

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**ЛАБОРАТОРНАЯ РАБОТА № 1**  
**по дисциплине «Операционные системы»**  
**Тема: Исследование структур загрузочных модулей.**

Студентка гр. 7381

Алясова А.Н.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

## Цель работы.

Исследование различий в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

## Ход работы.

1) На основе шаблона, приведенного в методических указаниях, был написан текст исходного .COM модуля, который определяет тип PC и версию системы. Был получен и “хороший” .COM модуль и “плохой” .EXE модуль. Результаты работы программ представлен ниже.

Файл com.asm

Полученный com.com

```
C:\>tasm com.asm
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International

Assembling file:    com.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  473k

C:\>tlink /t com.obj
Turbo Link Version 5.1 Copyright (c) 1992 Borland International

C:\>com.com
PC Type: FC
Modification number: 5.0
OEM: 255
Serial Number: 000000
```

Полученный com.exe

```
C:\>masm com.asm
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

Object filename [com.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:

49978 + 457282 Bytes symbol space free

0 Warning Errors
0 Severe Errors

C:\>link com.obj

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Run File [COM.EXE]:
List File [NUL.MAP]:
Libraries [LIB]:
LINK : warning L4021: no stack segment
```

```

C:\>com.exe
FC                    5 0          255          000000

          0¼ PC Type:
            5 0          255          000000

0¼ PC Type:
  255          000000

          000000          0¼ PC Type:

          0¼ PC Type:

```

2) Был написан текст программы, построен и отлажен исходный .EXE модуль, который выполняет те же функции, что и модуль .COM. Таким образом, был получен “хороший” .EXE модуль. Результат работы программы представлен ниже.

Файл exe.asm

Полученный exe.exe

```

C:\>masm exe.asm
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

Object filename [exe.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:

49978 + 457282 Bytes symbol space free

0 Warning Errors
0 Severe Errors

C:\>link exe.obj

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Run File [EXE.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

C:\>exe.exe
PC Type: FC
Modification number: 5.0
OEM: 0
Serial Number: 000000

```

## Функции программ

Названия функций	Описание
TETR_TO_HEX	Перевод десятичной цифры в код символа.
BYTE_TO_HEX	Перевод байта в 16-ной с/с в символьный код
WRD_TO_HEX	Перевод слова в 16-ной с/с в символьный код
BYTE_TO_DEC	Перевод байта в 16-ной с/с в символьный код в 10-ной с/с
PRINT_STRING	Вывод строки.

## Ответы на контрольные вопросы:

### *Отличия исходных текстов COM и EXE программ*

1. Сколько сегментов должна содержать COM-программа?

Один сегмент, в котором находятся код и данные.

2. EXE программа?

Программы в формате EXE могут иметь любое количество сегментов. EXE-программа предполагает отдельные сегменты для кода, данных и стека.

3. Какие директивы должны обязательно быть в тексте COM программы?

Директива `ORG 100h`, которая задает смещение для всех адресов программы на 256 байт для префикса программного сегмента (PSP).

Еще необходима директива `ASSUME`, когда она была закомментирована, при компиляции файла `com.com` были выявлены следующие ошибки:

```
**Error** mycom.asm(4) Near jump or call to different CS
**Error** mycom.asm(26) Near jump or call to different CS
**Error** mycom.asm(38) Near jump or call to different CS
**Error** mycom.asm(42) Near jump or call to different CS
**Error** mycom.asm(53) Near jump or call to different CS
**Error** mycom.asm(60) Near jump or call to different CS
**Error** mycom.asm(84) Near jump or call to different CS
**Error** mycom.asm(86) Near jump or call to different CS
**Error** mycom.asm(105) Near jump or call to different CS
**Error** mycom.asm(120) Near jump or call to different CS
**Error** mycom.asm(123) Near jump or call to different CS
**Error** mycom.asm(131) Near jump or call to different CS
**Error** mycom.asm(135) Near jump or call to different CS
**Error** mycom.asm(142) Near jump or call to different CS
**Error** mycom.asm(146) Near jump or call to different CS
**Error** mycom.asm(148) Near jump or call to different CS
**Error** mycom.asm(150) Near jump or call to different CS
**Error** mycom.asm(152) Near jump or call to different CS
Error messages:      18
Warning messages:   None
Passes:              1
Remaining memory:   473k
```

С помощью директивы `ASSUME` ассемблеру сообщается информация о соответствии между сегментными регистрами, и программными сегментами.

Все сегменты сами по себе равноправны, для того чтобы использовать их как сегменты кода, данных или стека, необходимо предварительно сообщить транслятору об этом, для чего используют специальную директиву `ASSUME`. Эта директива сообщает транслятору о том, какой сегмент к какому

сегментному регистру привязан. В свою очередь, это позволит транслятору корректно связывать символические имена, определенные в сегментах. Привязка сегментов к сегментным регистрам осуществляется с помощью операндов этой директивы, в которых ИмяСегмента должно быть именем сегмента, определенным в исходном тексте программы директивой SEGMENT или ключевым словом nothing. Если в качестве операнда используется только ключевое слово nothing, то предшествующие назначения сегментных регистров аннулируются, причем сразу для всех шести сегментных регистров. Но ключевое слово nothing можно использовать вместо аргумента ИмяСегмента, в этом случае будет выборочно разрываться связь между сегментом с именем ИмяСегмента и соответствующим сегментным регистром.

#### 4. Все ли форматы команд можно использовать в COM программе?

Нет, не все. COM-программа подразумевает наличие только одного сегмента, а значит, можно использовать только near-переходы, так как в far-переходах подразумевается использование нескольких сегментов. Так же нельзя использовать команды, связанные с адресом сегмента, потому что адрес сегмента до загрузки неизвестен, так как в COM-программах в DOS не содержится таблицы настройки, которая содержит описание адресов, зависящих от размещения загрузочного модуля в ОП, потому что подобные адреса в нем запрещены.

### ***Отличия форматов файлов COM и EXE модулей***

#### 1. Какова структура файла COM? С какого адреса располагается код?

.COM-файл состоит из команд, процедур и данных, используемых в программе. Код начинается с 0 адреса.

#### 2. Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с 0 адреса?

В файле EXE данные и код содержатся в одном сегменте. С 0 адреса располагается "подпись" компоновщика, указывающая, что файл является файлом EXE. Код начинается с адреса 300h.

C:\Users\Анастасия\Desktop\4 семестр\ОС\1\bad.EXE																h	1251	1069 Col	0	41%
000000001C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000001D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000001E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000001F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000200:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000210:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000220:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000230:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000240:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000250:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000260:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000270:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000280:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000290:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000002F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000000300:	E9	AC	00	50	43	20	54	79	70	65	3A	20	20	0D	0A	24	й~	PC Type:	♪\$	
00000000310:	4D	6F	64	69	66	69	63	61	74	69	6F	6E	20	6E	75	6D		Modification num		
00000000320:	62	65	72	3A	20	20	2E	20	20	0D	0A	24	4F	45	4D	3A	ber:	♪\$OEM:		
00000000330:	20	20	20	0D	0A	24	53	65	72	69	61	6C	20	4E	75	6D		♪\$Serial Num		
00000000340:	62	65	72	3A	20	20	20	20	20	20	20	0D	0A	24	B4	09	ber:	♪\$ro		
00000000350:	CD	21	C3	24	0F	3C	09	76	02	04	07	04	30	C3	51	8A	НИГ\$<ov@♦♦@ГQ/ь			
00000000360:	C4	E8	EF	FF	86	C4	B1	04	D2	E8	E8	E6	FF	59	C3	53	Дипя†Д±♦ТиижяYFS			
00000000370:	8A	FC	E8	E9	FF	88	25	4F	88	05	4F	8A	C7	32	E4	E8	льийя€%O€#O/ь32ди			

MZ в начале EXE модуля это формат исполняемых файлов MS DOS, он не является частью таблицы настроек адресов, так как таблица состоит из элементов, число которых записано в байтах 06-07. Элемент таблицы настройки состоит из двух полей: 2-х байтного смещения и 2-х байтного сегмента, и указывает слова в загрузочном модуле, содержащее адрес, который должен быть настроен на место памяти, в которое загружается задача.

C:\Users\Анастасия\Desktop\4 семестр\ОС\1\bad.EXE																h 1251		
0000000000:	4D	5A	2D	00	03	00	00	00	20	00	00	00	FF	FF	00	00	MZ- ♥	яя
0000000010:	00	00	C9	71	00	01	00	00	1E	00	00	00	01	00	00	00	Йq @ ▲ @	
0000000020:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000050:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000060:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000070:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000080:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		

3. Какова структура файла «хорошего» EXE? Чем он отличается от «плохого» EXE файла?



C:\Users\Анастасия\Desktop\4 семестр\OC\1\good.EXE										h 1251	824 Col									
00000000D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000000F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000100:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000110:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000120:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000130:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000140:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000150:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000160:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000170:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000180:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000190:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000001A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000001B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000001C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000001D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000001E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000001F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000000200:	50	43	20	54	79	70	65	3A	20	20	0D	0A	24	4D	6F	64	PC Type: 00\$Mod			
0000000210:	69	66	69	63	61	74	69	6F	6E	20	6E	75	6D	62	65	72	ification number			
0000000220:	3A	20	20	2E	20	20	0D	0A	24	4F	45	4D	3A	20	20	20	: . 00\$OEM:			
0000000230:	0D	0A	24	53	65	72	69	61	6C	20	4E	75	6D	62	65	72	00\$Serial Number			
0000000240:	3A	20	20	20	20	20	20	20	0D	0A	24	00	00	00	00	00	: 00\$			
0000000250:	B4	09	CD	21	C3	24	0F	3C	09	76	02	04	07	04	30	C3	r0H!Г\$α<ov0♦♦0Г			
0000000260:	51	8A	C4	E8	EF	FF	86	C4	B1	04	D2	E8	E8	E6	FF	59	QьДипя†Д±♦ТиижяУ			
0000000270:	C3	53	8A	FC	E8	E9	FF	88	25	4F	88	05	4F	8A	C7	32	ГSьийя€%0€*0ь32			

В отличие от плохого, хороший EXE-файл не содержит директивы ORG 100h (которая выделяет память под PSP), поэтому код начинается с адреса 200h. В «хорошем» EXE данные, стек и код разделены по сегментам.

### Загрузка COM модуля в основную память

1. Какой формат загрузки COM модуля? С какого адреса располагается код?

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h.

2. Что располагается с 0 адреса?

С адреса 0 располагается префикс программного сегмента (PSP).

3. Какие значения имеют сегментные регистры? На какие области памяти они указывают?

Сегментные регистры имеют значения, которые соответствуют сегменту, в который модуль был помещен управляющей программой. Все они указывают на один и тот же сегмент памяти, поэтому все регистры имеют значения 48DD. Они указывают на PSP.



ds	48DD
es	48DD
ss	48DD
cs	48DD

4. Как определяется стек? Какую область памяти он занимает? Какие адреса?

Стек создается автоматически, указатель стека в конце сегмента. Он занимает оставшуюся память и адреса изменяются от больших к меньшим, то есть от FFFh к 0000h.

### ***Загрузка «хорошего» EXE модуля в память***

1. Как загружается «хороший» EXE? Какие значения имеют сегментные регистры?

Сначала создается PSP. Затем определяется длина тела загрузочного модуля, определяется начальный сегмент. Загрузочный модуль считывается в начальный сегмент, таблица настройки считывается в рабочую память, к полю каждого сегмента прибавляется сегментный адрес начального сегмента, определяются значения сегментных регистров. DS и ES указывают на начало PSP (48DD), CS – на начало сегмента команд (4932), а SS – на начало сегмента стека (48ED).

ds	48DD
es	48DD
ss	48ED
cs	4932

2. На что указывают регистры DS и ES?

Изначально регистры DS и ES указывают на начало сегмента PSP.

3. Как определяется стек?

Стек определяется при объявлении сегмента стека, в котором указывается, сколько памяти необходимо выделить. В регистры SS и SP записываются значения, указанные в заголовке, а к SS прибавляется сегментный адрес начального сегмента.

4. Как определяется точка входа?

С помощью директивы END, операндом которой является адрес, с которого начинается выполнение программы.

# ПРИЛОЖЕНИЕ А

## КОДЫ ИСХОДНЫХ ПРОГРАММ

*com.asm*

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START:    JMP    BEGIN

;ДАННЫЕ
PC_Type      db    'PC Type:  ', 0dh, 0ah, '$'
Mod_numb     db    'Modification number:  .  ', 0dh, 0ah, '$'
OEM          db    'OEM:  ', 0dh, 0ah, '$'
S_numb       db    'Serial Number:          ', 0dh, 0ah, '$'

;ПРОЦЕДУРЫ
;-----
;печать строки
PRINT_STRING PROC near
    mov     ah, 09h
    int     21h
    ret
PRINT_STRING ENDP

;-----
;перевод десятичной цифры в код символа
TETR_TO_HEX  PROC near
    and     al, 0fh ;логическое умножение всех пар битов
    cmp     al, 09
    jbe     NEXT ;Переход если ниже или равно
    add     al, 07
NEXT: add     al, 30h
    ret
TETR_TO_HEX  ENDP

;-----
;-----
;перевод байта 16 с.с в символьный код
;байт в AL переводится в два символа шестнадцатеричного числа в AX
BYTE_TO_HEX  PROC near
    push    cx
    mov     al, ah
    call    TETR_TO_HEX
```

```

        xchg al, ah
        mov     cl, 4
        shr     al, cl ;логический сдвиг вправо
        call    TETR_TO_HEX ;в AL старшая цифра
        pop     cx           ;в AH младшая
        ret
BYTE_TO_HEX      ENDP

```

;-----  
;перевод в 16 с/с 16-ти разрядного числа  
;в AX - число, DI - адрес последнего символа

```

WRD_TO_HEX      PROC near
        push    bx
        mov     bh, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        dec     di
        mov     al, bh
        xor     ah, ah
        call    BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        pop     bx
        ret
WRD_TO_HEX      ENDP

```

;-----  
;перевод байта 16 с.с в символьный код 10 с.с  
;si - адрес поля младшей цифры

```

BYTE_TO_DEC      PROC near
        push    cx
        push    dx
        push    ax
        xor     ah, ah
        xor     dx, dx
        mov     cx, 10
loop_bd:div       cx
        or      dl, 30h
        mov     [si], dl
        dec     si
        xor     dx, dx

```

```

        cmp     ax, 10
        jae     loop_bd
        cmp     ax, 00h
        jbe     end_1
        or      al, 30h
        mov     [si], al
end_1:   pop     ax
        pop     dx
        pop     cx
        ret

BYTE_TO_DEC      ENDP

```

```

;-----
-----

```

BEGIN:

```

;PC_Type
push es
push bx
push ax
mov  bx, 0F000h
mov  es, bx
mov  ax, es:[0FFFEh]
mov  ah, al
call BYTE_TO_HEX
lea  bx, PC_Type
mov  [bx + 9], ax; смещение на количество символов
pop  ax
pop  bx
pop  es

mov  ah, 30h
int  21h

;Mod_numb
push ax
push si
lea  si, Mod_numb
add  si, 21
call BYTE_TO_DEC
add  si, 3
mov  al, ah
call BYTE_TO_DEC
pop  si
pop  ax

```

```

;OEM
mov  al, bh
lea  si, OEM
add  si, 7
call BYTE_TO_DEC

;S_numb
mov  al, bl
call BYTE_TO_HEX
lea  di, S_numb
add  di, 15
mov  [di], ax
mov  ax, cx
lea  di, S_numb
add  di, 20
call WRD_TO_HEX

;ВЫВОД
lea  dx, PC_Type
call PRINT_STRING
lea  dx, Mod_numb
call PRINT_STRING
lea  dx, OEM
call PRINT_STRING
lea  dx, S_numb
call PRINT_STRING

;ВЫХОД в dos
xor  al, al
mov  ah, 4ch
int  21h
ret
TESTPC ENDS
END  START

```

*exe.asm*

```
AStack    SEGMENT    STACK
AStack    ENDS
```

```
DATA SEGMENT
PC_Type      db    'PC Type:  ', 0dh, 0ah, '$'
Mod_numb     db    'Modification number:  .  ', 0dh, 0ah, '$'
OEM          db    'OEM:  ', 0dh, 0ah, '$'
S_numb       db    'Serial Number:      ', 0dh, 0ah, '$'
DATA ENDS
```

```
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA, SS:AStack
```

```
;ПРОЦЕДУРЫ
```

```
;-----
```

```
-----
```

```
;печать строки
```

```
PRINT_STRING PROC near
        mov     ah, 09h
        int     21h
        ret
```

```
PRINT_STRING ENDP
```

```
;-----
```

```
-----
```

```
;перевод десятичной цифры в код символа
```

```
TETR_TO_HEX      PROC near
        and     al, 0fh ;логическое умножение всех пар битов
        cmp     al, 09
        jbe     NEXT ;Переход если ниже или равно
        add     al, 07
NEXT: add         al, 30h
        ret
```

```
TETR_TO_HEX      ENDP
```

```
;-----
```

```
-----
```

```
;перевод байта 16 с.с в символьный код
```

```
;байт в AL переводится в 2 символа шестнадцетиричного числа в AX
```

```
BYTE_TO_HEX      PROC near
        push    cx
```



```

        mov     al, ah
        call   TETR_TO_HEX
        xchg   al, ah
        mov     cl, 4
        shr     al, cl ;логический сдвиг вправо
        call   TETR_TO_HEX
        pop     cx
        ret
BYTE_TO_HEX      ENDP

```

```

;-----
-----

```

```

;перевод в 16 с/с 16-ти разрядного числа
;в AX - число, DI - адрес последнего символа
WRD_TO_HEX      PROC near

```

```

        push   bx
        mov     bh, ah
        call   BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        dec     di
        mov     al, bh
        xor     ah, ah
        call   BYTE_TO_HEX
        mov     [di], ah
        dec     di
        mov     [di], al
        pop     bx
        ret
WRD_TO_HEX      ENDP

```

```

;-----
-----

```

```

;перевод байта 16 с.с в символьный код 10 с.с
;si - адрес поля младшей цифры
BYTE_TO_DEC      PROC near

```

```

        push   cx
        push   dx
        push   ax
        xor     ah, ah
        xor     dx, dx

```

```

        mov     cx, 10
loop_bd:div     cx
        or      dl, 30h
        mov     [si], dl
        dec     si
        xor     dx, dx
        cmp     ax, 10
        jae     loop_bd
        cmp     ax, 00h
        jbe     end_1
        or      al, 30h
        mov     [si], al
end_1:   pop     ax
        pop     dx
        pop     cx
        ret
BYTE_TO_DEC      ENDP

```

```

;-----
-----

```

```

main:
        push    ds
        sub     ax, ax
        push    ax
        mov     ax, DATA
        mov     ds, ax

        ;PC_Type
        push    es
        push    bx
        push    ax
        mov     bx, 0F000h
        mov     es, bx
        mov     ax, es:[0FFFEh]
        mov     ah, al
        call    BYTE_TO_HEX
        lea     bx, PC_Type
        mov     [bx + 9], ax ;смещение на количество символов
        pop     ax
        pop     bx
        pop     es

        mov     ah, 30h

```

```

int          21h

;Mod_numb
push ax
push si
lea         si, Mod_numb
add         si, 21
call BYTE_TO_DEC
add         si, 3
mov  al, ah
call        BYTE_TO_DEC
pop  si
pop  ax

;OEM
mov  al, bh
lea  si, OEM
add  si, 7
call BYTE_TO_DEC

;S_Numb
mov  al, bl
call BYTE_TO_HEX
lea  di, S_numb
add  di, 15
mov  [di], ax
mov  ax, cx
lea  di, S_numb
add  di, 20
call WRD_TO_HEX

lea  dx, PC_Type
call PRINT_STRING
lea  dx, Mod_numb
call PRINT_STRING
lea  dx, OEM
call PRINT_STRING
lea  dx, S_numb
call PRINT_STRING

;выход в dos
xor  al, al
mov  ah, 4ch
int  21h

```

```
CODE    ret
        ENDS
        END    main
```