

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. 7383

Корякин М.П.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

Постановка задачи.

Исследование структур модулей типов .COM и .EXE, различие типов.
Изучение способов их загрузки в основную память.

Описание функций:

- Write – вызов функции печати строки.
- TYPE_OS – запись типа ОС.
- VERSION – запись версии ОС, номер OEM, а также номер пользователя.
- HELP_BYTE_TO_HEX – выводит на экран номер пользователя.
- TETR_TO_HEX – вспомогательная функция для работы BYTE_TO_HEX.
- BYTE_TO_HEX - Байт AL переводится в два символа шестн. числа AX.
- BYTE_TO_DEC – Перевод в 10сс, SI - адрес поля младшей цифры.

Описание данных:

- TypeOS – для вывода типа ОС.
- VERS – для вывода версии ОС.
- OEM – для вывода номера OEM.
- USER_NUM – для вывода номера пользователя.
- ENDSTR – для переноса строки.
- PC, PC_XT, AT, PS2_30, PS2_80, PSjr, PC_Conv – для определения типа ОС, и в последствии вывода одного из них.

Программа определяет и выводит на экран: тип ОС, версию ОС, номер OEM, серийный номер пользователя.

Результаты работы программы.

```
type OS: AT
Version: 5.0
OEM: 255
Serial number user: 000000
```

Рисунок 1 – Результат работы программы l1com.com

```
F:\>l1com.exe

type OS:

type OS:      5 0

type OS:      255

type OS:

type OS: 000000

type OS:
```

Рисунок 2 – Результат работы программы l1com.exe

```
F:\>l1.exe
type OS: AT
Version: 5.0
OEM: 255
Serial number user: 000000
```

Рисунок 3 – Результат работы программы l1.exe

Вывод.

В результате исследование структур модулей .COM и .EXE были изучены различия этих структур, а также способы их загрузки в основную память.

Ответы на контрольные вопросы лабораторной работы №1:

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

Один сегмент.

2. EXE-программа?

EXE-программа должна содержать минимум один сегмент.

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

Обязательно должна быть директива `ORG 100h`, которая резервирует 256 байт для PSP. Также обязательной директивой является `ASSUME`, которая ставит в соответствие начало программы сегментам кода и данных.

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

Нет, не все. Нельзя использовать команды вида `mov` регистр, сегмент(`DATA`, `CODE`). Нельзя использовать `far` – адресацию (дальняя адресация), т.к. в этом случае будет использоваться таблица настройки, которой нет в COM файле(но есть в EXE файле).

Отличия форматов файлов COM и EXE модулей

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

000000000: E9 85 01 74 79 70 65 20	4F 53 3A 20 24 56 65 72	é...@type OS: \$Ver
000000010: 73 69 6F 6E 3A 20 20 2E	20 20 0D 0A 24 4F 45 4D	sion: . \$OEM
000000020: 3A 20 20 20 20 0D 0A 24	53 65 72 69 61 6C 20 6E	: \$Serial n
000000030: 75 6D 62 65 72 20 75 73	65 72 3A 20 24 20 20 20	umber user: \$
000000040: 20 24 0D 0A 24 50 43 0D	0A 24 50 43 2F 58 54 0D	\$PC\$PC/XT
000000050: 0A 24 41 54 0D 0A 24 50	53 32 20 6D 6F 64 65 6C	\$AT\$PS2 model
000000060: 20 33 30 0D 0A 24 50 53	32 20 6D 6F 64 65 6C 20	30\$PS2 model
000000070: 38 30 0D 0A 24 50 43 6A	72 0D 0A 24 50 43 20 43	80\$PCjr\$PC C
000000080: 6F 6E 76 65 72 74 69 62	6C 65 0D 0A 24 B4 09 CD	onvertible\$'oÍ
000000090: 21 C3 BA 03 01 E8 F5 FF	B8 00 F0 8E C0 26 A1 FE	!Ã♥0èöÿ. ôŽ&¡þ
0000000A0: FF 3C FF 74 1C 3C FE 74	1E 3C FB 74 1A 3C FC 74	ÿ<ÿtL<þt<ût"<ût
0000000B0: 1C 3C FA 74 1E 3C F8 74	20 3C FD 74 22 3C F9 74	L<ût<þt <ÿt"<ût
0000000C0: 24 BA 45 01 EB 25 90 BA	4A 01 EB 1F 90 BA 52 01	\$E0E%0J0E0E0E0E
0000000D0: EB 19 90 BA 57 01 EB 13	90 BA 66 01 EB 0D 90 BA	èJ0E0E0E0E0E0E0E
0000000E0: 75 01 EB 07 90 BA 7C 01	EB 01 90 E8 9F FF C3 B8	u0è.0E 0è0E0Eÿÿ.
0000000F0: 00 00 B4 30 CD 21 BE 0D	01 83 C6 09 E8 66 00 8A	0Í!%0f0e0ef Š
000000100: C4 83 C6 03 E8 5E 00 BA	0D 01 E8 80 FF BE 1D 01	ÄfÄ♥è^ 0J0èèÿ%00
000000110: 83 C6 07 8A C7 E8 4D 00	BA 1D 01 E8 6F FF BA 28	fÄ•ŠCèM 000è0ÿ0(
000000120: 01 E8 69 FF 8A C3 E8 11	00 8A C5 E8 0C 00 8A C1	0èiÿŠÄè ŠÄe0 ŠÄ
000000130: E8 07 00 BA 42 01 E8 54	FF C3 E8 17 00 8B D8 8A	è• 0B0èTÿÄè± <0Š
000000140: D3 B4 02 CD 21 8A D7 CD	21 24 0F 3C 09 76 02 04	Ó0Í!Š×Í!\$0<ov0♦
000000150: 07 04 30 C3 51 8A E0 E8	EF FF 86 C4 B1 04 D2 E8	♦0ÄQŠàèiÿ†Ä±0è
000000160: E8 E6 FF 59 C3 51 52 32	E4 33 D2 B9 0A 00 F7 F1	èæÿYÄQR2ä3D¹ ÷ñ
000000170: 80 CA 30 88 14 4E 33 D2	3D 0A 00 73 F1 3C 00 74	€Ê0^JN3D= sñ< t
000000180: 04 0C 30 88 04 5A 59 C3	E8 07 FF E8 61 FF 32 C0	♦00^♦ZYÄè•ÿèaÿ2Ä
000000190: B4 4C CD 21		ˆLÍ!

Рисунок 4 – Представление COM файла в шестнадцатеричном виде

COM файл не содержит таблицы настроек. (Таблица настроек

необходима для EXE файла, она содержит ссылки на сегменты

программы. При создании COM программы код находится в одном

Поэтому код располагается с нулевого адреса, как показано на рис.4.

- | | | | |
|-----------|-------------------------|-------------------------|-----------|
| 00000000: | 4D 5A 8F 00 03 00 00 00 | 20 00 00 00 FF FF 00 00 | MZUJ ♥ ЯЯ |
| 00000010: | 00 00 95 DD 00 01 00 00 | 1E 00 00 00 01 00 00 00 | •Э @ ▲ @ |
| 00000020: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000030: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000040: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000050: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000060: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000070: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000080: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000090: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000000A0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000000B0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000000C0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000000D0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000000E0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000000F0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000100: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000110: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000120: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000130: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000140: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000150: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000160: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000170: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000180: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000190: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000001A0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000001B0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000001C0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000001D0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000001E0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000001F0: | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |

5

00000001F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000200:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000210:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000220:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000230:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000240:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000250:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000260:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000270:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000280:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000290:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000002A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000002B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000002C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000002D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000002E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000002F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000000300:	E9 80 01 74 79 70 65 20	4F 53 3A 20 24 56 65 72
0000000310:	73 69 6F 6E 3A 20 20 2E	20 20 0D 0A 24 4F 45 4D
0000000320:	3A 20 20 20 20 0D 0A 24	53 65 72 69 61 6C 20 6E
0000000330:	75 6D 62 65 72 20 75 73	65 72 3A 20 24 0D 0A 24
0000000340:	50 43 0D 0A 24 50 43 2F	58 54 0D 0A 24 41 54 0D
0000000350:	0A 24 50 53 32 20 6D 6F	64 65 6C 20 33 30 0D 0A
0000000360:	24 50 53 32 20 6D 6F 64	65 6C 20 38 30 0D 0A 24
0000000370:	50 43 6A 72 0D 0A 24 50	43 20 43 6F 6E 76 65 72
0000000380:	74 69 62 6C 65 0D 0A 24	B4 09 CD 21 C3 BA 03 01
0000000390:	E8 F5 FF B8 00 F0 8E C0	26 A1 FE FF 3C FF 74 1C
00000003A0:	3C FE 74 1E 3C FB 74 1A	3C FC 74 1C 3C FA 74 1E
00000003B0:	3C F8 74 20 3C FD 74 22	3C F9 74 24 BA 40 01 EB
00000003C0:	25 90 BA 45 01 EB 1F 90	BA 4D 01 EB 19 90 BA 52
00000003D0:	01 EB 13 90 BA 61 01 EB	0D 90 BA 70 01 EB 07 90
00000003E0:	BA 77 01 EB 01 90 E8 9F	FF C3 B8 00 00 B4 30 CD

Рисунок 6 – Представление плохого EXE файла

В структуре данного файла нет распределения по сегментам. С нулевого адреса располагается заголовок, после него следует таблица настройки. Сам код располагается с 300h, потому как заголовок занимает 200h, а также 100h сдвига дает директива ORG 100h.

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

0-200h занимает заголовок и таблица настроек, 200-400h стек, 400-590h сегменты данных и кода. Хороший EXE файл отличается от плохого отдельно выделенными сегментами для кода и данных, дополнительным выделением под стек 100h. Также в хорошем EXE файле не используется директива ORG 100h, потому как загрузчик сам располагает программу после PSP.

Загрузка COM модуля в основную память

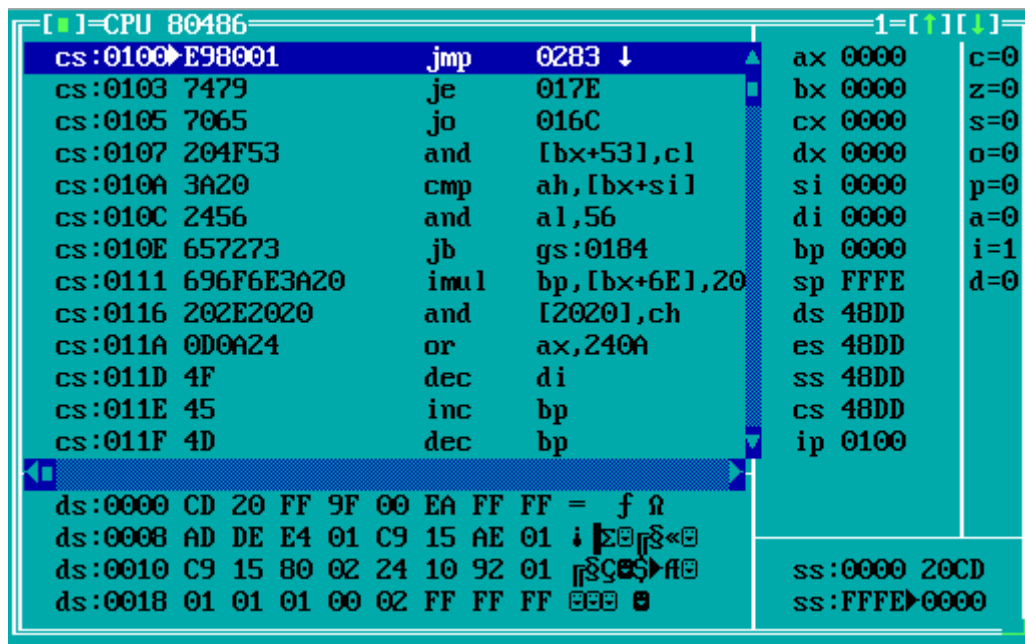


Рисунок 7 – Открытый COM файл в отладчике TD.EXE

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

Сперва выделяется сегмент памяти для загрузки модуля, далее устанавливаются все сегментные регистры на начало памяти. В первых 100h выстраивается PSP. После этого загружается содержимое COM файла и регистру IP присваивается значение 100h. Код располагается со значения CS, в нашем случае это 48DD.

2. Что располагается с адреса 0?
PSP
3. Какие значения имеют сегментные регистры? На какие области памяти они указывают?

Все сегментные регистры имеют значение 48DD и указывают на PSP.

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

Адреса стека расположены в диапазоне 0000h-FFFEh и он занимает весь сегмент COM программы. Начало стека находится в регистре SS, а его конец в SP.

Загрузка «хорошего» EXE модуля в основную память

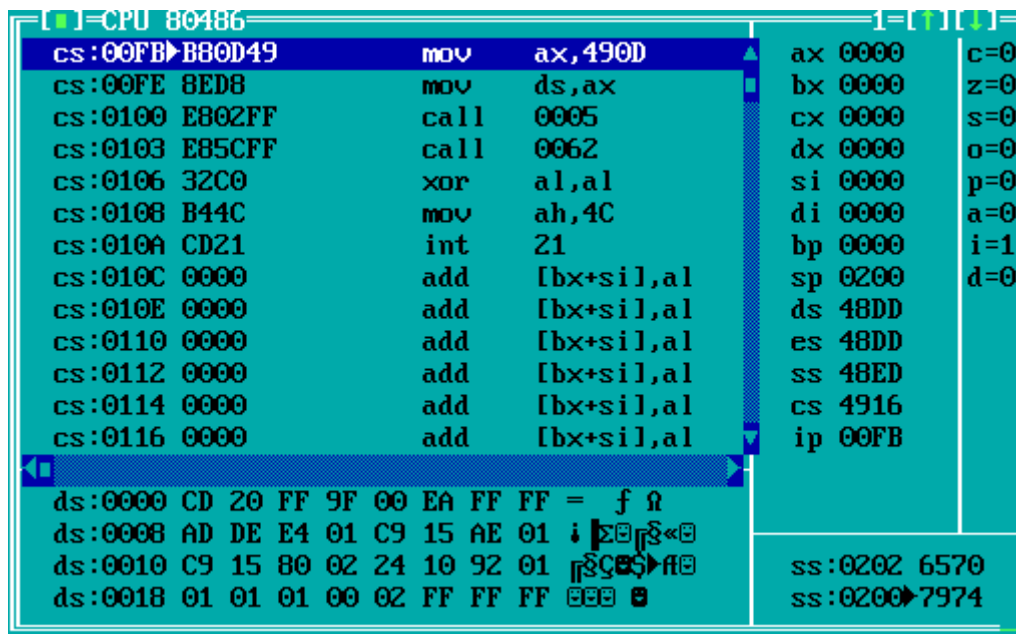


Рисунок 8 – Открытый EXE файл в отладчике TD.EXE

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

При загрузке модуля программы в память в начало программы ставится PSP. После этого этого инициализируются сегментные регистры. DS и ES равны 48DD, SS равен 48ED и указывает на начало сегмента стека, CS равен 4916 и указывает на начало сегмента команд.

2. На что указывают регистры DS и ES?

На начало PSP.

3. Как определяется стек?

Стек определяется директивой STACK. На начало стека указывает регистр SS, а на его конец регистр SP.

4. Как определяется точка входа?

Вход определяется директивой END. После этой директивы указывается метка, в которую переходит программа.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл I1.asm:

```
STACK SEGMENT STACK
    DW 100h DUP(?)
STACK ENDS

DATA SEGMENT
TypeOS db 'type OS: $'
VERS db 'Version:  .  ',0DH,0AH,'$'
OEM db 'OEM:      ',0DH,0AH,'$'
USR_NUM db 'Serial number user: ', '$'
ENDSTR db 0DH,0AH,'$'

PC db 'PC',0DH,0AH,'$'
PC_XT db 'PC/XT',0DH,0AH,'$'
AT db 'AT',0DH,0AH,'$'
PS2_30 db 'PS2 model 30',0DH,0AH,'$'
PS2_80 db 'PS2 model 80',0DH,0AH,'$'
```

```

PCjr db 'PCjr',0DH,0AH,'$'
PC_Conv db 'PC Convertible',0DH,0AH,'$'
DATA ENDS

```

```

CODE SEGMENT

```

```

    ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:STACK

```

```

; Вывод на консоль

```

```

Write PROC near

```

```

    mov AH,09h

```

```

    int 21h

```

```

    ret

```

```

Write ENDP

```

```

; Запись и вывод типа системы

```

```

TYPE_OS PROC near

```

```

    mov dx, OFFSET TypeOS

```

```

    call Write

```

```

    mov ax,0F000h

```

```

    mov es,ax

```

```

    mov ax,es:0FFFEh

```

```

; Сравниваем ключи для определения типа системы

```

```

    cmp al,0FFh

```

```

        je PCpt

```

```

    cmp al,0FEh

```

```

        je PC_XTpt

```

```

    cmp al,0FBh

```

```

        je PC_XTpt

```

```

    cmp al,0FCh

```

```

        je ATpt

```

```

    cmp al,0FAh

```

```

        je PS2_30pt

```

```

    cmp al,0F8h

```

```

        je PS2_80pt

```

```

    cmp al,0FDh

```

```

        je PCjrpt

```

```

    cmp al,0F9h

```

```

        je PC_Convpt

```

```

PCpt:

```

```

    mov dx, OFFSET PC

```

```

    jmp close

```

```

PC_XTpt:

```

```

        mov dx, OFFSET PC_XT
        jmp close
ATpt:
        mov dx, OFFSET AT
        jmp close
PS2_30pt:
        mov dx, OFFSET PS2_30
        jmp close
PS2_80pt:
        mov dx, OFFSET PS2_80
        jmp close
PCjrpt:
        mov dx, OFFSET PCjr
        jmp close
PC_Convpt:
        mov dx, OFFSET PC_Conv
        jmp close

close:
        call Write
        ret
TYPE_OS ENDP

; Запись Version и OEM
VERSION PROC near
        ; Получаем номер версии и модификации версии
системы
        mov ax,0
        mov ah,30h
        int 21h

        ; Запись версии системы в строку VERS
        mov si,offset VERS
        add si,9
        call BYTE_TO_DEC

        ; Запись модификации в строку VERS
        mov al,ah
        add si,3
        call BYTE_TO_DEC

        ; Вывод полной версии в консоль
        mov dx,offset VERS
        call Write

```

```

; Вывод OEM
mov si,offset OEM
add si,7
mov al,bh
call BYTE_TO_DEC
mov dx,offset OEM
call Write

; Вывод серийного номера юзера
mov dx,offset USR_NUM
call Write
mov al,bl
call HELP_BYTE_TO_HEX
mov al,ch
call HELP_BYTE_TO_HEX
mov al,cl
call HELP_BYTE_TO_HEX

mov dx,offset ENDSTR
call Write

ret
VERSION ENDP

HELP_BYTE_TO_HEX PROC near
call BYTE_TO_HEX
mov bx,ax
mov dl,bl
mov ah,02h
int 21h
mov dl, bh
int 21h
HELP_BYTE_TO_HEX ENDP

TETR_TO_HEX PROC near
and AL,0Fh
cmp AL,09
jbe NEXT
add AL,07
NEXT: add AL,30h
ret
TETR_TO_HEX ENDP

```

```

; Байт AL переводится в два символа шестн. числа AX
BYTE_TO_HEX PROC near
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX
    pop CX
    ret
BYTE_TO_HEX ENDP

```

```

; Перевод в 10сс, SI - адрес поля младшей цифры
BYTE_TO_DEC PROC near
    push CX
    push DX
    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 AL,00h
        je end_1
    or AL,30h
    mov [SI],AL
end_1: pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

```

```

BEGIN:
    mov ax,DATA
    mov ds,ax
    call TYPE_OS
    call VERSION
    xor AL,AL

```

```

        mov AH,4Ch
        int 21H
CODE ENDS
END BEGIN

```

Файл lcom.asm:

```

TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
TypeOS db 'type OS: $'
VERS db 'Version:  .  ',0DH,0AH,'$'
OEM db 'OEM:      ',0DH,0AH,'$'
USR_NUM db 'Serial number user: ', '$'
ENDSTR db 0DH,0AH,'$'

PC db 'PC',0DH,0AH,'$'
PC_XT db 'PC/XT',0DH,0AH,'$'
AT db 'AT',0DH,0AH,'$'
PS2_30 db 'PS2 model 30',0DH,0AH,'$'
PS2_80 db 'PS2 model 80',0DH,0AH,'$'
PCjr db 'PCjr',0DH,0AH,'$'
PC_Conv db 'PC Convertible',0DH,0AH,'$'

; Вывод на консоль
Write PROC near
    mov AH,09h
    int 21h
    ret
Write ENDP

; Запись и вывод типа системы
TYPE_OS PROC near
    mov dx, OFFSET TypeOS
    call Write
    mov ax,0F000h
    mov es,ax
    mov ax,es:0FFFEh

    ; Сравниваем ключи для определения типа системы
    cmp al,0FFh
    je PCpt

```

```

cmp al,0FEh
    je PC_XTpt
cmp al,0FBh
    je PC_XTpt
cmp al,0FCh
    je ATpt
cmp al,0FAh
    je PS2_30pt
cmp al,0F8h
    je PS2_80pt
cmp al,0FDh
    je PCjrpt
cmp al,0F9h
    je PC_Convpt

```

```

PCpt:
    mov dx, OFFSET PC
    jmp close

```

```

PC_XTpt:
    mov dx, OFFSET PC_XT
    jmp close

```

```

ATpt:
    mov dx, OFFSET AT
    jmp close

```

```

PS2_30pt:
    mov dx, OFFSET PS2_30
    jmp close

```

```

PS2_80pt:
    mov dx, OFFSET PS2_80
    jmp close

```

```

PCjrpt:
    mov dx, OFFSET PCjr
    jmp close

```

```

PC_Convpt:
    mov dx, OFFSET PC_Conv
    jmp close

```

```

close:
    call Write
    ret

```

```

TYPE_OS ENDP

```

```

; Запись Version и OEM
VERSION PROC near

```

```
    ; Получаем номер версии и модификации версии  
системы
```

```
    mov ax,0  
    mov ah,30h  
    int 21h
```

```
    ; Запись версии системы в строку VERS  
    mov si,offset VERS  
    add si,9  
    call BYTE_TO_DEC
```

```
    ; Запись модификации в строку VERS  
    mov al,ah  
    add si,3  
    call BYTE_TO_DEC
```

```
    ; Вывод полной версии в консоль  
    mov dx,offset VERS  
    call Write
```

```
    ; Вывод OEM  
    mov si,offset OEM  
    add si,7  
    mov al,bh  
    call BYTE_TO_DEC  
    mov dx,offset OEM  
    call Write
```

```
    ; Вывод серийного номера юзера  
    mov dx,offset USR_NUM  
    call Write  
    mov al,bl  
    call HELP_BYTE_TO_HEX  
    mov al,ch  
    call HELP_BYTE_TO_HEX  
    mov al,cl  
    call HELP_BYTE_TO_HEX
```

```
    mov dx,offset ENDSTR  
    call Write
```

```
    ret  
VERSION ENDP
```



```

HELP_BYTE_TO_HEX PROC near
    call BYTE_TO_HEX
    mov bx,ax
    mov dl,bl
    mov ah,02h
    int 21h
    mov dl, bh
    int 21h
HELP_BYTE_TO_HEX ENDP

```

```

TETR_TO_HEX PROC near
    and AL,0Fh
    cmp AL,09
        jbe NEXT
    add AL,07
NEXT: add AL,30h
    ret
TETR_TO_HEX ENDP

```

; Байт AL переводится в два символа шестн. числа AX

```

BYTE_TO_HEX PROC near
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX
    pop CX
    ret
BYTE_TO_HEX ENDP

```

; Перевод в 10сс, SI - адрес поля младшей цифры

```

BYTE_TO_DEC PROC near
    push CX
    push DX
    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 AL,00h
            je end_1
        or AL,30h
        mov [SI],AL
end_1:  pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

BEGIN:
        call TYPE_OS
        call VERSION
        xor AL,AL
        mov AH,4Ch
        int 21H
TESTPC ENDS
END START

```