выводить каждую секунду в правом верхнем углу экрана системное время “часы:минуты:секунды”.
11. Вывести несколько строк произвольного текста, содержащие лишь латинские буквы. Каждые 10 секунд заглавные буквы сменяются строчными и т. д.
12. В программе имеется циклический счётчик, считающий от 00h до FFh. Его значение преобразуется в ASCII код и выводится в левом верхнем углу экрана через 18 тиков таймера. При нажатии клавиши ‘2’ время вывода уменьшается вдвое, а при повторном нажатии время вывода увеличивается в два раза. Для анализа нажатия клавиши и подсчёта числа тиков таймера использовать вектор 1Ch.
13. Заполнить экран произвольной информацией. Перехватить прерывание 1Ch, по нажатию клавиши ‘1’ осуществить горизонтальный скроллинг всего экрана влево на один столбец, при нажатии клавиши ‘2’ скроллинг вправо на один столбец.

14. Очистить экран. Вывести несколько строк произвольного текста. Перехватить прерывание экрана (Int 5h). Первый вызов этого прерывания располагает строки вертикально, следующий «нормально» и т.д.
15. Очистить экран. Заполнить его произвольной информацией. Перехватить прерывание экрана (Int 5h). Первый вызов этого прерывания переносит строки верхней половины экрана на место нижних, а нижние на место верхних. Следующий вызов прерывания снова меняет их местами и т.д.
16. Выводить последовательно цифры от 0 до 9 в одно место экрана. При вводе с клавиатуры какой-либо цифры менять темп вывода. Значение задержки между выводом очередного символа определять следующим способом: введённую цифру умножить на  $2^9$ , это и будет число повторений цикла задержки. Для анализа нажатия клавиши использовать вектор 1Ch.
17. Выводить в одно место экрана поочерёдно код пробела и код какого-нибудь символа. Задержка между выводом каждого символа определяется нажатием цифровой клавиши следующим способом: введённую цифру умножить на  $2^9$ , это и будет число повторений цикла задержки. Для анализа нажатия клавиши использовать вектор 1Ch.
18. Выводить в одно место экрана введённый символ до тех пор пока не будет введён другой символ. Менять при выводе атрибут символа циклически от 1 до 15. Для анализа нажатия клавиши использовать вектор 1Ch.
19. Выводить в текущее положение курсора символ #. Следующий символ # выводить в позицию выше, ниже, левее или правее текущего символа, в зависимости от нажатия клавиш "8", "2", "4", "6" на цифровой клавиатуре. Вывод осуществлять непрерывно с некоторой задержкой. Задержка между выводом каждого символа определяется нажатием цифровой клавиши, следующим способом: введённую цифру умножить на  $2^9$ , это и будет число повторений цикла задержки. Для анализа нажатия клавиши использовать вектор 1Ch.



20. Выводить в текущее положение курсора символ, введенный с клавиатуры. Этот же символ выводить в позицию выше, ниже, левее или правее текущего символа, в зависимости от нажатия клавиш “8”, “2”, “4”, “6” на цифровой клавиатуре. С клавиатуры можно ввести любую латинскую букву, при этом, выводимый символ изменяется на введенный символ. Вывод осуществлять непрерывно с некоторой задержкой. Задержка между выводом каждого символа определяется нажатием цифровой клавиши, следующим способом: введенную цифру умножить на  $2^9$ , это и будет число повторений цикла задержки. Для анализа нажатия клавиши использовать вектор 1Ch.
21. В программе имеются два циклических счётчика, считающих от 0 до 23 и от 0 до 79. Их значение определяет соответственно строку и столбец для вывода символа на экран. При нажатии какой-либо клавиши на экран выводится символ % в положение, определяемое состоянием счётчиков на момент вывода. Для анализа нажатия клавиши использовать вектор 1Ch.
22. В программе имеется циклический счётчик, считающий от 1 до 6. При нажатии любой клавиши содержимое счётчика преобразуется в ASCII код и выводится в определённое место экрана, после чего счётчик продолжает считать. Для анализа нажатия клавиши использовать вектор 1Ch.
23. Посчитать за какое время процессор выполнить 1 000 000 команд `mov DI, SI; add DI, SI; mul SI`. Для подсчёта времени использовать вектор 1Ch. Выводить на экран преобразованные в ASCII коды число тиков таймера, затраченное на операцию.
24. Очистить экран. Вывести несколько строк произвольного текста (атрибут 14). Перехватив прерывание печати экрана `Print Screen (Int 5h)`, менять атрибуты всех строк экрана циклически от 1 до 15. Каждое нажатие клавиши `Print Screen` вызывает изменение атрибута.

25. Выводить каждую секунду в правом верхнем углу экрана системное время “часы:минуты:секунды” .

#### **7.6. Вопросы по теме**

1. В чём суть концепции прерывания?
2. Как работает система прерывания по вектору?
3. В чём отличие команд **ret** и **iret**?
4. Какие способы получения/изменения вектора прерывания Вы знаете?
5. Как вызвать программное прерывание?
6. Какие существуют способы передачи параметров в подпрограмму обработки прерываний и возврата параметров из неё?
7. Какие действия производит процессор при получении запроса на прерывание?
8. В чём отличие команд **call** и **int**?