

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 моделей

Студент гр.0382

Литягин С.М.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

Задание.

1. Написать текст исходного .COM модуля, который определяет тип РС и версию системы. Результатом будет “хороший” .COM модуль, а также “плохой” .EXE, полученный из исходного текста для .COM модуля.

2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1. Результатом будет “хороший” .EXE.

3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы “Отличия исходных текстов COM и EXE программ”.

4. Запустить FAR и открыть файл загрузочного модуля .COM и файл “плохого” .EXE в шестнадцатеричном виде. Затем открыть файл загрузочного модуля “хорошего” .EXE и сравнить его с предыдущими файлами. Ответить на контрольные вопросы “Отличия форматов файлов COM и EXE модулей”.

5. Открыть отладчик TD.EXE и загрузить .COM. Ответить на контрольные вопросы “Загрузка COM модуля в основную память”. Представить в отчете план загрузки .COM модуля в основную память.

6. Открыть отладчик TD.EXE и загрузить “хороший” .EXE. Ответить на контрольные вопросы “Загрузка “хорошего” EXE модуля в основную память”.

7. Оформить отчет в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Выполнение работы:

1. За основу был взят шаблон .COM модуля из методического пособия, в котором реализованы процедуры преобразования двоичных кодов в символы шестнадцатеричных и десятичных чисел. Для определения типа PC и версии системы были написаны процедуры: IBM_TYPE, DOS_VER, OEMN, USERN. Тип IBM PC был получен согласно байту по адресу 0F000:0FFFFh. Для определения версии системы же использовалась функция 30H прерывания 21H. Ее выходными параметрами являются: AL – номер основной версии, AH – номер модификации, BH – серийный номер OEM, BL:CH – 24-битовый серийный номер пользователя.

В результате шага имеем “хороший” .COM модуль и “плохой” .EXE модуль. Выводы, полученные при их запуске, представлены на рисунке 1 и рисунке 2 соответственно.

```
C:\>lb1_com
IBM type: AT
MS DOS version: 5.0
OEM number:255
User_s number: 000000h
```

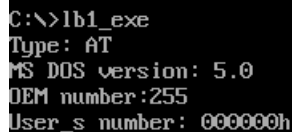
Рисунок 1 – результат запуска модуля lb1_com.com

```
C:\>lb1_com.exe

e:          5 0          0 IBM type:          0 IBM typ
          255          0 IBM type:          0 IBM typ
          000000 0 IBM type:          0 IBM typ
C:\>_
```

Рисунок 2 – результат запуска модуля lb1_com.exe

2. Для написания “хорошего” .EXE модуля разобьем программу на сегменты кода, данных и стека, также добавим главную процедуру. В .COM модуле имелась директива org 100h, что нужна, поскольку при загрузке COM модуля в память DOS первые 256 байт блоком данных занимает PSP, код программы располагается лишь после этого блока. В .EXE модуле же мы в этом не нуждаемся, поскольку блок PSP расположен вне сегмента кода. На рисунке 3 представлены результаты запуска данного модуля.



```
C:\>lb1_exe
Type: AT
MS DOS version: 5.0
OEM number: 255
User_s number: 000000h
```

Рисунок 3 – результат запуска модуля lb1_exe.exe

3. Сравнивая исходные тексты для .COM и .EXE модулей, ответим на контрольные вопросы “Отличия исходных текстов COM и EXE программ”:

- Сколько сегментов должна содержать COM-программа?

Один сегмент (содержит и код, и данные; стек же генерируется сам).

- EXE-программа?

Обязательно один – сегмент кода. Также можно добавить сегмент данных и стека, они описываются отдельно друг от друга.

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

Обязательно нужна директива `ORG 100h`, поскольку первые 256 байт занимает блок данных PSP. Поэтому нужно, чтобы адресация имела смещение в 256 байт от нулевого адреса. Также директива `ASSUME` указывает, с каким сегментом нужно связать регистры сегмента кода и сегмента данных.

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

Поскольку сегментные регистры в .COM-программе определяются в момент запуска программы, а не при линковке (не имеет relocation table), то мы не можем использовать команды с указанием сегментов.

4. Рассматривая шестнадцатеричные виды модулей, ответим на контрольные вопросы “Отличия форматов файлов COM и EXE модулей”:

- Какова структура файла COM? С какого адреса располагается код?

.COM модуль состоит из одного сегмента, содержащего и код, и данные. Код располагается с нулевого адреса. Как уже упоминалось ранее, PSP занимает 100h байт, поэтому устанавливается смещение 100h. Модуль в шестнадцатеричном виде представлен на рисунке 4.

начало сегмента →	0000000000: E9 09 02 49 42 4D 20 74 79 70 65 3A 20 20 0D 0A йоIBM type: ь
	0000000010: 24 49 42 4D 20 74 79 70 65 3A 20 50 43 0D 0A 24 \$IBM type: PCь
	0000000020: 49 42 4D 20 74 79 70 65 3A 20 50 43 2F 58 54 0D IBM type: PC/XTь
	0000000030: 0A 24 49 42 4D 20 74 79 70 65 3A 20 41 54 0D 0A \$IBM type: ATь
	0000000040: 24 49 42 4D 20 74 79 70 65 3A 20 50 53 32 20 6D \$IBM type: PS2 m
	0000000050: 6F 64 65 6C 20 33 30 0D 0A 24 49 42 4D 20 74 79 odel 30ь\$IBM ty
	0000000060: 70 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 35 30 pe: PS2 model 50
	0000000070: 20 6F 72 20 36 30 0D 0A 24 49 42 4D 20 74 79 70 or 60ь\$IBM typ
	0000000080: 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 38 30 0D e: PS2 model 80ь
	0000000090: 0A 24 49 42 4D 20 74 79 70 65 3A 20 50 43 6A 72 \$IBM type: PCjr
	00000000A0: 0D 0A 24 49 42 4D 20 74 79 70 65 3A 20 50 43 20 ь\$IBM type: PC
	00000000B0: 43 6F 6E 76 65 72 74 69 62 6C 65 0D 0A 24 4D 53 Convertibleь\$MS
	00000000C0: 20 44 4F 53 20 76 65 72 73 69 6F 6E 3A 20 20 2E DOS version: .
	00000000D0: 20 20 0D 0A 24 4F 45 4D 20 6E 75 6D 62 65 72 3A ь\$OEM number:
	00000000E0: 20 20 20 0D 0A 24 55 73 65 72 5F 73 20 6E 75 6D ь\$User_s num
	00000000F0: 62 65 72 3A 20 20 20 20 20 20 20 68 0D 0A 24 24 ber: ь\$
	0000000100: 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF FF о<ov0♦♦0ГQьаипя
	0000000110: 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9 †Д±♦ТиияYГСьльий
	0000000120: FF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88 я€%O€♦Oь3иЮя€%O€
	0000000130: 05 5B C3 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA ♦[ГQR2д3TW чсЪК
	0000000140: 30 88 14 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C 0€JN3T= sc< т♦Q
	0000000150: 30 88 04 5A 59 C3 50 B4 09 CD 21 58 C3 53 52 06 0€♦ZYГProH!XГSR♦
	0000000160: BB 00 F0 8E C3 26 A0 FE FF 3C FF 74 2A 3C FE 74 » рЪГ& юя<ят*<ют
	0000000170: 2C 3C FB 74 28 3C FC 74 2A 3C FA 74 2C 3C F8 74 ,<yt(<ьт*<ьт,<шт
	0000000180: 2E 3C FD 74 30 3C F9 74 32 BF 0D 01 E8 7B FF 89 .<эт0<щт2iь0и{я%
	0000000190: 05 BA 03 01 EB 28 90 BA 11 01 EB 22 90 BA 20 01 ♦еV0л(ђе<0л"ђе 0
	00000001A0: EB 1C 90 BA 32 01 EB 16 90 BA 41 01 EB 10 90 BA лLђе20л"ђеA0л"ђе
	00000001B0: 79 01 EB 0A 90 BA 92 01 EB 04 90 BA A3 01 E8 95 у0л"ђе'0л♦ђеJ0и♦
	00000001C0: FF 07 5A 58 C3 50 53 51 33 C0 B4 30 CD 21 BE CE я♦ZXGPSQ3Ar0H!sO
	00000001D0: 01 E8 5F FF 8A C4 83 C6 03 E8 57 FF BA BE 01 E8 0и_ялДfЖвиWяes0и
	00000001E0: 74 FF BE E2 01 8A C7 E8 49 FF BA D5 01 E8 66 FF тясв0ь3иIяеX0ифя
	00000001F0: BF FA 01 8B C1 E8 23 FF 8A C3 E8 0D FF BF F5 01 ьь0<Би#ялГиJяix0
	0000000200: 89 05 BA E6 01 E8 4E FF 59 5B 58 C3 B8 00 F0 8E %♦еж0иNяY[ХГё рЪ
	0000000210: C0 26 A0 FF FF E8 45 FF E8 AA FF 32 C0 B4 4C CD A& яяиЕяиЕя2ArLH
конец сегмента →	0000000220: 21 !

Рисунок 4 – “хороший” .COM в 16-м виде

- Какова структура файла “плохого” EXE? С какого адреса располагается код? Что располагается с адреса 0?

Также как и .COM модуль, состоит из одного сегмента. Код располагается с адреса 300h (200h занимает заголовок и relocation table, 100h - смещение). Модуль в шестнадцатеричном виде представлен на рисунке 5.



Рисунок 5 – “плохой” .EXE в 16-м виде

- Какова структура файла “хорошего” EXE? Чем он отличается от файла “плохого” EXE?

Имеет несколько сегментов, в начале модуля располагается заголовок и relocation table (200h байт), затем сегменты в порядке их определения в коде, т.е. сначала сегмент стека, потом сегмент данных, а затем – сегмент кода. Модуль в шестнадцатеричном виде представлен на рисунке 6.

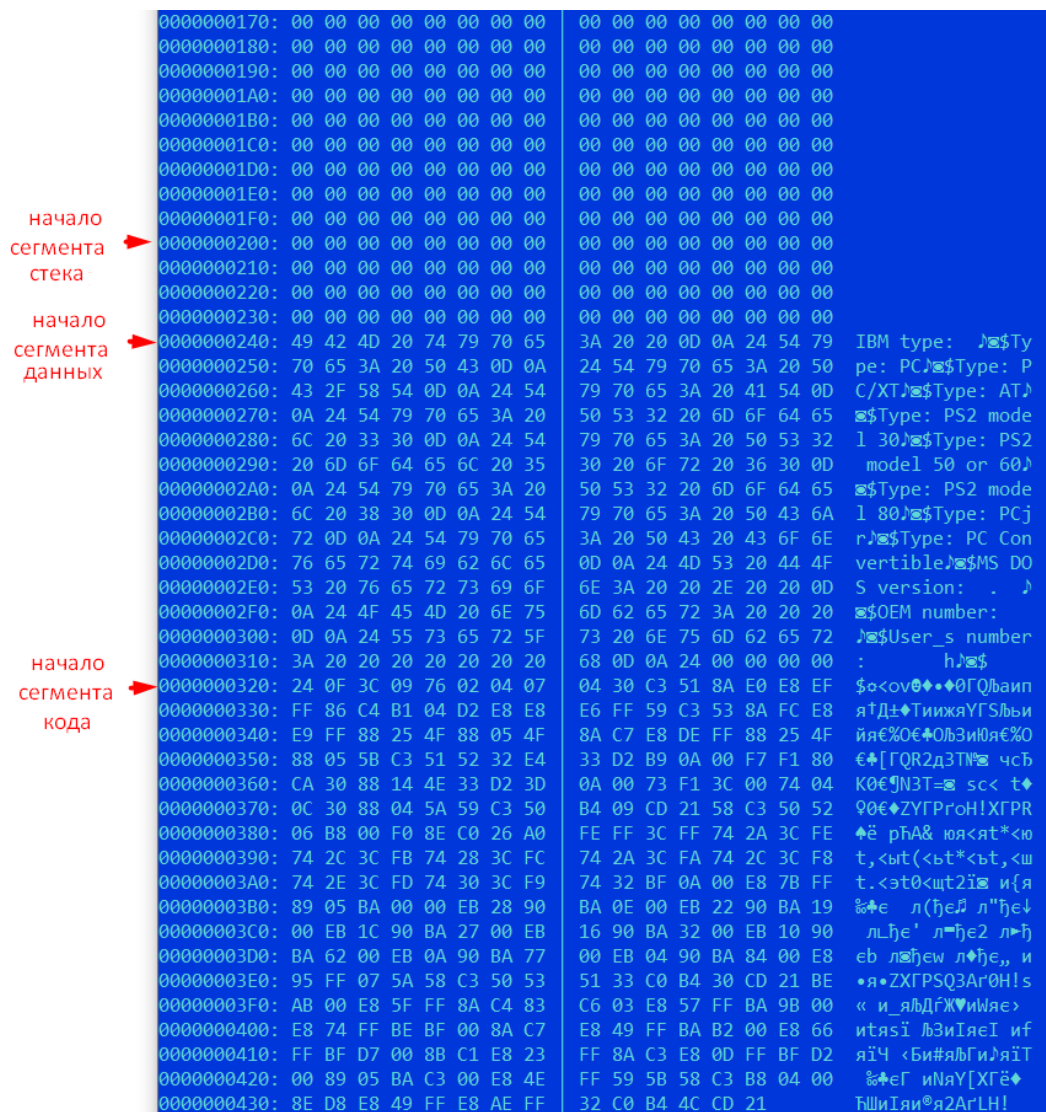


Рисунок 6 – “хороший” .EXE в 16-м виде

5. Загружая .COM модуль в отладчике TD.EXE, ответим на контрольные вопросы “Загрузка COM модуля в основную память”. Результат загрузки программы в отладчик представлены на рисунке 7.

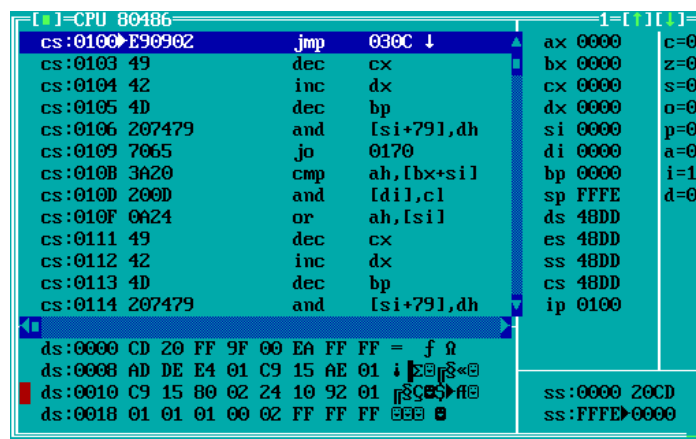


Рисунок 7 – загруженный в отладчик “хороший” .COM модуль

- Какой формат загрузки модуля COM? С какого адреса располагается код?

Сначала определяется сегментный адрес участка ОП, у которого достаточно места для загрузки программы. Затем создается блок памяти для PSP и программы. Начиная с PSP:0100h помещается считанный образ COM-файла. Код же располагается с адреса CS:0100h.

- Что располагается с адреса 0?

Сегмент PSP.

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

Сегментные регистры CS, DS, ES, SS указывают на PSP (в данном случае имеет значение 48DDh)

- Как определяется стек? Какую область памяти он занимает? Какие адреса?

Для .COM файлов стек определяется от FFFh до 0000h, т.е. весь сегмент. SS указывает на начало сегмента (т.е. на 48DDh); SP – на последний адрес, что кратен двум (т.е. на FFFh). Соответственно, адреса 48DD:0000h – 48DD:FFFh.

6. Загружая “хороший” .EXE модуль в отладчике TD.EXE, ответим на контрольные вопросы “Загрузка “хорошего” EXE модуля в основную память”. Результат загрузки программы в отладчик представлены на рисунке 8.

The screenshot shows the CPU 80486 window of a debugger. The main pane displays assembly instructions with their addresses and hex values. The instruction list is as follows:

Address	Hex	Instruction
cs:0100	B8F148	mov ax, 48F1
cs:0110	8ED8	mov ds, ax
cs:0112	E849FF	call 005E
cs:0115	E8AEFF	call 00C6
cs:0118	32C0	xor al, al
cs:011A	B44C	mov ah, 4C
cs:011C	CD21	int 21
cs:011E	0000	add [bx+si], al
cs:0120	0000	add [bx+si], al
cs:0122	0000	add [bx+si], al
cs:0124	0000	add [bx+si], al
cs:0126	0000	add [bx+si], al
cs:0128	0000	add [bx+si], al

Below the instruction list, there is a section for data segments:

Address	Hex	Comment
ds:0000	CD 20 FF 9F 00 EA FF FF	= f 0
ds:0008	AD DE E4 01 C9 15 AE 01	1 2 3 4 5 6 7 8
ds:0010	C9 15 80 02 24 10 92 01	9 10 11 12 13 14 15 16
ds:0018	01 01 01 00 02 FF FF FF	17 18 19 20 21 22 23 24

On the right side, the register window shows the following values:

Register	Value
ax	0000
bx	0000
cx	0000
dx	0000
si	0000
di	0000
bp	0000
sp	0040
ds	48DD
es	48DD
ss	48ED
cs	48FF
ip	010D

At the bottom right, the stack window shows:

Address	Value
ss:0042	204D
ss:0040	4249

Рисунок 8 – загруженный в отладчик “хороший” .COM модуль

- Как загружается “хороший” EXE? Какие значения имеют сегментные регистры?

Загружается в память PSP. Затем, по информации в заголовке, загружается .EXE модуль. Значения регистров: DS = ES = 48DD, CS = 48FF, SS = 48ED.

- На что указывают регистры DS и ES?

Эти регистры указывают на начало PSP.

- Как определяется стек?

В коде программы описывается сегмент стека. При помощи директивы ASSUME устанавливает сегментный регистр на описанный сегмент стека. Соответственно, SS будет указывать на начало сегмента стека, а SP – на его конец.

- Как определяется точка входа?

Точка входа определяется директивой END *метка*. Данная метка определяет адрес, с которого начинается выполнение программы, т.е. – точка входа в программу.

Исходный код программы см. в приложении А.

Выводы.

В ходе работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, в структурах файлов загрузочных моделей и способов их загрузки в основную память; была написана программа, выводящую информацию о типе PC и версии системы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb1_com.asm

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

; ДАННЫЕ
NOTYPE db 'IBM type: ', 0DH, 0AH, '$'
PC db 'IBM type: PC', 0DH, 0AH, '$'
XT db 'IBM type: PC/XT', 0DH, 0AH, '$'
AT db 'IBM type: AT', 0DH, 0AH, '$'
PS230 db 'IBM type: PS2 model 30', 0DH, 0AH, '$'
PS25060 db 'IBM type: PS2 model 50 or 60', 0DH, 0AH, '$'
PS280 db 'IBM type: PS2 model 80', 0DH, 0AH, '$'
PCjr db 'IBM type: PCjr', 0DH, 0AH, '$'
PCConvertible db 'IBM type: PC Convertible', 0DH, 0AH, '$'
DOSV db 'MS DOS version: . ', 0DH, 0AH, '$'
OEM db 'OEM number: ', 0DH, 0AH, '$'
USR db 'User_s number:      h', 0DH, 0AH, '$'

; ПРОЦЕДУРЫ
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

;-----
BYTE_TO_HEX PROC near
; байт в AL переводится в два символа 16-го числа в AX
    push CX
    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL
    call TETR_TO_HEX ; в AL старшая цифра
    pop CX           ; в AH младшая
    ret
BYTE_TO_HEX ENDP

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

```

    push BX
    mov BH, AH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    dec DI
    mov AL, BH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    pop BX
    ret
WRD_TO_HEX ENDP
;-----
BYTE_TO_DEC PROC near
; перевод в 10 с/с, в SI - адрес поля младшей цифры
    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
;-----
PRINT PROC near
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
PRINT ENDP
;-----
IBM_TYPE PROC near
    push BX
    push DX
    push ES

```

```

    mov BX, 0F000h
    mov ES, BX
    mov AL, ES:[0FFFEh]

    cmp AL, 0FFh
    je _pc

    cmp AL, 0FEh
    je _xt

    cmp AL, 0FBh
    je _xt

    cmp AL, 0FCh
    je _at

    cmp AL, 0FAh
    je _ps230

    cmp AL, 0F8h
    je _ps280

    cmp AL, 0FDh
    je _pcjr

    cmp AL, 0F9h
    je _pcc

    mov DI, offset NOTYPE + 10
    call BYTE_TO_HEX
    mov [DI], AX
    mov DX, offset NOTYPE
    jmp p_ibm

_pc:
    mov DX, offset PC
    jmp p_ibm
_xt:
    mov DX, offset XT
    jmp p_ibm
_at:
    mov DX, offset AT
    jmp p_ibm
_ps230:
    mov DX, offset PS230
    jmp p_ibm
_ps280:
    mov DX, offset PS280
    jmp p_ibm
_pcjr:
    mov DX, offset PCjr

```

```

        jmp p_ibm
_pcc:
    mov DX, offset PCConvertible
p_ibm:
    call PRINT
    pop ES
    pop DX
    pop AX
    ret
IBM_TYPE ENDP
;-----
DOS_VER PROC near
    push AX
    push BX
    push CX
    xor AX, AX
    mov AH, 30h
    int 21h

    mov SI, offset DOSV + 16
    call BYTE_TO_DEC
    mov AL, AH
    add SI, 3
    call BYTE_TO_DEC
    mov DX, offset DOSV
    call PRINT

    mov SI, offset OEM + 13
    mov AL, BH
    call BYTE_TO_DEC
    mov DX, offset OEM
    call PRINT

    mov DI, offset USR + 20
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    mov di, offset USR + 15
    mov [DI], AX
    mov DX, offset USR
    call PRINT
    pop CX
    pop BX
    pop AX
    ret
DOS_VER ENDP
;-----
; КОД
BEGIN:
    mov AX, 0F000h

```

```

    mov ES, AX
    mov AL, ES:[0FFFFh]
    call IBM_TYPE
    call DOS_VER
    xor AL, AL
    mov AH, 4Ch
    int 21h
TESTPC ENDS
    END START

```

Название файла: lbl_exe.asm

```

AStack SEGMENT STACK
    DW 32 DUP(?)
AStack ENDS

```

```

DATA SEGMENT
    NOTYPE db 'IBM type: ', 0DH, 0AH, '$'
    PC db 'Type: PC', 0DH, 0AH, '$'
    XT db 'Type: PC/XT', 0DH, 0AH, '$'
    AT db 'Type: AT', 0DH, 0AH, '$'
    PS230 db 'Type: PS2 model 30', 0DH, 0AH, '$'
    PS25060 db 'Type: PS2 model 50 or 60', 0DH, 0AH, '$'
    PS280 db 'Type: PS2 model 80', 0DH, 0AH, '$'
    PCjr db 'Type: PCjr', 0DH, 0AH, '$'
    PCConvertible db 'Type: PC Convertible', 0DH, 0AH, '$'
    DOSV db 'MS DOS version: . ', 0DH, 0AH, '$'
    OEM db 'OEM number: ', 0DH, 0AH, '$'
    USR db 'User_s number:      h', 0DH, 0AH, '$'
DATA ENDS

```

```

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

```

; ПРОЦЕДУРЫ

```

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

```

;-----

```

BYTE_TO_HEX PROC near

```

; байт в AL переводится в два символа 16-го числа в AX

```

    push CX
    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL

```

```

        call TETR_TO_HEX ; в AL старшая цифра
        pop CX           ; в AH младшая
        ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, в DI - адрес последнего символа
        push BX
        mov BH, AH
        call BYTE_TO_HEX
        mov [DI], AH
        dec DI
        mov [DI], AL
        dec DI
        mov AL, BH
        call BYTE_TO_HEX
        mov [DI], AH
        dec DI
        mov [DI], AL
        pop BX
        ret
WRD_TO_HEX ENDP
;-----
BYTE_TO_DEC PROC near
; перевод в 10 с/с, в SI - адрес поля младшей цифры
        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
;-----
PRINT PROC near
        push AX
        mov AH, 09h
        int 21h

```

```

        pop AX
        ret
PRINT ENDP
;-----
IBM_TYPE PROC near
    push AX
    push DX
    push ES
    mov AX, 0F000h
    mov ES, AX
    mov AL, ES:[0FFFEh]

    cmp AL, 0FFh
    je _pc

    cmp AL, 0FEh
    je _xt

    cmp AL, 0FBh
    je _xt

    cmp AL, 0FCh
    je _at

    cmp AL, 0FAh
    je _ps230

    cmp AL, 0F8h
    je _ps280

    cmp AL, 0FDh
    je _pcjr

    cmp AL, 0F9h
    je _pcc

    mov DI, offset NOTYPE + 10
    call BYTE_TO_HEX
    mov [DI], AX
    mov DX, offset NOTYPE
    jmp p_ibm

_pc:
    mov DX, offset PC
    jmp p_ibm
_xt:
    mov DX, offset XT
    jmp p_ibm
_at:
    mov DX, offset AT
    jmp p_ibm

```



```

_ps230:
    mov DX, offset PS230
    jmp p_ibm
_ps280:
    mov DX, offset PS280
    jmp p_ibm
_pcjr:
    mov DX, offset PCjr
    jmp p_ibm
_pcc:
    mov DX, offset PCConvertible
p_ibm:
    call PRINT
    pop ES
    pop DX
    pop AX
    ret
IBM_TYPE ENDP
;-----
DOS_VER PROC near
    push AX
    push BX
    push CX
    xor AX, AX
    mov AH, 30h
    int 21h

    mov SI, offset DOSV + 16
    call BYTE_TO_DEC
    mov AL, AH
    add SI, 3
    call BYTE_TO_DEC
    mov DX, offset DOSV
    call PRINT

    mov SI, offset OEM + 13
    mov AL, BH
    call BYTE_TO_DEC
    mov DX, offset OEM
    call PRINT

    mov DI, offset USR + 20
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    mov di, offset USR + 15
    mov [DI], AX
    mov DX, offset USR
    call PRINT
    pop CX

```

```

        pop BX
        pop AX
        ret
DOS_VER ENDP
;-----
; КОД
MAIN PROC far
        mov ax, data
        mov ds, ax
        call IBM_TYPE
        call DOS_VER
; Выход в DOS
        xor AL, AL
        mov AH, 4Ch
        int 21h
MAIN ENDP
TESTPC ENDS
END MAIN

```