

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

Студентка гр. 9382

Круглова В.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование различий в структурах исходных текстов модулей .COM и .EXE, структур файлов загрузочных модулей и изучение 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.

Необходимо изготовить .COM модуль, который определяет тип PC и версию системы. Программа на ассемблере должна читать содержимое в байте по адресу 0F000:0FFFEh. Затем, сравнивая коды по таблице, определять тип PC и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код должен переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения. Далее определяется версия системы. 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 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, построить его, отладить и сравнить исходные тексты для .COM и .EXE модулей.

Ход работы.

Шаг 1. Запуск «хорошего» .COM модуля.



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

Рисунок 1 – «Хороший» .COM модуль

Шаг 2. Запуск «плохого» .EXE модуля.

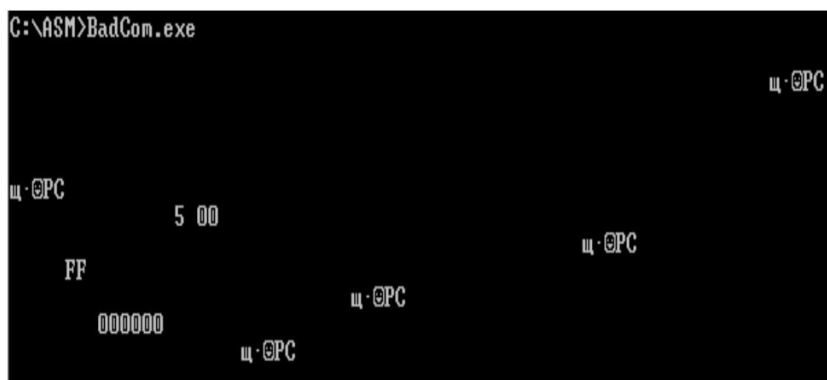


Рисунок 2 – «Плохой» .EXE модуль

Шаг 3. Запуск «хорошего» .EXE модуля.

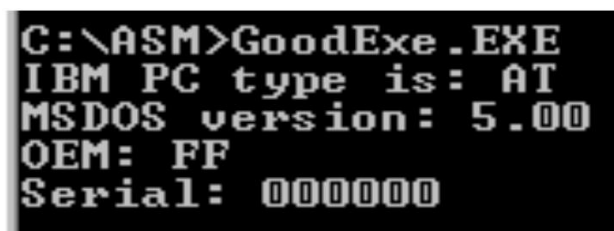


Рисунок 3 – «Хороший» .EXE модуль

Ответы на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

1. Сколько сегментов должна содержать COM-программа?

Только один сегмент.

2. EXE программа?

Один или более в зависимости от используемой модели памяти.

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

Директива ORG 100h для 256 байт (100h) блока данных PSP. Директива ASSUME, устанавливающая соответствие сегментных регистров одному сегменту, т. к. в COM программе он один.

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

Нет, так как в отличие от EXE-программы, где есть таблица настройки адресов, в которой указано, где лежат сегменты, в COM-программе смещения должны оставаться неизменными, т. е. команды вида mov [регистр], seg [сегмент] использовать нельзя.

Шаг 4. .COM модуль, «плохой» и «хороший» .EXE модули в шестнадцатеричном виде.

```

0000000000: E9 FA 01 50 43 0D 0A 24 50 43 2F 58 54 0D 0A 24 eu@PCJPC/XTJPC$
0000000010: 41 54 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 33 ATJPC$PS2 model 3
0000000020: 30 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 35 30 0JPC$PS2 model 50
0000000030: 2F 36 30 0D 0A 24 50 53 32 20 6D 6F 64 65 6C 20 /60JPC$PS2 model
0000000040: 38 30 0D 0A 24 50 43 4A 52 0D 0A 24 50 43 20 63 80JPC$PCJRJPC$PC c
0000000050: 6F 6E 76 65 72 74 69 62 6C 65 0D 0A 24 55 4E 4B onvertibleJPC$UNK
0000000060: 4E 4F 57 4E 3A 20 20 20 0D 0A 24 49 42 4D 20 50 NOWN: JPC$IBM P
0000000070: 43 20 74 79 70 65 20 69 73 3A 20 24 4D 53 44 4F C type is: $MSDO
0000000080: 53 20 76 65 72 73 69 6F 6E 3A 20 20 2E 20 0D 0A S version: . JPC
0000000090: 24 4F 45 4D 3A 20 20 20 0D 0A 24 53 65 72 69 61 $OEM: JPC$Seria
00000000A0: 6C 3A 20 20 20 20 20 20 20 0D 0A 24 24 0F 3C 09 l: JPC$*$<C
00000000B0: 76 02 04 07 04 30 C3 51 8A E0 E8 EF FF 86 C4 B1 v@+@QAQSaeytA+
00000000C0: 04 D2 E8 E8 E6 FF 59 C3 53 8A FC E8 E9 FF 88 25 *Oee?yYASSueey?%
00000000D0: 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88 05 5B C3 0?%OSCe?y?%O?%[A
00000000E0: 51 52 32 E4 33 D2 B9 0A 00 F7 F1 80 CA 30 88 14 QR2a30?@ ?n?E0?%
00000000F0: 4E 33 D2 3D 0A 00 73 F1 3C 00 74 04 0C 30 88 04 N30=@ sn< t+?@?%
0000000100: 5A 59 C3 50 53 51 06 BA 6B 01 E8 EB 00 32 E4 B4 ZYAPSQ+?k@ee 2a?
0000000110: 00 B9 00 F0 8E C1 26 A0 FE FF 3D FF 00 74 2E 3D ? ?ZA& ?y=y t.=
0000000120: FE 00 74 2F 3D FB 00 74 2A 3D FC 00 74 2B 3D FA ? t/=u t*=u t+=u
0000000130: 00 74 2C 3D F6 00 74 2D 3D F8 00 74 2E 3D FD 00 t.=o t-=o t.=y
0000000140: 74 2F 3D F9 00 74 30 BA 5D 01 EB 31 90 BA 03 01 t/=u t0?l@e1??%@
0000000150: EB 2B 90 BA 08 01 EB 25 90 BA 10 01 EB 1F 90 BA e+??%e%??%@e%??%
0000000160: 15 01 EB 19 90 BA 24 01 EB 13 90 BA 36 01 EB 0D $@e!??%e!!??%6@eF
1Help 2Text 3Quit 4Dump 5 6Edit 7Search 8ANSI 9 10Quit

```

Рисунок 4 - .COM модуль в шестнадцатеричном виде

```

0000000000: 4D 5A 0E 01 03 00 00 00 20 00 00 00 FF FF 00 00 MZPC$ y9
0000000010: 00 00 00 00 00 00 01 00 1E 00 00 00 01 00 00 00
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
0000000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000D0: 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
00000000F0: 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
0000000110: 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
0000000130: 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
0000000150: 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
1Help 2Text 3Quit 4Dump 5 6Edit 7Search 8ANSI 9 10Quit

```

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

```

0000000000: 4D 5A 17 00 04 00 01 00 20 00 00 00 FF FF 00 00 MZPC$ y9
0000000010: 00 02 0A F0 51 01 2B 00 1E 00 00 00 01 00 56 01 @?Q@+ y9 U@
0000000020: 2B 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
0000000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000D0: 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
00000000F0: 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
0000000110: 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
0000000130: 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
0000000150: 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
1Help 2Text 3Quit 4Dump 5 6Edit 7Search 8ANSI 9 10Quit

```

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

Ответы на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

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

COM файл содержит код и данные программы, которые располагаются в одном сегменте. Код располагается с адреса 0h, при загрузке происходит смещение на 100h.

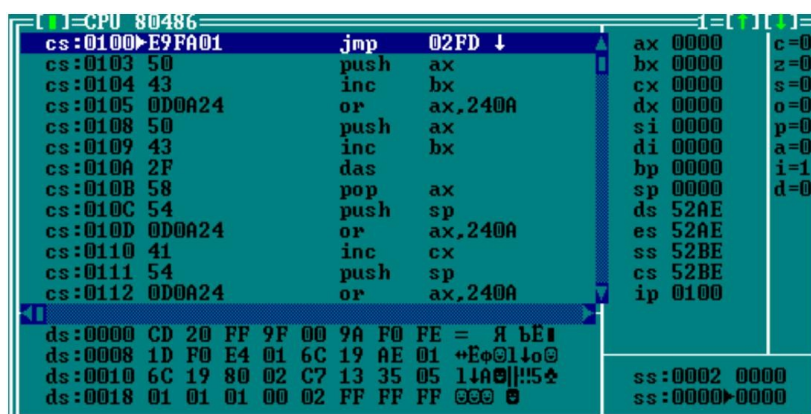
2. Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?

"Плохой" EXE-файл содержит только один сегмент. В самом начале идет заголовок EXE-файла, в котором содержится полезная информация, такая как сигнатура, длина заголовка в 16-байтных параграфах, значение SP и IP при входе и т.д. Заголовок занимает адреса 00-1BH(28 байт). За заголовком следует таблица настроек адресов, которая занимает ровно столько места, сколько указано в заголовке. Сам код в данном EXE-файле начинается с адреса 300H

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

Также, как и в "плохом" в "хорошем" в самом начале идет заголовок и таблица настроек адресов. "Хороший" EXE-файл содержит три сегмента: стека (в шестнадцатеричном виде представлен множеством символов S), данных и кода. Сам код в "хорошем" EXE-файле начинается с адреса 400H, так как в отличие от "плохого" в нем определен стек.

Шаг 5. Загрузка COM модуля в основную память.



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

CPU 80486		1=	
cs:0100	E9FA01 jmp 02FD ↓	ax	0000 c=0
cs:0103	50 push ax	bx	0000 z=0
cs:0104	43 inc bx	cx	0000 s=0
cs:0105	0D0A24 or ax,240A	dx	0000 o=0
cs:0108	50 push ax	si	0000 p=0
cs:0109	43 inc bx	di	0000 a=0
cs:010A	2F das	bp	0000 i=1
cs:010B	58 pop ax	sp	0000 d=0
cs:010C	54 push sp	ds	52AE
cs:010D	0D0A24 or ax,240A	es	52AE
cs:0110	41 inc cx	ss	52BE
cs:0111	54 push sp	cs	52BE
cs:0112	0D0A24 or ax,240A	ip	0100

ds:0000	CD 20 FF 9F 00 9A F0 FE = Я bE1
ds:0008	1D F0 E4 01 6C 19 AE 01 +Eф@14o@
ds:0010	6C 19 80 02 C7 13 35 05 14A@ !5♠
ds:0018	01 01 01 00 02 FF FF FF @@@ @

ss:0002	0000
ss:0000	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?

Определяется сегментный адрес свободного участка ОП, в который можно загрузить программу. В поля PSP заносятся значения, затем COM файл загружается со смещением 100h. Счетчик команд принимает значение 100h, и программа запускается.

2. Что располагается с 0 адреса?

PSP.

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

2AEh. Они указывают на PSP.

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

Стек определяется автоматически. DOS устанавливает в регистре SP адрес 000h. Адреса расположены в диапазоне 0000h-FFFEh.

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

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

CPU 80486		1=[]				
cs:0151	1E	push	ds	ax	0000	c=0
cs:0152	2BC0	sub	ax,ax	bx	0000	z=0
cs:0154	50	push	ax	cx	0000	s=0
cs:0155	B8DE52	mov	ax,52DE	dx	0000	o=0
cs:0158	8ED8	mov	ds,ax	si	0000	p=0
cs:015A	E8FAFE	call	0057	di	0000	a=0
cs:015D	E879FF	call	00D9	bp	0000	i=1
cs:0160	32C0	xor	al,al	sp	0200	d=0
cs:0162	B44C	mov	ah,4C	ds	52AE	
cs:0164	CD21	int	21	es	52AE	
cs:0166	CB	retf		ss	52BE	
cs:0167	DE528E	ficom	word ptr[bp+5]	cs	52E9	
cs:016A	D8E8	fsubr	st,st(0)	ip	0151	
ds:0000 CD 20 FF 9F 00 9A F0 FE = Я ЬЕ				ss:0202 0A0D		
ds:0008 1D F0 E4 01 6C 19 AE 01 *ЕФ0140				ss:0200 4350		
ds:0010 6C 19 80 02 C7 13 35 05 1A0 :5						
ds:0018 01 01 01 00 02 FF FF FF 000						

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?

DS и ES устанавливаются на начало сегмента PSP, SS—на начало сегмента стека, CS—на начало сегмента кода. В IP записывается адрес первой команды

2) На что указывают регистры DS и ES?

DS и ES указывают на начало префикса программного сегмента (PSP)

3) Как определяется стек?

Стек определяется в коде следующим образом:

<имя сегмента> SEGMENT STACK

DW <количество выделяемой памяти> DUP(<значение по умолчанию>)

<имя сегмента> ENDS

ASSUME SS:AStack

4) Как определяется точка входа?

Точка входа — начальное значение IP—вычисляется с помощью директивы
END <метка>.

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 BADCOM.ASM

SPECS SEGMENT

ASSUME CS:SPECS, DS:SPECS, ES:NOTHING, SS:NOTHING
ORG 100H

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: . ', 0dh, 0ah, '\$'
OEM db 'OEM: ', 0dh, 0ah, '\$'
SERIAL db 'Serial: ', 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_1
        or AL,30h
        mov [SI],AL
end_1:  pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

```

```

GET_PC PROC near
        push AX
        push BX
        push CX
        push ES

        mov DX, offset IBM_PC
        call PRINT
        xor AH,AH
        mov CX,0F000h
        mov ES,CX
        mov AL,ES:[0FFFEh] ; TYPE
        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
ENDPC:
        call PRINT ; Output
        pop AX
        pop BX
        pop CX
        pop ES
        ret
GET_PC ENDP

```

```

GET_SPECS PROC near
        push AX
        push BX
        push CX
        push DX

        xor AX,AX
        mov ah,30h
        int 21h ; Get specs
        push BX ; BH = OEM
        push CX ; BL:CX = serial
        ; MS DOS
        mov BX,offset MS_DOS_VERSION
        mov DH,AH
        xor AH,AH
        call BYTE_TO_HEX ; Major
        mov [BX + 15],AH
        add BX,2
        xor AX,AX
        mov AL,DH
        call BYTE_TO_HEX ; Minor
        mov [BX + 15],AX
        mov DX, offset MS_DOS_VERSION

```



```
call PRINT
; OEM
pop CX
pop BX
xor DX,DX
mov DX,BX
mov BX,offset OEM

xor AX,AX
mov AL,DH
call BYTE_TO_HEX
mov [BX + 5],AX

mov BX,DX
mov DX,offset OEM
call PRINT
; Serial
mov AL,BL
mov BX,offset SERIAL
; 1
call BYTE_TO_HEX
mov [BX + 8],AX
add BX,2
; 2
mov AL,CH
call BYTE_TO_HEX
mov [BX + 8],AX
add BX,2
; 3
mov AL,CL
call BYTE_TO_HEX
mov [BX + 8],AX
mov DX, offset SERIAL
call PRINT

pop AX
```

```
    pop BX
    pop CX
    pop DX
    ret
GET_SPECS ENDP
```

```
PRINT PROC NEAR
    mov AH,9
    int 21h
    ret
PRINT ENDP
```

```
BEGIN:
    push DS
    sub AX,AX
    push AX
    call GET_PC
    call GET_SPECS
    xor AL,AL
    mov AH,4Ch
    int 21H
    ret
```

```
SPECS ENDS
    END START
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД EXE.ASM

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: . ', 0dh, 0ah, '\$'

OEM db 'OEM: ', 0dh, 0ah, '\$'

SERIAL db 'Serial: ', 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

```

```

GET_PC PROC near
        push AX
        push BX
        push CX
        push ES

        mov DX, offset IBM_PC
        call PRINT
        xor AH,AH
        mov AH,0
        mov CX,0F000h
        mov ES,CX
        mov AL,ES:[0FFFEh] ; TYPE
        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
ENDPC:
        call PRINT ; Output
        pop AX
        pop BX
        pop CX
        pop ES
        ret
GET_PC ENDP

```

```

GET_SPECS PROC near
        push AX
        push BX
        push CX
        push DX

        xor AX,AX
        mov ah,30h
        int 21h ; Get specs
        push BX ; BH = OEM
        push CX ; BL:CX = serial
        ; MS DOS
        mov BX,offset MS_DOS_VERSION
        mov DH,AH
        xor AH,AH
        call BYTE_TO_HEX ; Major
        mov [BX + 15],AH
        add BX,2
        xor AX,AX
        mov AL,DH
        call BYTE_TO_HEX ; Minor
        mov [BX + 15],AX

```

```
mov DX, offset MS_DOS_VERSION
call PRINT
; OEM
pop CX
pop BX
xor DX,DX
mov DX,BX
mov BX,offset OEM
```

```
xor AX,AX
mov AL,DH
call BYTE_TO_HEX
mov [BX + 5],AX
```

```
mov BX,DX
mov DX,offset OEM
call PRINT
; Serial
mov AL,BL
mov BX,offset SERIAL
; 1
call BYTE_TO_HEX
mov [BX + 8],AX
add BX,2
; 2
mov AL,CH
call BYTE_TO_HEX
mov [BX + 8],AX
add BX,2
; 3
mov AL,CL
call BYTE_TO_HEX
mov [BX + 8],AX
mov DX, offset SERIAL
call PRINT
```

```
    pop AX
    pop BX
    pop CX
    pop DX
    ret
GET_SPECS ENDP
```

```
PRINT PROC NEAR
    mov AH,9
    int 21h
    ret
PRINT ENDP
```

```
Main PROC FAR
    push DS
    sub AX,AX
    push AX
    mov AX,DATA
    mov DS,AX
    call GET_PC
    call GET_SPECS
    xor AL,AL
    mov AH,4Ch
    int 21H
    ret
```

```
Main ENDP
```

```
CODE ENDS
    END Main
```