

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

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Операционные системы»
ТЕМА: ИССЛЕДОВАНИЕ СТРУКТУР ЗАГРУЗОЧНЫХ МОДУЛЕЙ.

Студент гр. 9381

Шахин Н.С

Преподаватель

Ефремов М. А.

Санкт-Петербург

2021

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 com.asm, который определяет тип PC и версию системы. Далее при помощи транслятора MASM.exe и компоновщика LINK.exe был скомпилирован плохой .EXE модуль. При помощи EXE2BIN.EXE был построен хороший .COM модуль.

2. Был написан текст исходного .Exe модуля exe.asm. Далее при помощи транслятора MASM.exe и компоновщика LINK.exe был скомпилирован хороший .EXE модуль.

3. Было выполнено сравнение исходных текстов com.asm и exe.asm.

4. При помощи программы FAR файлы загрузочных модулей com.com, com.exe и exe.exe были открыты в шестнадцатеричном виде.

5. Был исследован загрузочный модуль .com при помощи отладчика.

6. был исследован загрузочный модуль .exe при помощи отладчика.

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).

Директива `ASSUME`. Указывает ассемблеру, с каким сегментом или группой сегментов связан тот или иной сегментный регистр. Она не изменяет значений сегментных регистров, а только позволяет ассемблеру проверять допустимость ссылок и самостоятельно вставлять при необходимости префиксы переопределения сегментов. Без этой директивы программа не скомпилируется, так как ассемблер не будет понимать, относительно чего вычислять смещения меток.

Директива `END`. Этой директивой завершается любая программа на ассемблере.

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

Нет, не все. 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?

В COM-файле код, данные и стек располагаются в одном сегменте. Код (как и данные) начинается с адреса `0h`.

```

C:\Users\ASUS\Documents\asm\MASM\COM.COM
00000000: E9 AC 00 50 43 20 54 79 70 65 3A 20 20 0D 0A 24 й- PC Type: Jm$
00000001: 4D 6F 64 69 66 69 63 61 74 69 6F 6E 20 6E 75 6D Modification num
00000002: 62 65 72 3A 20 20 2E 20 20 0D 0A 24 4F 45 4D 3A ber: . Jm$OEM:
00000003: 20 20 20 0D 0A 24 53 65 72 69 61 6C 20 4E 75 6D Jm$Serial Num
00000004: 62 65 72 3A 20 20 20 20 20 20 20 0D 0A 24 B4 09 ber: Jm$ro
00000005: CD 21 C3 24 0F 3C 09 76 02 04 07 04 30 C3 51 8A HIF$<ov***0GQb
00000006: C4 E8 EF FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 Дипя*Д+ТиижяYGS
00000007: 8A FC E8 E9 FF 88 25 4F 88 05 4F 8A C7 32 E4 E8 Ъийл€%O€+OЉ32ди
00000008: DC FF 88 25 4F 88 05 5B C3 51 52 50 32 E4 33 D2 бя€%O€+[GQRP2д3T
00000009: 89 0A 00 F7 F1 80 CA 30 88 14 4E 33 D2 3D 0A 00 M чсбK0€N3T=ж
0000000A: 73 F1 3D 00 00 76 04 0C 30 88 04 58 5A 59 C3 06 sc= v+90€+XZYГ+
0000000B: 53 50 8B 00 F0 8E C3 26 A1 FE FF 8A E0 E8 9E FF SP рбГ&YюяЪийлЯ
0000000C: 8D 1E 03 01 89 47 09 58 5B 07 B4 30 CD 21 50 56 K▲%0G0X[+r0H!PV
0000000D: 8D 36 10 01 83 C6 15 E8 AF FF 83 C6 03 8A C4 E8 Kб-0ГЖЉиІяГЖЉди
0000000E: A7 FF 5E 58 8A C7 8D 36 2C 01 83 C6 07 E8 99 FF 5я^XЉ3K6,0ГЖ+и"я
0000000F: 8A C3 E8 69 FF 8D 3E 36 01 83 C7 0F 89 05 8B C1 ЪГиіяK>60Г30Љ+б
00000010: 8D 3E 36 01 83 C7 14 E8 65 FF 8D 16 03 01 E8 3D K>60Г3ЉиіяK-▼0и=
00000011: FF 8D 16 10 01 E8 36 FF 8D 16 2C 01 E8 2F FF 8D яK→0ибяK-,0и/яK
00000012: 16 36 01 E8 28 FF 32 C0 B4 4C CD 21 C3 =60и(я2ArLHIF

```

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

В плохом EXE-файле код, данные и стек располагаются в одном сегменте. Код (как и данные) начинается с адреса 300h. С адреса 0h располагается управляющая информация для загрузчика, которая содержит заголовок и таблицу настройки адресов.

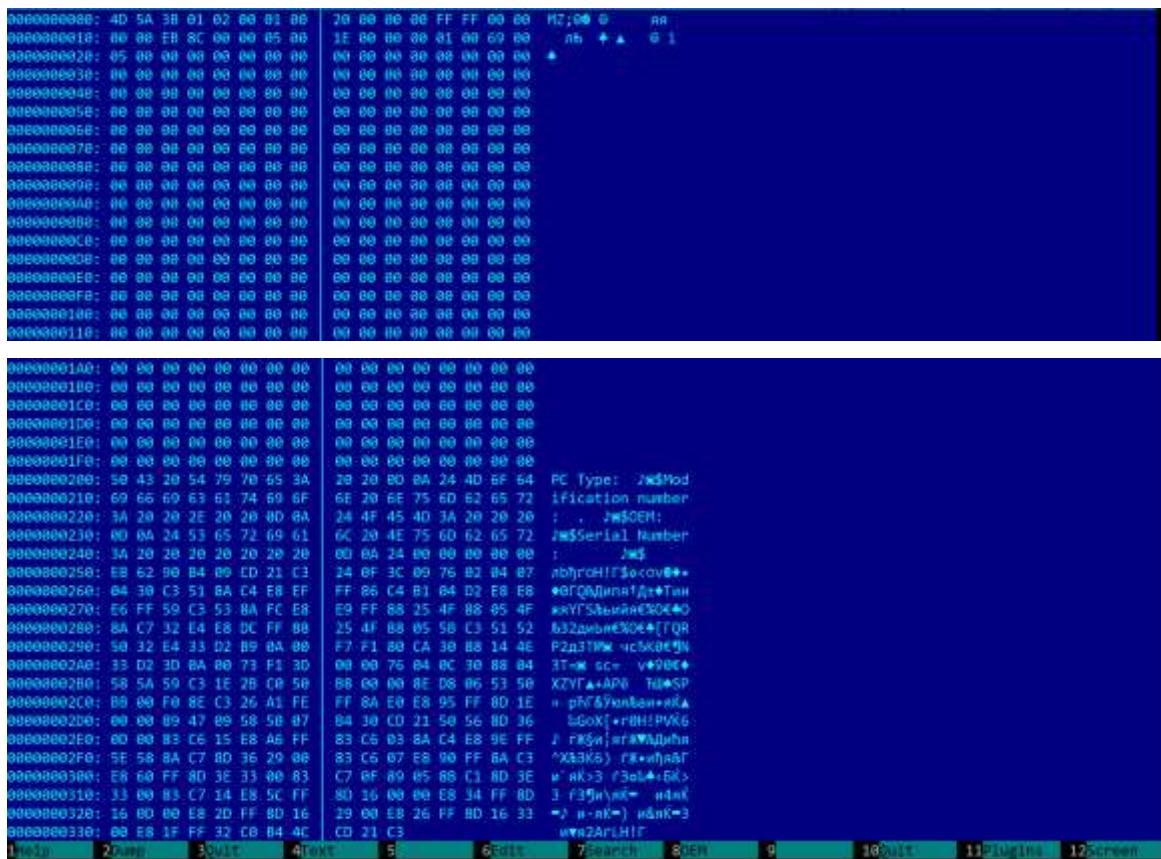
```

C:\Users\ASUS\Documents\asm\MASM\COM.EXE
00000000: 4D 5A 2D 00 03 00 00 00 20 00 00 00 FF FF 00 00 MZ- ♥   яя
00000001: 00 00 C9 71 00 01 00 00 1E 00 00 00 01 00 00 00 йq  @   @
00000002: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000003: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000004: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000006: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000007: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000009: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000011: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000012: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000013: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000014: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000015: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000017: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000019: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000021: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000022: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000023: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000024: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000025: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000026: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000027: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000028: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000029: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030: E9 AC 00 50 43 20 54 79 70 65 3A 20 20 0D 0A 24 а- PC Type: Jm$
00000031: 4D 6F 64 69 66 69 63 61 74 69 6F 6E 20 6E 75 6D Modification num
00000032: 62 65 72 3A 20 20 2E 20 20 0D 0A 24 4F 45 4D 3A ber: . Jm$OEM:
00000033: 20 20 20 0D 0A 24 53 65 72 69 61 6C 20 4E 75 6D Jm$Serial Num
00000034: 62 65 72 3A 20 20 20 20 20 20 20 0D 0A 24 B4 09 ber: Jm$ro
00000035: CD 21 C3 24 0F 3C 09 76 02 04 07 04 30 C3 51 8A HIF$<ov***0GQb
00000036: C4 E8 EF FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 Дипя*Д+ТиижяYGS
00000037: 8A FC E8 E9 FF 88 25 4F 88 05 4F 8A C7 32 E4 E8 Ъийл€%O€+OЉ32ди
00000038: DC FF 88 25 4F 88 05 5B C3 51 52 50 32 E4 33 D2 бя€%O€+[GQRP2д3T
00000039: 89 0A 00 F7 F1 80 CA 30 88 14 4E 33 D2 3D 0A 00 M чсбK0€N3T=ж
0000003A: 73 F1 3D 00 00 76 04 0C 30 88 04 58 5A 59 C3 06 sc= v+90€+XZYГ+
0000003B: 53 50 8B 00 F0 8E C3 26 A1 FE FF 8A E0 E8 9E FF SP рбГ&YюяЪийлЯ
0000003C: 8D 1E 03 01 89 47 09 58 5B 07 B4 30 CD 21 50 56 K▲%0G0X[+r0H!PV
0000003D: 8D 36 10 01 83 C6 15 E8 AF FF 83 C6 03 8A C4 E8 Kб-0ГЖЉиІяГЖЉди
0000003E: A7 FF 5E 58 8A C7 8D 36 2C 01 83 C6 07 E8 99 FF 5я^XЉ3K6,0ГЖ+и"я
0000003F: 8A C3 E8 69 FF 8D 3E 36 01 83 C7 0F 89 05 8B C1 ЪГиіяK>60Г30Љ+б
00000040: 8D 3E 36 01 83 C7 14 E8 65 FF 8D 16 03 01 E8 3D K>60Г3ЉиіяK-▼0и=
00000041: FF 8D 16 10 01 E8 36 FF 8D 16 2C 01 E8 2F FF 8D яK→0ибяK-,0и/яK
00000042: 16 36 01 E8 28 FF 32 C0 B4 4C CD 21 C3 =60и(я2ArLHIF

```

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

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-файле код, данные и стек находятся в различных сегментах, а в плохом – в одном и том же сегменте.



Загрузка СОМ 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?

Регистры DS, ES, CS, SS указывают на начало блока PSP.

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

Стек создается автоматически. Регистр SS указывает на начало блока PSP (0h), а SP указывает на конец модуля (FFFFh). То есть стек расположен между адресами SS:0000h и SS:FFFFh и заполняется с конца модуля в сторону уменьшения адресов.

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 (19f5), CS – на начало сегмента команд (1A0A), а SS – на начало сегмента стека (1A05).

DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...

AX 0000	SI 0000	CS 1A0A	IP 0000	Stack +0 4350	Flags 7202
BX 0000	DI 0000	DS 19F5		+2 5420	
CX 013B	BP 0000	ES 19F5	HS 19F5	+4 7079	OF DF IF SF ZF AF PF CF
DX 0000	SP 0000	SS 1A05	FS 19F5	+6 3A65	0 0 1 0 0 0 0 0

CMD >

0000 EB62	JMP	0064
0002 90	NOP	
0003 B409	MOV	AH,09
0005 CD21	INT	21
0007 C3	RET	
0008 240F	AND	AL,0F
000A 3C09	CMP	AL,09
000C 7602	JNA	0010

DS:0000	CD 20 FF 9F 00 EA F0 FE
DS:0008	AD DE 1B 05 C5 06 00 00
DS:0010	18 01 10 01 18 01 92 01
DS:0018	01 01 01 00 02 FF FF FF
DS:0020	FF FF FF FF FF FF FF FF
DS:0028	FF FF FF FF EB 19 C0 11
DS:0030	A2 01 14 00 18 00 F5 19
DS:0038	FF FF FF FF 00 00 00 00
DS:0040	05 00 00 00 00 00 00 00
DS:0048	00 00 00 00 00 00 00 00

1 Step 2ProcStep 3Retrieve 4Help ON 5BRK Menu 6 7 ↑ 8 ↓ 9 ← 10 →

2. На что указывают регистры DS и ES?

Изначально регистры DS и ES указывают на начало сегмента PSP.

3. Как определяется стек?

Регистр SS указывает на начало сегмента стека, а SP – на конец сегмента стека.

4. Как определяется точка входа?

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

Вывод.

Были изучены различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл 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 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 ;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_l
        or      al, 30h
        mov     [si], al
end_l:         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

        ;Mod_numb
        mov     ah, 30h
        int     21h

```

```

        push    ax
        push    si
        lea     si, Mod_num
        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_num
        mov     al, bl
        call    BYTE_TO_HEX
        lea     di, S_num
        add     di, 15
        mov     [di], ax
        mov     ax, cx
        lea     di, S_num
        add     di, 20
        call    WRD_TO_HEX

        lea     dx, PC_Type
        call    PRINT_STRING
        lea     dx, Mod_num
        call    PRINT_STRING
        lea     dx, OEM
        call    PRINT_STRING
        lea     dx, S_num
        call    PRINT_STRING

        xor     al, al
        mov     ah, 4ch
        int     21h
        ret

TESTPC  ENDS
        END     START

```

Файл exe.asm:

```

AStack  SEGMENT  STACK
AStack  ENDS

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

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

START:
        jmp BEGIN
;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 символный 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_l
    or      al, 30h
    mov     [si], al
end_l:    pop     ax
    pop     dx
    pop     cx
    ret
BYTE_TO_DEC ENDP

```

```

;-----
BEGIN:
    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

    ;Mod_numb
    mov     ah, 30h
    int     21h
    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
ret

CODE     ENDS

END       START

```