

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

ОТЧЕТ
по практической работе №1
по дисциплине «Операционные системы»
Тема: Исследование структур загрузочных модулей

Студент гр. 9382

Михайлов Д.А.

Преподаватель

Губкин А.Ф.

Санкт-Петербург

2021

Цель работы.

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

Необходимые сведения для составления программы.

Тип IBM PC хранится в байте по адресу 0F000:0FFFE, в предпоследнем байте ROM BIOS. Соответствие кода и типа в Таблице 1:

Таблица 1 — соответствие кодов типа компьютера

PC	FF
PC/XT	FE,FB
AT	FC
PS2 модель 30	FA
PS2 модель 50 или 60	FC
PS2 модель 80	F8
PCjr	FD
PC Convertible	F9

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH:

MOV AH,30h

INT 21h

Таблица 2 - Выходные параметры функции 30h:

AL	Major номер версии. 0 => <2.0
AH	Minor номер версии
BH	Номер OEM (Original Equipment Manufacturer)
BL:CH	Серийный номер (24 бита)

Постановка задачи.

Требуется реализовать текст исходного .COM модуля, который определяет тип РС и версию системы. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип РС и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения. Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx - номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM (Original Equipment Manufacturer) и серийным номером пользователя. Полученные строки выводятся на экран.

Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить и сравнить исходные тексты для .COM и .EXE модулей.

Процедуры используемые в программе.

TETR_TO_HEX – Перевода половины байта в шестнадцатеричную систему счисления.

BYTE_TO_HEX – Перевода регистра AL в шестнадцатеричную систему счисления, результат в AX.

WRD_TO_HEX – Перевода регистра AX в шестнадцатеричную систему счисления, результат в регистр DI.

BYTE_TO_DEC – Перевода регистра AL в десятичную систему счисления, результат в SI.

TYPE_PC – Определяет тип IBM PC.

MS_DOS_OEM_SERIAL – Определяет версию MS DOS, номер OEM и серийный номер пользователя.

WriteMsg – Вывод на экран.

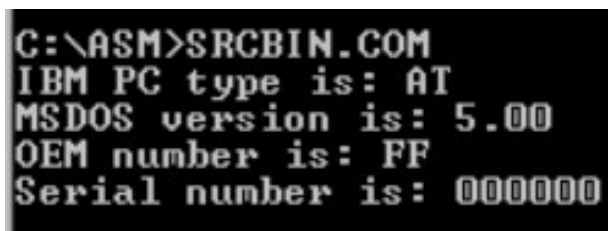
Структуры данных.

Таблица 3 – Структуры данных

Название поля данных	Тип	Назначение
T_PC	db	PC
T_PC_XT	db	PC/XT
T_AT	db	AT
T_PS2_30	db	PS2 модель 30
T_PS2_5060	db	PS2 модель 50 или 60
T_PS2_80	db	PS2 модель 80
T_PCJR	db	PCjr
T_PC_CONVERTIBLE	db	PC Convertible
T_PC_UNKNOWN	db	Неизвестный тип PC
IBM_PC	db	Тип IBM PC
MS_DOS_VERSION	db	Номер версии MS DOS
OEM	db	Серийный номер OEM
SERIAL	db	Серийный номер пользователя

Ход работы.

Шаг 1. Запуск «хорошего» .COM модуля.



```
C:\ASM>SRCBIN.COM
IBM PC type is: AT
MSDOS version is: 5.00
OEM number is: FF
Serial number is: 000000
```

Рисунок 1 – «Хороший» .COM модуль

Шаг 2. Запуск «плохого» .EXE модуля.

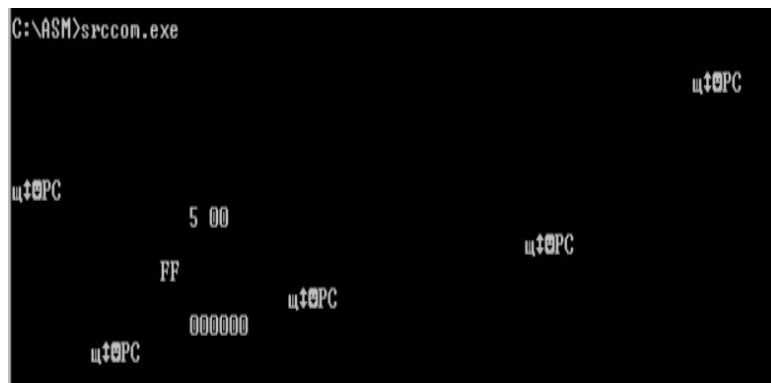


Рисунок 2 – «Плохой» .EXE модуль

Шаг 3. Запуск «хорошего» .EXE модуля.

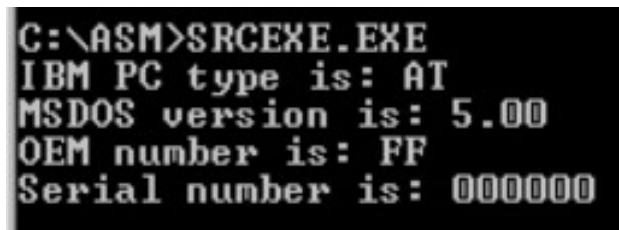


Рисунок 3 – «Хороший» .EXE модуль

Шаг 4. Ответы на контрольные вопросы. Отличия исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM-программа?

Один.

2) EXE программа?

Один и более.

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

Директива ORG 100h (смещение 100h) для 256 байт (100h) блока данных PSP. Директива ASSUME, ставящая в соответствие начало программы сегментам кода и данных.

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

Нет, не все, так как в отличие от EXE-программы, в которой существует таблица настроек (таблица разметки), называемая Relocation Table, COM-программа ею не располагает. Адреса сегментов определяются

загрузчиком в момент запуска программы на основе информации о местоположении полей адресов в файле из Relocation Table. Следовательно, в связи с отсутствием этой таблицы в COM-программах, команды вида mov [регистр], seg [сегмент] недопустимы.

Шаг 4. .COM модуль в шестнадцатеричном виде.

```

00000000: E9 12 02 50 43 0D 0A 24 50 43 2F 58 54 0D 0A 24 et0PCJ0$PC/XTJ0$
00000010: 41 54 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 33 ATJ0$PS2 model 3
00000020: 30 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 35 30 0J0$PS2 model 50
00000030: 2F 36 30 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 /60J0$PS2 model
00000040: 38 30 0D 0A 24 50 43 4A 52 0D 0A 24 50 43 20 63 80J0$PCJRJ0$PC c
00000050: 6F 6E 76 65 72 74 69 62 6C 65 0D 0A 24 55 4E 4B onvertibleJ0$UNK
00000060: 4E 4F 57 4E 3A 20 20 20 0D 0A 24 49 42 4D 20 50 NOWN: J0$IBM P
00000070: 43 20 74 79 70 65 20 69 73 3A 20 24 4D 53 44 4F C type is: $MSDO
00000080: 53 20 76 65 72 73 69 6F 6E 20 69 73 3A 20 23 2E S version is: #.
00000090: 23 0D 0A 24 4F 45 4D 20 6E 75 6D 62 65 72 20 69 #J0$OEM number i
000000A0: 73 3A 20 23 23 0D 0A 24 53 65 72 69 61 6C 20 6E s: ##J0$Serial n
000000B0: 75 6D 62 65 72 20 69 73 3A 20 23 23 23 23 23 23 umber is: #####
000000C0: 0D 0A 24 24 0F 3C 09 76 02 04 07 04 30 C3 51 8A J0$*$<0v0*+0AQS
000000D0: E0 E8 EF FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 aeiytA+0ee?yVAS
000000E0: 8A FC E8 E9 FF 88 25 4F 88 05 4F 8A C7 E8 DE FF Sueey?%0?AOSCe?y
000000F0: 88 25 4F 88 05 5B C3 51 52 32 E4 33 D2 B9 0A 00 ?%0?AIAQR2a30?0
00000100: F7 F1 80 CA 30 88 14 4E 33 D2 3D 0A 00 73 F1 3C ?n?E0?4N30=0 sn<
00000110: 00 74 04 0C 30 88 04 5A 59 C3 B4 09 CD 21 C3 50 t00?0ZYA?OI!AP
00000120: 53 51 06 BA 6B 01 E8 F1 FF 32 E4 B9 00 F0 8E C1 SQ*?k0eny2a? ?ZA
00000130: 26 A0 FE FF 3D FF 00 74 2E 3D FE 00 74 2F 3D FB & ?y=y t.=? t/=u
00000140: 00 74 2A 3D FC 00 74 2B 3D FA 00 74 2C 3D F6 00 t*=u t+=u t.=o
00000150: 74 2D 3D F8 00 74 2E 3D FD 00 74 2F 3D F9 00 74 t-=o t.=y t/=u t
00000160: 30 BA 5D 01 EB 31 90 BA 03 01 EB 2B 90 BA 08 01 0?l0e1??0e+??00
1Help 2Dump 3Quit 4Text 5 6Edit 7Search 8ANSI 9 10Quit

```

Рисунок 4 - .COM модуль в шестнадцатеричном виде

«Плохой» .EXE модуль в шестнадцатеричном в

```

00000000: 4D 5A 26 01 03 00 00 00 20 00 00 00 FF FF 00 00 MZ&0w yy
00000010: 00 00 00 00 00 01 00 00 1E 00 00 00 01 00 00 00
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
1Help 2Dump 3Quit 4Text 5 6Edit 7Search 8ANSI 9 10Quit

```

Рисунок 5 - «Плохой» .EXE модуль в шестнадцатеричном виде

«Хороший» .EXE модуль в шестнадцатеричном виде.

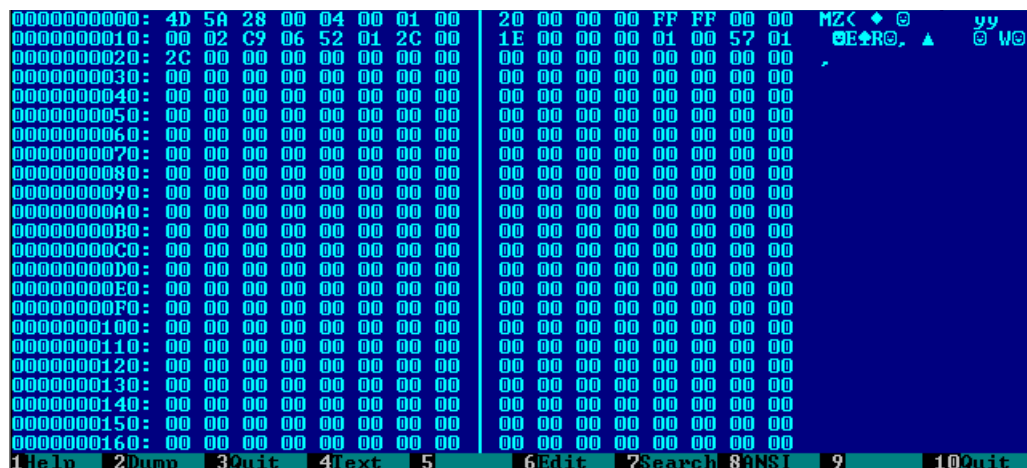


Рисунок 6 - «Хороший» .EXE модуль в шестнадцатеричном виде

Ответы на контрольные вопросы. Отличия форматов файлов COM и EXE программ.

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

COM файл состоит из одного сегмента и содержит данные и машинные команды. Код начинается с адреса 0h, но при загрузке модуля устанавливается смещение в 100h.

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

В «плохом» EXE файле данные и код содержатся в одном сегменте. Код располагается с адреса 300h. С адреса 0h располагается Relocation Table (таблица разметки).

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

В «хорошем» файле EXE содержится информация для загрузчика, сегмент стека, сегмент данных и сегмент кода (3 сегмента вместо одного в «плохом» .EXE). Код располагается с адреса 200h в отличии от 300h в «плохом» .EXE файле.

Шаг 5. Загрузка COM модуля в основную память.

The screenshot shows a DOS debugger window with the following content:

CPU 80486		1=[]	
cs:0100	E91202	jmp	0315 ↓
cs:0103	50	push	ax
cs:0104	43	inc	bx
cs:0105	0D0A24	or	ax,240A
cs:0108	50	push	ax
cs:0109	43	inc	bx
cs:010A	2F	das	
cs:010B	58	pop	ax
cs:010C	54	push	sp
cs:010D	0D0A24	or	ax,240A
cs:0110	41	inc	cx
cs:0111	54	push	sp
cs:0112	0D0A24	or	ax,240A
cs:0115	50	push	ax
cs:0116	53	push	bx

ax	0000	c=0
bx	0000	z=0
cx	0000	s=0
dx	0000	o=0
si	0000	p=0
di	0000	a=0
bp	0000	i=1
sp	FFFE	d=0
ds	52AE	
es	52AE	
ss	52AE	
cs	52AE	
ip	0100	

ds:0000	CD 20 FF 9F 00 9A F0 FE	= Я ЬЕ
ds:0008	1D F0 E4 01 6C 19 AE 01	•ЕФ01400
ds:0010	6C 19 80 02 C7 13 35 05	1A0 :54
ds:0018	01 01 01 00 02 FF FF FF	000 0
ds:0020	FF FF FF FF FF FF FF FF	

ss:0000	20CD
ss:FFFE	0000
ss:FFFC	0000
ss:FFFA	0000
ss:FFF8	0000

Рисунок 7 – Загрузка COM модуля в основную память

Ответы на контрольные вопросы. 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 (ip = 0100h).

- 2) Что располагается с 0 адреса?

Адрес начала PSP.

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

48DDh. Они указывают на начало PSP.

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

Стек определяется автоматически, указатель стека устанавливается на конец сегмента. Если для программы размер сегмента в 64КБ является достаточным, то DOS устанавливает в регистре SP адрес конца сегмента – FFFEH. Адреса расположены в диапазоне 0000h-FFFEh.

Шаг 6. Загрузка «хорошего» EXE модуля в память.


```

cs:0152 1E push ds
cs:0153 2BC0 sub ax, ax
cs:0155 50 push ax
cs:0156 B8DE52 mov ax, 52DE
cs:0159 8ED8 mov ds, ax
cs:015B E8FEFE call 005C
cs:015E E87BFF call 00DC
cs:0161 32C0 xor al, al
cs:0163 B44C mov ah, 4C
cs:0165 CD21 int 21
cs:0167 CB retf
cs:0168 0000 add [bx+si], al
cs:016A 0000 add [bx+si], al
cs:016C 0000 add [bx+si], al
cs:016E 0000 add [bx+si], al

ds:0000 CD 20 FF 9F 00 9A F0 FE = Я бЕ
ds:0008 1D F0 E4 01 6C 19 AE 01 *Еф0100
ds:0010 6C 19 80 02 C7 13 35 05 10000000
ds:0018 01 01 01 00 02 FF FF FF 0000
ds:0020 FF FF FF FF FF FF FF

ax 0000 c=0
bx 0000 z=0
cx 0000 s=0
dx 0000 o=0
si 0000 p=0
di 0000 a=0
bp 0000 i=1
sp 0200 d=0
ds 52AE
es 52AE
ss 52BE
cs 52EA
ip 0152

ss:0202 0A0D
ss:0200 4350
ss:01FE 0000
ss:01FC 0000
ss:01FA 0000

```

Рисунок 8 – Загрузка «хорошего» EXE модуля в память

Ответы на контрольные вопросы. Загрузка «хорошего» EXE модуля в память.

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

В области памяти строится PSP, стандартная часть заголовка считывается в память, определяется длина тела загрузочного модуля, определяется начальный сегмент, загрузочный модуль считывается в начальный сегмент, таблица настройки считывается в рабочую память, определяются значения сегментных регистров. DS и ES устанавливаются на начало PSP, SS - на начало стека, CS - на начало сегмента кода.

2) На что указывают регистры DS и ES?

DS и ES указывают на начало PSP. После выполнения команд `mov ax, @data` и `mov ds, ax` регистре DS содержит адрес начала сегмента данных.

3) Как определяется стек?

В исходном коде модуля стек определяется при помощи директивы `STACK`, а при исполнении в регистры SS и SP записываются адрес начала сегмента стека и его вершины соответственно.

4) Как определяется точка входа?

При помощи команды `END`.

Вывод.

В ходе работы было проведено исследование различий в структурах исходных текстов модулей .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД SRCCOM.ASM

```
LAB1 SEGMENT
    ASSUME CS:LAB1, DS:LAB1, ES:NOTHING, SS:NOTHING
    ORG 100H ; PSP

OFF_MSDOS EQU 18
OFF_OEM EQU 15
OFF_SERIAL EQU 18

START: JMP BEGIN

;DATA SEGMENT
    T_PC          db 'PC', 0dh, 0ah, '$'
    T_PC_XT       db 'PC/XT', 0dh, 0ah, '$'
    T_AT          db 'AT', 0dh, 0ah, '$'
    T_PS2_30      db 'PS2 model 30', 0dh, 0ah, '$'
    T_PS2_5060    db 'PS2 model 50/60', 0dh, 0ah, '$'
    T_PS2_80      db 'PS2 model 80', 0dh, 0ah, '$'
    T_PCJR        db 'PCJR', 0dh, 0ah, '$'
    T_PC_CONVERTIBLE db 'PC convertible', 0dh, 0ah, '$'
    T_PC_UNKNOWN  db 'UNKNOWN: ', 0dh, 0ah, '$'

    IBM_PC        db 'IBM PC type is: ', '$'
    MS_DOS_VERSION db 'MSDOS version is: #.#', 0dh, 0ah, '$'
    OEM           db 'OEM number is: ##', 0dh, 0ah, '$'
    SERIAL        db 'Serial number is: #####', 0dh, 0ah, '$'
;DATA ENDS

;CODE SEGMENT
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
    push CX ; байт в AL переводится в два символа шестн. числа в AX
    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 ; 16 с/с 16 bit. В AX - число, 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 ; 10 с/с, 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_l
    or AL,30h
    mov [SI],AL
end_l: pop DX
    pop CX
    ret

```

BYTE_TO_DEC ENDP

WriteMsg PROC NEAR

```

    mov AH,9
    int 21h
    ret

```

WriteMsg ENDP

TYPE_PC PROC near

```

    ; Save stack
    push AX
    push BX
    push CX
    push ES
    ; Show 'IBM PC' message
    mov DX, offset IBM_PC
    call WriteMsg
    ; Get PC type
    xor AH,AH
    mov CX, 0F000h
    mov ES,CX
    mov AL, ES:[0FFFEh]
    ; Check PC type name
    cmp AX, 0FFh
    jz PCM
    cmp AX, 0FEh
    jz PCXTM
    cmp AX, 0FBh
    jz PCXTM
    cmp AX, 0FCh
    jz ATM
    cmp AX, 0FAh

```

```

        jz PS230M
    cmp AX, 0F6h
        jz PS250M
    cmp AX, 0F8h
        jz PS280M
    cmp AX, 0FDh
        jz PCjr_TYPEM
    cmp AX, 0F9h
        jz PC_CONVERTM
    mov DX,offset T_PC_UNKNOWN
    jmp ENDP

PCM:
    mov DX,offset T_PC
    jmp ENDP

PCXTM:
    mov DX,offset T_PC_XT
    jmp ENDP

ATM:
    mov DX,offset T_AT
    jmp ENDP

PS230M:
    mov DX,offset T_PS2_30
    jmp ENDP

PS250M:
    mov DX,offset T_PS2_5060
    jmp ENDP

PS280M:
    mov DX,offset T_PS2_80
    jmp ENDP

PCjr_TYPEM:
    mov DX,offset T_PCJR
    jmp ENDP

PC_CONVERTM:
    mov DX,offset T_PC_CONVERTIBLE
    jmp ENDP

ENDP:
    ; Write PC type
    call WriteMsg
    pop AX
    pop BX
    pop CX
    pop ES
    ret
TYPE_PC ENDP

MS_DOS_OEM_SERIAL PROC near
    ; Save stack
    push AX
    push BX
    push CX
    push DX
    ; Get system info
    xor AX, AX
    mov ah, 30h
    int 21h
    push BX ; BH = OEM
    push CX ; BL:CX = serial
    ; Get ms-dos version
    mov BX, offset MS_DOS_VERSION
    add BX, OFF_MSDOS
    ; Major number
    mov DH, AH
    xor AH, AH

```

```

    call BYTE_TO_HEX
    mov [BX], AH
    add BX, 2
    ; Minor number
    xor AX, AX
    mov AL, DH
    call BYTE_TO_HEX
    mov [BX], AX
    ; Print version
    mov DX, offset MS_DOS_VERSION
    call WriteMsg
    ; Get OEM
    pop CX
    pop BX
    xor DX, DX
    mov DX, BX
    mov BX, offset OEM
    add BX, OFF_OEM
    ; Convert OEM
    xor AX, AX
    mov AL, DH
    call BYTE_TO_HEX
    mov [BX], AX
    ; Print OEM
    mov BX, DX
    mov DX, offset OEM
    call WriteMsg
    ; Get serial number
    mov AL, BL
    mov BX, offset SERIAL
    add BX, OFF_SERIAL
    ; Convert first byte
    call BYTE_TO_HEX
    mov [BX], AX
    add BX, 2
    ; Convert second byte
    mov AL, CH
    call BYTE_TO_HEX
    mov [BX], AX
    add BX, 2
    ; Convert third byte
    mov AL, CL
    call BYTE_TO_HEX
    mov [BX], AX

    mov DX, offset SERIAL
    call WriteMsg

    pop AX
    pop BX
    pop CX
    pop DX
    ret
MS_DOS_OEM_SERIAL ENDP
BEGIN:
    push DS
    sub AX, AX
    push AX
    ;mov AX, DATA
    ;mov DS, AX
    ; Начало
    call TYPE_PC
    call MS_DOS_OEM_SERIAL
    ; Выход

```

```
xor AL,AL  
mov AH,4Ch  
int 21H  
ret
```

```
LAB1 ENDS  
END START
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД SRCEXE.ASM

```
OFF_MSDOS EQU 18
OFF_OEM EQU 15
OFF_SERIAL EQU 18
```

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

```
DATA SEGMENT
    T_PC          db 'PC', 0dh, 0ah, '$'
    T_PC_XT       db 'PC/XT', 0dh, 0ah, '$'
    T_AT          db 'AT', 0dh, 0ah, '$'
    T_PS2_30      db 'PS2 model 30', 0dh, 0ah, '$'
    T_PS2_5060    db 'PS2 model 50/60', 0dh, 0ah, '$'
    T_PS2_80      db 'PS2 model 80', 0dh, 0ah, '$'
    T_PCJR        db 'PCJR', 0dh, 0ah, '$'
    T_PC_CONVERTIBLE db 'PC convertible', 0dh, 0ah, '$'
    T_PC_UNKNOWN  db 'UNKNOWN: ', 0dh, 0ah, '$'

    IBM_PC        db 'IBM PC type is: ', '$'
    MS_DOS_VERSION db 'MSDOS version is: #.#', 0dh, 0ah, '$'
    OEM           db 'OEM number is: ##', 0dh, 0ah, '$'
    SERIAL        db 'Serial number is: #####', 0dh, 0ah, '$'
DATA ENDS
```

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

```
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
    push CX ; байт в AL переводится в два символа шестн. числа в AX
    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 ; 16 с/с 16 bit. В AX - число, 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 ; 10 с/с, 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

```

```

WriteMsg PROC NEAR
    mov AH,9
    int 21h
    ret
WriteMsg ENDP

```

```

TYPE_PC PROC near
; Save stack
    push AX
    push BX
    push CX
    push ES
; Show 'IBM PC' message
    mov DX, offset IBM_PC
    call WriteMsg
; Get PC type
    xor AH,AH
    mov CX, 0F000h
    mov ES,CX
    mov AL, ES:[0FFFEh]
; Check PC type name
    cmp AX, 0FFh
    jz PCM
    cmp AX, 0FEh
    jz PCXTM
    cmp AX, 0FBh
    jz PCXTM
    cmp AX, 0FCh
    jz ATM
    cmp AX, 0FAh
    jz PS230M
    cmp AX, 0F6h

```

```

        jz PS250M
    cmp AX, 0F8h
        jz PS280M
    cmp AX, 0FDh
        jz PCjr_TYPEM
    cmp AX, 0F9h
        jz PC_CONVERTM
    mov DX,offset T_PC_UNKNOWN
    jmp ENDP

PCM:
    mov DX,offset T_PC
    jmp ENDP

PCXTM:
    mov DX,offset T_PC_XT
    jmp ENDP

ATM:
    mov DX,offset T_AT
    jmp ENDP

PS230M:
    mov DX,offset T_PS2_30
    jmp ENDP

PS250M:
    mov DX,offset T_PS2_5060
    jmp ENDP

PS280M:
    mov DX,offset T_PS2_80
    jmp ENDP

PCjr_TYPEM:
    mov DX,offset T_PCJR
    jmp ENDP

PC_CONVERTM:
    mov DX,offset T_PC_CONVERTIBLE
    jmp ENDP

ENDP:
    ; Write PC type
    call WriteMsg
    pop AX
    pop BX
    pop CX
    pop ES
    ret
TYPE_PC ENDP

MS_DOS_OEM_SERIAL PROC near
    ; Save stack
    push AX
    push BX
    push CX
    push DX
    ; Get system info
    xor AX, AX
    mov ah, 30h
    int 21h
    push BX ; BH = OEM
    push CX ; BL:CX = serial
    ; Get ms-dos version
    mov BX, offset MS_DOS_VERSION
    add BX, OFF_MSDOS
    ; Major number
    mov DH, AH
    xor AH, AH
    call BYTE_TO_HEX
    mov [BX], AH

```

```

    add BX, 2
    ; Minor number
    xor AX, AX
    mov AL, DH
    call BYTE_TO_HEX
    mov [BX], AX
    ; Print version
    mov DX, offset MS_DOS_VERSION
    call WriteMsg
    ; Get OEM
    pop CX
    pop BX
    xor DX, DX
    mov DX, BX
    mov BX, offset OEM
    add BX, OFF_OEM
    ; Convert OEM
    xor AX, AX
    mov AL, DH
    call BYTE_TO_HEX
    mov [BX], AX
    ; Print OEM
    mov BX, DX
    mov DX, offset OEM
    call WriteMsg
    ; Get serial number
    mov AL, BL
    mov BX, offset SERIAL
    add BX, OFF_SERIAL
    ; Convert first byte
    call BYTE_TO_HEX
    mov [BX], AX
    add BX, 2
    ; Convert second byte
    mov AL, CH
    call BYTE_TO_HEX
    mov [BX], AX
    add BX, 2
    ; Convert third byte
    mov AL, CL
    call BYTE_TO_HEX
    mov [BX], AX

    mov DX, offset SERIAL
    call WriteMsg

    pop AX
    pop BX
    pop CX
    pop DX
    ret
MS_DOS_OEM_SERIAL ENDP

Main PROC FAR
    push DS
    sub AX, AX
    push AX
    mov AX, DATA
    mov DS, AX
    ; Начало
    call TYPE_PC
    call MS_DOS_OEM_SERIAL
    ; Выход
    xor AL, AL

```

```
mov AH,4Ch  
int 21H  
ret
```

```
Main      ENDP
```

```
CODE ENDS  
END Main
```