

МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

**ЛАБОРАТОРНАЯ РАБОТА №1**

**ОТЧЁТ**

студента 2 курса 251 группы  
направления 09.03.04 — Программная инженерия  
факультета КНиИТ  
Григорьева Даниила Евгеньевича

Проверено:

Старший преподаватель

\_\_\_\_\_

Е. М. Черноусова

## СОДЕРЖАНИЕ

1 Задание №1 .....	3
1.1 Текст задания .....	3
1.2 Тексты программ .....	3
1.3 Скриншоты запуска программ .....	5
1.4 Тексты 2-х командных файлов (для exe-программ и для com- программы) .....	6
2 Задание №2 .....	7
2.1 Текст задания .....	7
2.2 Таблицы трассировки программ .....	7
3 Ответы на контрольные вопросы .....	8

## 1 Задание №1

### 1.1 Текст задания

Измените программы из примеров 1, 2 и 3 так, чтобы они выводили на экран ваши фамилию, имя и номер группы. Используя командные файлы (с расширением bat), подготовьте к выполнению и запустите 3 программы. Убедитесь, что они выводят на экран нужный текст и успешно завершаются.

### 1.2 Тексты программ

```
stack segment stack 'stack'      ;Начало сегмента стека
db 256 dup (?)                    ;Резервируем 256 байт для стека
stack ends                        ;Конец сегмента стека
data segment 'data'              ;Начало сегмента данных
Names db 'Grogoriev Danila, 251$' ;Строка для вывода
data ends                        ;Конец сегмента данных
code segment 'code'              ;Начало сегмента кода
assume CS:code,DS:data,SS:stack  ;Сегментный регистр CS будет
указывать на сегмент команд,    ;регистр DS - на сегмент данных, SS
                                  ;на стек
start:                            ;Точка входа в программу start
;Обязательная инициализация регистра DS в начале программы
mov AX,data                       ;Адрес сегмента данных сначала
загрузим в AX,
mov DS,AX                        ;а затем перенесем из AX в DS
mov AH,09h                       ;Функция DOS 9h вывода на экран
mov DX,offset Names              ;Адрес начала строки записывается в
регистр DX
int 21h                          ;Вызов функции DOS
mov AX,4C00h                     ;Функция 4Ch завершения программы с
кодом возврата 0
int 21h                          ;Вызов функции DOS
code ends                        ;Конец сегмента кода
end start                        ;Конец текста программы с точкой
входа
```

Текст программы №1

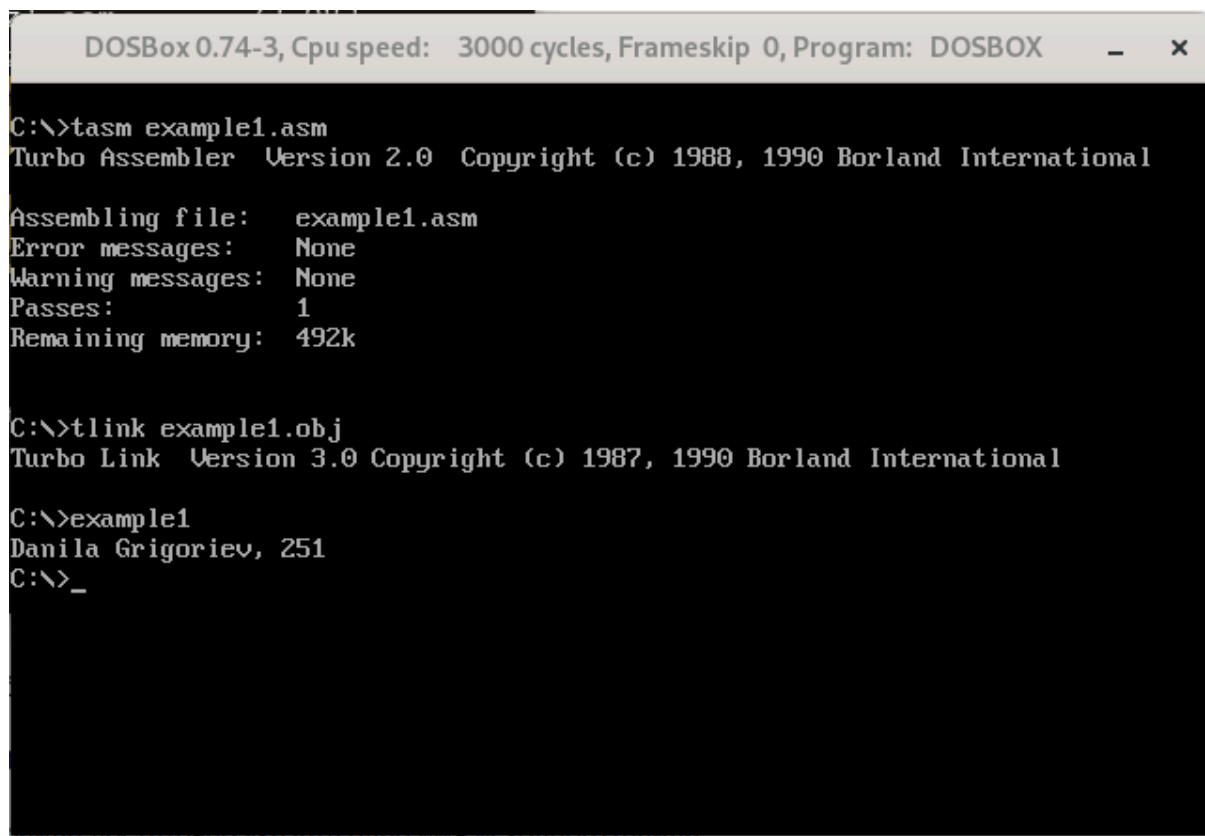
<code>.model small</code>	;Модель памяти SMALL использует
сегменты	
	;размером не более 64Kб
<code>.stack 100h</code>	;Сегмент стека размером 100h (256
байт)	
<code>.data</code>	;Начало сегмента данных
<code>Names db 'Danila Grigoriev, 251\$'</code>	
<code>.code</code>	;Начало сегмента кода
<code>start:</code>	;Точка входа в программу start
	;Предопределенная метка @data
обозначает	
	;адрес сегмента данных в момент
запуска программы,	
<code>mov AX, @data</code>	;который сначала загрузим в AX,
<code>mov DS, AX</code>	;а затем перенесем из AX в DS
<code>mov AH, 09h</code>	
<code>mov DX, offset Names</code>	
<code>int 21h</code>	
<code>mov AX, 4C00h</code>	
<code>int 21h</code>	
<code>end start</code>	

#### Текст программы №2

<code>.model tiny</code>	;Модель памяти TINY, в которой код,
данные и стек	
	;размещаются в одном и том же сегменте
размером до 64Kб	
<code>.code</code>	;Начало сегмента кода
<code>org 100h</code>	;Устанавливает значение программного
счетчика в 100h	
	;Начало необходимое для COM-программы,
	;которая загружается в память с адреса
<code>PSP:100h</code>	
<code>start:</code>	
<code>mov AH, 09h</code>	
<code>mov DX, offset Names</code>	
<code>int 21h</code>	
<code>mov AX, 4C00h</code>	
<code>int 21h</code>	
<code>;==== Data =====</code>	
<code>Names db 'Grigoriev Danila, 251\$'</code>	
<code>end start</code>	

#### Текст программы №3

### 1.3 Скриншоты запуска программ



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

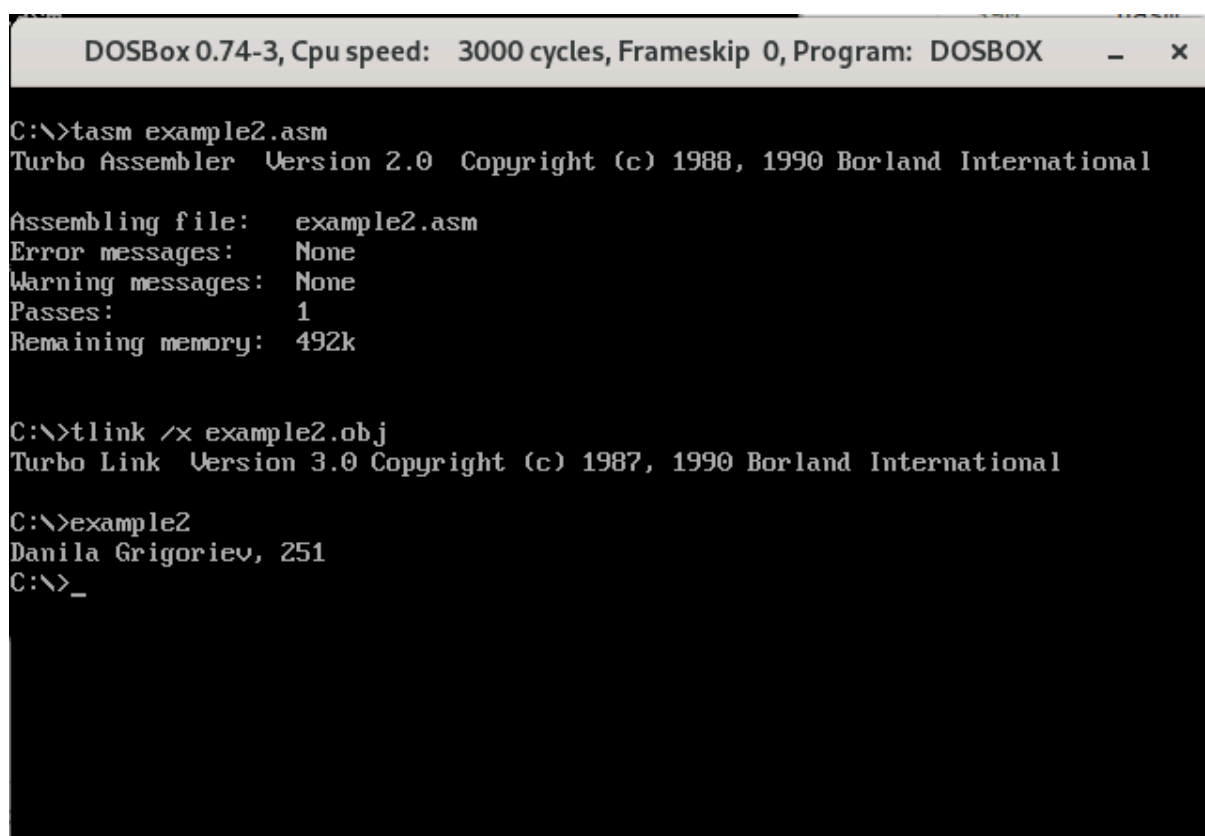
C:\>tasm example1.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International

Assembling file:    example1.asm
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   492k

C:\>tlink example1.obj
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland International

C:\>example1
Danila Grigoriev, 251
C:\>_
```

Запуск программы №1



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

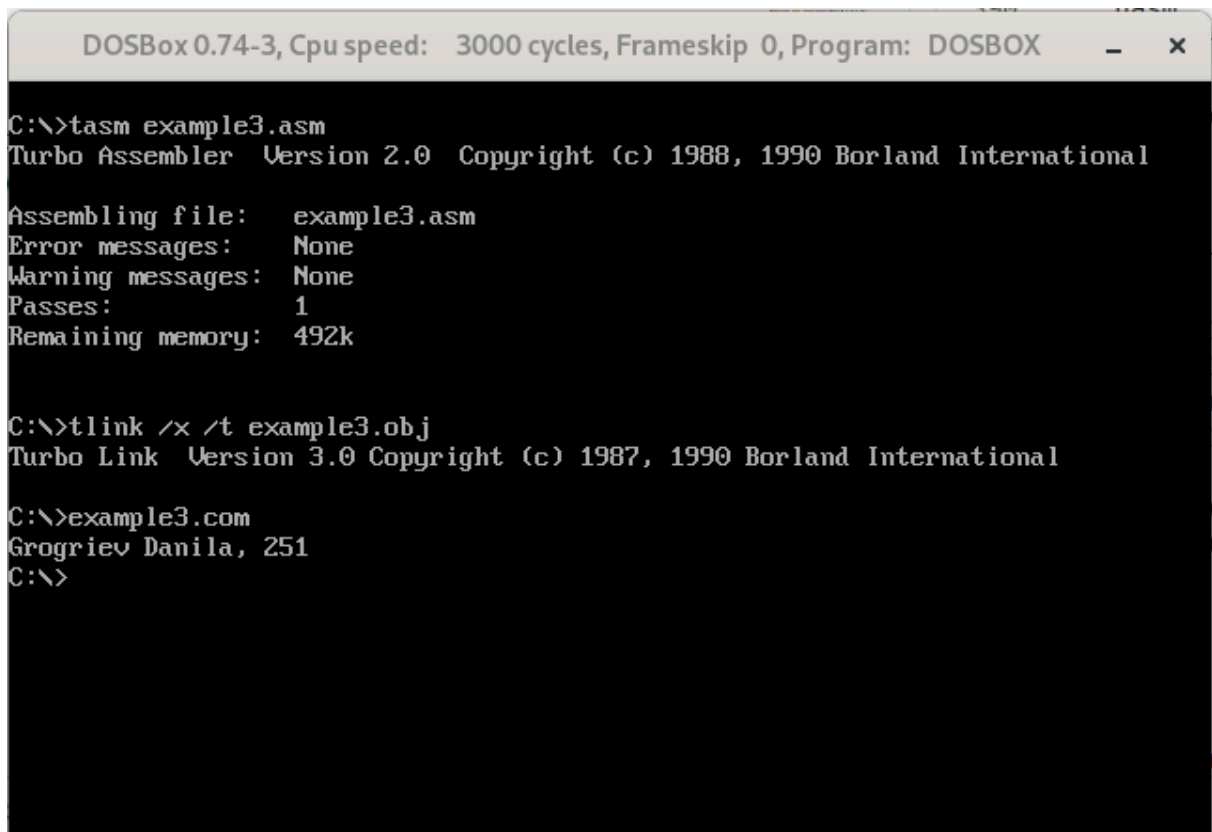
C:\>tasm example2.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International

Assembling file:    example2.asm
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   492k

C:\>tlink /x example2.obj
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland International

C:\>example2
Danila Grigoriev, 251
C:\>_
```

## Запуск программы №2



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\>tasm example3.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International

Assembling file:    example3.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  492k

C:\>tlink /x /t example3.obj
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland International

C:\>example3.com
Grogriev Danila, 251
C:\>
```

## Запуск программы №3

### 1.4 Тексты 2-х командных файлов (для exe-программ и для com-программы)

```
cls
tasm.exe %1.asm
tlink.exe /x %1.obj
%1
```

Текст командного файла для exe-программ

```
cls
tasm.exe %1.asm
tlink.exe /x /t %1.obj
%1
```

Текст командного файла для com-программ

## 2 Задание №2

### 2.1 Текст задания

Заполните таблицы трассировки для 3-х программ.

### 2.2 Таблицы трассировки программ

Шаг	Машинный код	Команда	Регистры									Флаги
			AX	BX	CX	DX	SP	DS	SS	CS	IP	
1	B8BD48	mov ax, 48BD	0000	0000	0000	0000	0100	489D	48AD	48BF	0000	00000010
2	8ED8	mov ds, ax	48BD	0000	0000	0000	0100	489D	48AD	48BF	0003	00000010
3	B409	mov ah, 09	48BD	0000	0000	0000	0100	489D	48AD	48BF	0005	00000010
4	BA0000	mov dx, 0000	09BD	0000	0000	0000	0100	489D	48AD	48BF	0007	00000010
5	CD21	int 21	09BD	0000	0000	0000	0100	489D	48AD	48BF	000A	00000010
6	B8004C	mov ax, 4C00	09BD	0000	0000	0000	0100	489D	48AD	48BF	000C	00000010
7	CD21	int 21	4C00	0000	0000	0000	0100	489D	48AD	48BF	000F	00000010

Таблица 1. Трассировка программы №1

Шаг	Машинный код	Команда	Регистры									Флаги
			AX	BX	CX	DX	SP	DS	SS	CS	IP	
1	B8BD48	mov ax, 48BD	0000	0000	0000	0000	0100	489D	48AD	48BF	0000	00000010
2	8ED8	mov ds, ax	48BD	0000	0000	0000	0100	489D	48AD	48BF	0003	00000010
3	B409	mov ah, 09	48BD	0000	0000	0000	0100	489D	48AD	48BF	0005	00000010
4	BA0000	mov dx, 0000	09BD	0000	0000	0000	0100	489D	48AD	48BF	0007	00000010
5	CD21	int 21	09BD	0000	0000	0000	0100	489D	48AD	48BF	000A	00000010
6	B8004C	mov ax, 4C00	09BD	0000	0000	0000	0100	489D	48AD	48BF	000C	00000010
7	CD21	int 21	4C00	0000	0000	0000	0100	489D	48AD	48BF	000F	00000010

Таблица 2. Трассировка программы №2

Шаг	Машинный код	Команда	Регистры									Флаги
			AX	BX	CX	DX	SP	DS	SS	CS	IP	
1	B409	mov ah, 09	0000	0000	0000	0000	FFFE	489D	48AD	48BF	0100	00000010
2	BA0000	mov dx, 010C	0900	0000	0000	0000	FFFE	489D	48AD	48BF	0102	00000010
3	CD21	int 21	0900	0000	0000	010C	FFFE	489D	48AD	48BF	0105	00000010
4	B8004C	mov ax, 4C00	0900	0000	0000	010C	FFFE	48BD	48AD	48BF	0107	00000010
5	CD21	int 21	4C00	0000	0000	010C	FFFE	48BD	48AD	48BF	010A	00000010

Таблица 3. Трассировка программы №3

### 3 Ответы на контрольные вопросы

#### Вопрос 1 Что такое сегментный (базовый) адрес?

В микропроцессоре есть несколько 16-битных регистров, которые называются сегментными и обозначаются как CS, DS, SS и ES. В них хранятся 16-битные значения, которые называются базовым адресом сегмента.

Микропроцессор объединяет 16-битный исполнительный адрес и 16-битный базовый адрес следующим образом: он расширяет содержимое сегментного регистра (базовый адрес) четырьмя нулями в младших разрядах, делая его 20-битным (полный адрес сегмента), и прибавляет смещение (исполнительный адрес). В результате получается 20-битный адрес, который является физическим или абсолютным адресом ячейки памяти. Физический адрес нулевого смещения есть начало сегмента в виртуальной памяти.

**Вопрос 2** Сделайте листинг для первой программы (файл с расширением `.lst`), выпишите из него размеры сегментов. Из таблицы трассировки к этой программе выпишите базовые адреса сегментов (значение DS при этом нам нужно взять после инициализации адресом сегмента данных). В каком порядке расположились сегменты программы в памяти? Расширяя базовый адрес сегмента до физического адреса, прибавляя размер этого сегмента и округляя до кратного 16 значения, мы можем получить физический адрес следующего за ним сегмента. Сделайте это для первых 2-х сегментов. (Если данные не совпали, значит, неверно заполнена таблица трассировки.)

Code size 0011 (PARA), data size 0014 (PARA), stak size 0100 (PARA).

Выпишем регистры после инициализации регистра DS:

- DS = 48BD
- SS = 48AD
- CS = 48BF

В памяти программа разделена на сегменты, которые расположены в порядке увеличения размера: SS, DS, CS. Физический адрес определяется как базовый адрес, умноженный на 16h, плюс размер сегмента. То есть расширенный базовый адрес сегмента  $SS = 48AD \cdot 10 + 0100(staksize) = 48AD0 + 0100 = 48BD0$  (кратно 16)  $= 48BD = DS$ .



Аналогично для  $DS = 48BD \cdot 10 + 0014$  (data size) =  $48BD0 + 0014 = 48BE4$  (не кратно 16, округляем до ближайшего кратного 16 вверх) =  $48BF = CS$ .

**Вопрос 3** Почему перед началом выполнения первой программы содержимое регистра DS в точности на 10h меньше содержимого регистра SS? (Сравниваются данные из первой строки таблицы трассировки)

Потому что сегменты расположены последовательно и выровнены по 0x10, сегмент стека весит 0x100 байт, а в сегментных регистрах адреса памяти хранятся без разрядов, отвечающих за смещение.

**Вопрос 4** Из таблицы трассировки к первой программе выпишите машинные коды команд `mov AX,data` и `mov AH,09h`. Сколько места в памяти в байтах они занимают? Почему у них разный размер?

`mov AX, data - B8BD48`

0003 - 0000 = 3 байта сдвиг - размер машинного кода.

`mov AH, 09h - B409`

0005 - 0000 = 2 байта сдвиг - размер машинного кода.

Размер команд разный в связи с разным типом данных, помещаемых в регистры.

**Вопрос 5** Из таблицы трассировки ко второй программе выпишите базовые адреса сегментов (значение DS при этом нам нужно взять после инициализации). При использовании модели small сегмент кода располагается в памяти первым. Убедитесь в этом. (Если это не так, значит, вы неверно заполнили таблицу трассировки.)

- DS = 48AF
- SS = 48B1
- CS = 48AD

Сегменты в модели small расположились в следующем порядке: CS, DS, SS. Убедимся в этом:  $CS = 48AD \cdot 10h + 0011$  (code size) =  $48AD0 + 0011 = 48AE1$  (не кратно 16, округляем вверх) =  $48AF0$  (кратно 16, значит можем условно отбросить ноль в конце) =  $48AF = DS$ .  $DS = 48AF \cdot 10h + 0012 = 48AF0 + 0012 = 48B02$  (не кратно 16, округляем вверх) =  $48B1 = SS$ , в чём и следовало убедиться.

**Вопрос 6** Сравните содержимое регистра SP в таблицах трассировки для программ 2 и 3. Объясните, почему получены эти значения.

В программах модели TINY используется только сегмент стека, в связи с этим он инициализируется максимальным возможным значением, поскольку он будет сдвигаться при помещении в него данных. В программе SMALL используются ещё сегменты данных и кода.

**Вопрос 7** Какие операторы называют директивами ассемблера? Приведите примеры директив.

Директивы передают ассемблеру метаданные, необходимые для создания объектного и исполняемого файлов или же листинга.

Директива ASSUME передаёт ассемблеру информацию о соответствии между адресами сегментных регистров и программными сегментами. Директива имеет следующий формат:

```
assume <сегментный регистр>[[, <нара>]]  
assume nothing
```

где - это <сегментный регистр> :<имя сегмента> либо <сегментный регистр> :NOTHING Например, директива

```
assume es:a, ds:b, cs:c
```

сообщает ассемблеру, что для сегментирования адресов из сегмента А выбирается регистр ES, для адресов из сегмента В – регистр DS, а для адресов из сегмента С – регистр CS.

**Вопрос 8** Зачем в последнем предложении end указывают метку, помечающую первую команду программы?

Программа на языке ассемблера состоит из программных модулей, содержащихся в различных файлах. Каждый модуль, в свою очередь, состоит из инструкций процессора или директив ассемблера и заканчивается директивой END. Метка, стоящая после кода псевдооперации END, определяет адрес, с которого должно начаться выполнение программы и называется **точкой входа в программу**.

Каждый модуль также разбивается на отдельные части **директивами сегментации**, определяющими начало и конец сегмента. Каждый сегмент начинается директивой начала сегмента – SEGMENT и заканчивается директивой конца сегмента – ENDS . В начале директив ставится имя сегмента.

Таким образом, метка, указанная в END определяет начальный адрес, с которого процессор должен начать выполнение команды.

**Вопрос 9** Как числа размером в слово хранятся в памяти и как они заносятся в 2-ух байтовые регистры?

В зависимости от архитектуры процессора, применяется прямой или обратный порядок байт. Почти во всех современных процессорах байты с меньшим адресом считаются младшими, такой порядок называется Little Endian. То есть 2 байта ложатся в 2 байта и сначала байт с младшими битами числа, затем байт со старшими. В случае с Big Endian (прямым порядком байт) ситуация обратная, сначала идут старшие разряды: так, как мы привыкли записывать числа на бумаге.

**Вопрос 10** Как инициализируются в программе выводимые на экран текстовые строки?

Выводимые на экран текстовые строки инициализируются в секции .data с помощью db, строка должна оканчиваться знаком \$.

Прежде чем делать int 21h нужно в DX положить адрес начала строки

```
mov dx, offset имя_метки_с_которой_начинается_строка
```

**Вопрос 11** Что нужно сделать, чтобы обратиться к DOS для вывода строки на экран? Как DOS определит, где строка закончилась?

Вывод на экран строки текста и выход из программы осуществляются путем вызова стандартных процедур DOS, называемых **прерываниями**. Прерывания под кодом 21h (33 – в десятичной системе счисления) называются **функциями DOS**, у них нет названий, а только идентификаторы. Номер прерывания и его параметры передаются в регистрах процессора, при этом номер должен находиться в регистре АХ. Так, например, прерывание INT 21h, с помощью которого на экран выводится строка символов, управляется двумя параметрами: в регистре АХ должно быть число 9, а в регистре DX – адрес строки символов, оканчивающейся знаком '\$': это спецсимвол языка ассемблера, которым обозначается нулевой байт (так называемый NUL-символ). Адрес строки Hello загружается в регистр DX с помощью оператора OFFSET (смещение):

```
offset имя
```

Выход из программы осуществляется через функцию DOS с номером 4Ch. Эта функция предполагает, что в регистре AL находится код завершения программы, который она передаст DOS. Если программа завершилась успешно, код завершения должен быть равен 0, поэтому в примере загружаем регистры AH и AL с помощью одной команды `MOV ax,4C00h`, после чего вызываем прерывание 21h.

**Вопрос 12** Программы, которые должны исполняться как .EXE и .COM, имеют существенные различия по:

- размеру
- сегментной структуре
- механизму инициализации

EXE-программы содержат несколько программных сегментов, включая сегмент кода, данных и стека. EXE-файл загружается, начиная с адреса `PSP:0100h`. В процессе загрузки считывается информация EXE-заголовка в начале файла, при помощи которого загрузчик выполняет настройку ссылок на сегменты в загруженном модуле, чтобы учесть тот факт, что программа была загружена в произвольно выбранный сегмент. После настройки ссылок управление передается загрузочному модулю к адресу `CS:IP`, извлеченному из заголовка EXE.

COM-программы содержат единственный сегмент (или, во всяком случае, не содержат явных ссылок на другие сегменты). Образ COM-файла считывается с диска и помещается в память, начиная с `PSP:0100h`, в связи с этим COM-программа должна содержать в начале сегмента кода директиву позволяющую осуществить такую загрузку (`ORG 100h`). Они быстрее загружаются, ибо не требуется перемещения сегментов, и занимают меньше места на диске, поскольку EXE-заголовок и сегмент стека отсутствуют в загрузочном модуле.