

**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. 9383

Моисейченко К. А.

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.

## **Задание.**

Шаг 1. Напишите 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 и серийным номером пользователя. Полученные строки выводятся на экран. Отладьте полученный исходный модуль. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

Шаг 2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» .EXE.

Шаг 3. Сравните исходные тексты для .COM и .EXE модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» .EXE и сравните его с предыдущими файлами.

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

Шаг 5. Откройте отладчик TD.EXE и загрузите .COM. Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля .COM в основную память.

Шаг 6. Откройте отладчик TD.EXE и загрузите «хороший» .EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

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

### **Контрольные вопросы по лабораторной работе.**

Отличия исходных текстов COM и EXE программ.

- 1) Сколько сегментов должна содержать COM-программа?
- 2) EXE-программа?
- 3) Какие директивы должны обязательно быть в тексте COM-программы?
- 4) Все ли форматы команд можно использовать в COM-программе?

Отличия форматов файлов COM и EXE модулей.

- 1) Какова структура файла COM? С какого адреса располагается код?
- 2) Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?
- 3) Какова структура файла «хорошего» EXE? Чем он отличается от файла «плохого» EXE?

Загрузка COM модуля в основную память.

- 1) Какой формат загрузки модуля COM? С какого адреса располагается код?
- 2) Что располагается с адреса 0?

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

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

Загрузка «хорошего» EXE модуля в основную память.

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

2) На что указывают регистры DS и ES?

3) Как определяется стек?

4) Как определяется точка входа?

### **Выполнение работы.**

Шаг 1.

Была разработана программа для модуля .COM (исходный код см. в приложении А). В результате был построен «плохой» .EXE, полученный из исходного текста для .COM модуля, и с помощью EXE2BIN.EXE был построен модуль .COM. Результаты работы модулей см. на рисунках 1 и 2.

```
C:\>lab1_com.com
Type of PC : AT
DOS version: 5.0
OEM number: 255
User serial number: 000000h
```

Рисунок 1 - Результат работы модуля .COM

```
C:\>lab1_com.exe

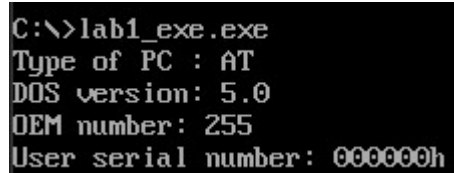
5 0
255
0+8Type 0000000 PC

0+8Type of PC : PC
0+8Type of PC : PC
0+8Type of PC : PC
```

Рисунок 2 - Результат работы «плохого» модуля .EXE

## Шаг 2.

Была разработана программа для «хорошего» .EXE модуля (исходный код см. в приложении А). Был построен «хороший» .EXE модуль. Результат работы модуля см. на рисунке 3.



```
C:\>lab1_exe.exe
Type of PC : AT
DOS version: 5.0
OEM number: 255
User serial number: 000000h
```

Рисунок 3 - Результат работы «хорошего» модуля .EXE

## Шаг 3.

Отличия исходных текстов COM и EXE программ.

Ответы на вопросы:

1) Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать один сегмент, потому что данные и код находятся в одном сегменте, а стек устанавливается на последнюю ячейку сегмента автоматически.

2) EXE-программа?

EXE-программа должна содержать один или более сегментов. При этом сегменты стека, кода и данных отделены друг от друга.

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

В COM-программе должна быть директива ORG 100h, так как первые 100h занимает PSP. Также обязательна директива ASSUME, которая указывает с каким сегментом или группой сегментов связаны регистры. Обязательна директива END, которая завершает работу программы.

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

В COM-программе не существует таблицы настроек, которая есть в EXE-программе (Relocation Table). Поэтому команды вида mov [register], seg [segment] нельзя использовать, т.к. в момент ассемблирования и редактирования

связей .COM не может предоставить загрузчику перечня всех сегментных ссылок.

Шаг 4.

Отличия форматов файлов COM и EXE модулей.

На рисунках представлены модули в шестнадцатеричном виде.

C:\OS\LAB1\COM.COM																
0000000000:	E9	2B	02	54	79	70	65	20	6F	66	20	50	43	20	3A	20
0000000010:	50	43	0D	0A	24	54	79	70	65	20	6F	66	20	50	43	20
0000000020:	3A	20	50	43	2F	58	54	0D	0A	24	54	79	70	65	20	6F
0000000030:	66	20	50	43	20	3A	20	41	54	0D	0A	24	54	79	70	65
0000000040:	20	6F	66	20	50	43	20	3A	20	50	53	32	20	6D	6F	64
0000000050:	65	6C	20	33	30	0D	0A	24	54	79	70	65	20	6F	66	20
0000000060:	50	43	20	3A	20	50	53	32	20	6D	6F	64	65	6C	20	35
0000000070:	30	20	6F	72	20	36	30	0D	0A	24	54	79	70	65	20	6F
0000000080:	66	20	50	43	20	3A	20	50	53	32	20	6D	6F	64	65	6C
0000000090:	20	38	30	0D	0A	24	54	79	70	65	20	6F	66	20	50	43
00000000A0:	20	3A	20	50	43	6A	72	0D	0A	24	54	79	70	65	20	6F
00000000B0:	66	20	50	43	3A	20	50	43	20	43	6F	6E	76	65	72	74
00000000C0:	69	62	6C	65	0D	0A	24	45	72	72	6F	72	20	0D	0A	24
00000000D0:	44	4F	53	20	76	65	72	73	69	6F	6E	3A	20	20	2E	20
00000000E0:	20	0D	0A	24	4F	45	4D	20	6E	75	6D	62	65	72	3A	20
00000000F0:	20	20	0D	0A	24	55	73	65	72	20	73	65	72	69	61	6C
0000000100:	20	6E	75	6D	62	65	72	3A	20	20	20	20	20	20	20	68
0000000110:	0D	0A	24	24	0F	3C	09	76	02	04	07	04	30	C3	51	8A
0000000120:	E0	E8	EF	FF	86	C4	B1	04	D2	E8	E8	E6	FF	59	C3	53
0000000130:	8A	FC	E8	E9	FF	88	25	4F	88	05	4F	8A	C7	E8	DE	FF
0000000140:	88	25	4F	88	05	5B	C3	51	52	32	E4	33	D2	B9	0A	00
0000000150:	F7	F1	80	CA	30	88	14	4E	33	D2	3D	0A	00	73	F1	3C
0000000160:	00	74	04	0C	30	88	04	5A	59	C3	B8	00	F0	8E	C0	BF
0000000170:	FE	FF	26	8A	25	80	FC	FF	75	06	BA	03	01	EB	5F	90
0000000180:	80	FC	FE	75	06	BA	15	01	EB	54	90	80	FC	FB	75	06
0000000190:	BA	15	01	EB	49	90	80	FC	FC	75	06	BA	2A	01	EB	3E
00000001A0:	90	80	FC	FA	75	06	BA	3C	01	EB	33	90	80	FC	FC	75
00000001B0:	06	BA	58	01	EB	28	90	80	FC	F8	75	06	BA	96	01	EB
00000001C0:	1D	90	80	FC	FD	75	06	BA	AA	01	EB	12	90	80	FC	FD
00000001D0:	75	06	BA	AA	01	EB	07	90	BA	C7	01	EB	01	90	B4	09
00000001E0:	CD	21	C3	B4	30	CD	21	BE	D0	01	83	C6	0D	50	E8	56
00000001F0:	FF	58	83	C6	03	8A	C4	E8	4D	FF	BA	D0	01	B4	09	CD
0000000200:	21	BE	E4	01	83	C6	0E	8A	C7	E8	3B	FF	BA	E4	01	B4
0000000210:	09	CD	21	BF	F5	01	83	C7	19	8B	C1	E8	11	FF	8A	C3
0000000220:	E8	FB	FE	89	45	FE	BA	F5	01	B4	09	CD	21	C3	E8	39
0000000230:	FF	E8	AF	FF	32	C0	B4	4C	CD	21						

Рисунок 4 - Модуль .COM











Ответы на вопросы:

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

Файл состоит из одного сегмента, содержит данные и машинные команды. COM-файл имеет ограничение в размере - 64 КБ. Код начинается с адреса 0h, при загрузке модуля устанавливается смещение 256 байт.

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

Файл состоит из одного сегмента. Сегмент с кодом и данными начинается с адреса 300h. С адреса 0h расположена управляющая информация загрузчика, которая содержит заголовок и таблицу настроек.

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

«Хороший» .EXE модуль, в отличие от «плохого», содержит 3 отдельных сегмента - сегмент стека, сегмент данных и сегмент кода. Файл не имеет ограничений в размере. С адреса 0h в «хорошем» модуле располагается таблица настроек. В «хорошем» .EXE модуле выделяется память под стек между PSP и кодом.

Шаг 5.

Загрузка COM модуля в основную память.

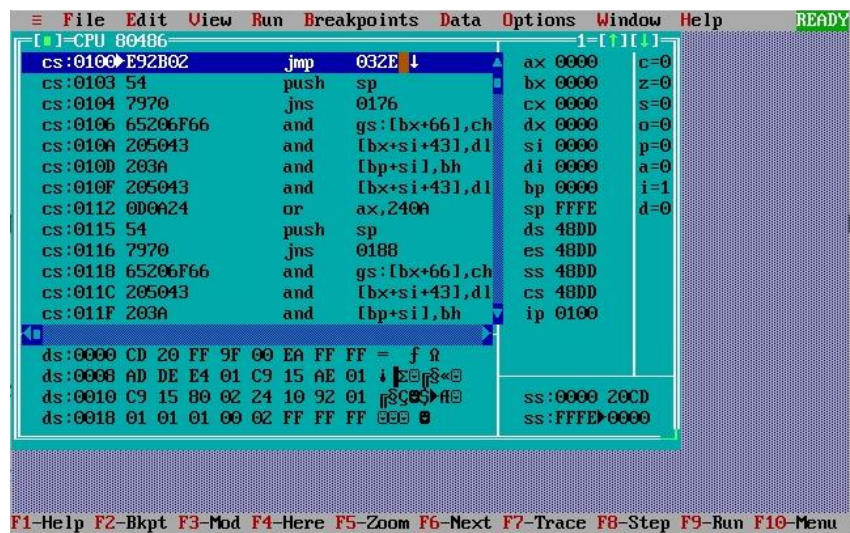


Рисунок 7 - Загрузка .COM модуля

Ответы на вопросы:

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

Сначала определяется сегментный адрес участка оперативной памяти для COM модуля. Затем туда помещается PSP и по смещению 100h считанный модуль.

2) Что располагается с адреса 0?

С адреса 0 располагается PSP размером в 100h байт.

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

CS, ES, DS, SS указывают на начало блока PSP, SP – на конец сегмента - FFFE.

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

Стек определяется автоматически при загрузке. Регистр SS указывает на начало блока PSP, а SSP на конец стека. Стек расположен между адресами SS:0000h и SS:FFFFh и заполняется с конца модуля в сторону уменьшения адресов.

Шаг 6.

Загрузка «хорошего» EXE модуля в основную память.

The screenshot shows a debugger window titled "CPU 80486". The main window is divided into three panes. The top pane shows assembly instructions with their addresses and hex values. The middle pane shows the current instruction being executed. The bottom pane shows the state of the registers and memory.

Address	Instruction	Hex Value
cs:011B	2BC0	sub ax,ax
cs:011D	50	push ax
cs:011E	B80D49	mov ax,490D
cs:0121	8ED8	mov ds,ax
cs:0123	E831FF	call 0057
cs:0126	E8A7FF	call 00D0
cs:0129	32C0	xor al,al
cs:012B	B44C	mov ah,4C
cs:012D	CD21	int 21
cs:012F	0000	add [bx+si],al
cs:0131	0000	add [bx+si],al
cs:0133	0000	add [bx+si],al
cs:0135	0000	add [bx+si],al

Registers:

Register	Value
ax	0000
bx	0000
cx	0000
dx	0000
si	0000
di	0000
bp	0000
sp	0200
ds	48DD
es	48DD
ss	48ED
cs	491E
ip	011B

Memory:

Address	Value
ds:0000	CD 20 FF 9F 00 EA FF FF = f 0
ds:0008	AD DE E4 01 C9 15 AE 01 i 20 8<0
ds:0010	C9 15 80 02 24 10 92 01 f 8<0
ds:0018	01 01 01 00 02 FF FF FF 00 0

Stack:

Address	Value
ss:0202	6570
ss:0200	7954

Footer: F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

Рисунок 8 - Загрузка .EXE модуля



Ответы на вопросы:

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

Загрузка начинается с адреса 0100h относительно PSP. ES и DS устанавливаются в начало PSP, SS - на начало сегмента стека, а CS – на начало сегмента команд. В IP загружается смещение точки входа в программу.

2) На что указывают регистры DS и ES?

Они указывают на начало сегмента PSP.

3) Как определяется стек?

Стек определяется при помощи директивы .STACK. SS указывает на начало сегмента, SP на конец.

4) Как определяется точка входа?

Она определяется параметром после директивы END.

### **Выводы.**

Были исследованы 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

Название файла: lab1\_com.asm

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

PC db 'Type of PC : PC' , 0dh, 0ah, '$'
PC_XT db 'Type of PC : PC/XT' , 0dh, 0ah, '$'
AT db 'Type of PC : AT', 0dh, 0ah, '$'
PS2_M30 db 'Type of PC : PS2 model 30', 0dh, 0ah, '$'
PS2_M50_M60 db 'Type of PC : PS2 model 50 or 60', 0dh, 0ah, '$'
PS2_M80 db 'Type of PC : PS2 model 80', 0dh, 0ah, '$'
PCjr db 'Type of PC : PCjr', 0dh, 0ah, '$'
PC_CONV db 'Type of PC: PC Convertible' , 0dh, 0ah, '$'
ERROR db 'Error ', 0dh, 0ah, '$'
DOS db 'DOS version: . ', 0dh, 0ah, '$'
OEM db 'OEM number: ', 0dh, 0ah, '$'
USER db 'User serial number:      h', 0dh, 0ah, '$'

; 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 шестн. 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, 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 10с/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
;-----
; КОД

PC_TYPE PROC near
mov ax, 0f000h
mov es,ax
mov di, 0ffffh
mov ah, es:[di]

cmp ah, 0FFh
jne PRINT_PC_XT_1
mov dx, offset PC
jmp PRINT_TYPE

PRINT_PC_XT_1:
cmp ah, 0FEh
jne PRINT_PC_XT_2
mov dx, offset PC_XT
jmp PRINT_TYPE

PRINT_PC_XT_2:

```

```

cmp ah,0FBh
jne PRINT_AT
mov dx, offset PC_XT
jmp PRINT_TYPE

PRINT_AT:
cmp ah,0FCh
jne PRINT_PS2_M30
mov dx,offset AT
jmp PRINT_TYPE

PRINT_PS2_M30:
cmp ah,0FAh
jne PRINT_PS2_M50_M60
mov dx, offset PS2_M30
jmp PRINT_TYPE

PRINT_PS2_M50_M60:
cmp ah,0FCh
jne PRINT_PS2_M80
mov dx,offset PS2_M50_M60
jmp PRINT_TYPE

PRINT_PS2_M80:
cmp ah,0F8h
jne PRINT_PCjr
mov dx,offset PCjr
jmp PRINT_TYPE

PRINT_PCjr:
cmp ah,0FDh
jne PRINT_PC_CONV
mov dx, offset PC_CONV
jmp PRINT_TYPE

PRINT_PC_CONV:
cmp ah,0FDh
jne PRINT_ERROR
mov dx, offset PC_CONV
jmp PRINT_TYPE

PRINT_ERROR:
mov dx,offset ERROR
jmp PRINT_TYPE

PRINT_TYPE:
mov ah,9h
int 21h

ret
PC_TYPE ENDP

OS_TYPE PROC near

mov ah,30h

```



```

int 21h

mov si,offset DOS
add si,13
push ax
call BYTE_TO_DEC

pop ax
add si,3
mov al,ah
call BYTE_TO_DEC

mov dx,offset DOS
mov ah,9h
int 21h

mov si,offset OEM
add si,14
mov al,bh
call BYTE_TO_DEC

mov dx, offset OEM
mov ah,9h
int 21h

mov di, offset USER
add di,25
mov ax,cx
call WRD_TO_HEX
mov al,bl
call BYTE_TO_HEX
mov [di-2],ax

mov dx,offset USER
mov ah,9h
int 21h

ret
OS_TYPE ENDP

BEGIN:
call PC_TYPE
call OS_TYPE

; Выход в DOS
xor AL,AL
mov AH,4Ch
int 21h
TESTPC ENDS
END START ;конец модуля, START - точка входа

```

**Название файла: lab1\_exe.asm**

AStack SEGMENT STACK

DW 256 DUP(?)

AStack ENDS

DATA SEGMENT

PC db 'Type of PC : PC' , 0dh, 0ah, '\$'

PC\_XT db 'Type of PC : PC/XT' , 0dh, 0ah, '\$'

AT db 'Type of PC : AT', 0dh, 0ah, '\$'

PS2\_M30 db 'Type of PC : PS2 model 30', 0dh, 0ah, '\$'

PS2\_M50\_M60 db 'Type of PC : PS2 model 50 or 60', 0dh, 0ah, '\$'

PS2\_M80 db 'Type of PC : PS2 model 80', 0dh, 0ah, '\$'

PCjr db 'Type of PC : PCjr', 0dh, 0ah, '\$'

PC\_CONV db 'Type of PC: PC Convertible' , 0dh, 0ah, '\$'

ERROR db 'Error ', 0dh, 0ah, '\$'

DOS db 'DOS version: . ', 0dh, 0ah, '\$'

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

USER db 'User serial number: h', 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

; байт в AL переводится в два символа шестн. 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, 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 10с/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
```

```
PC_TYPE PROC near
mov ax, 0f000h
mov es,ax
mov di, 0fffeh
mov ah, es:[di]
```

```
cmp ah, 0FFh
jne PRINT_PC_XT_1
mov dx, offset PC
jmp PRINT_TYPE
```

```
PRINT_PC_XT_1:
cmp ah, 0FEh
jne PRINT_PC_XT_2
mov dx,offset PC_XT
jmp PRINT_TYPE
```

```
PRINT_PC_XT_2:
cmp ah,0FBh
jne PRINT_AT
mov dx, offset PC_XT
jmp PRINT_TYPE
```



PRINT\_AT:

cmp ah,0FCh

jne PRINT\_PS2\_M30

mov dx,offset AT

jmp PRINT\_TYPE

PRINT\_PS2\_M30:

cmp ah,0FAh

jne PRINT\_PS2\_M50\_M60

mov dx, offset PS2\_M30

jmp PRINT\_TYPE

PRINT\_PS2\_M50\_M60:

cmp ah,0FCh

jne PRINT\_PS2\_M80

mov dx,offset PS2\_M50\_M60

jmp PRINT\_TYPE

PRINT\_PS2\_M80:

cmp ah,0F8h

jne PRINT\_PCjr

mov dx,offset PCjr

jmp PRINT\_TYPE

PRINT\_PCjr:

cmp ah,0FDh

jne PRINT\_PC\_CONV

mov dx, offset PC\_CONV

jmp PRINT\_TYPE

PRINT\_PC\_CONV:

cmp ah,0FDh

jne PRINT\_ERROR

mov dx, offset PC\_CONV

jmp PRINT\_TYPE

```
PRINT_ERROR:
mov dx,offset ERROR
jmp PRINT_TYPE
```

```
PRINT_TYPE:
mov ah,9h
int 21h
```

```
ret
PC_TYPE ENDP
```

```
OS_TYPE PROC near
```

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

```
mov si,offset DOS
add si,13
push ax
call BYTE_TO_DEC
```

```
pop ax
add si,3
mov al,ah
call BYTE_TO_DEC
```

```
mov dx,offset DOS
mov ah,9h
int 21h
```

```
mov si,offset OEM
add si,14
mov al,bh
call BYTE_TO_DEC
```

```
mov dx, offset OEM
mov ah,9h
int 21h
```

```
mov di, offset USER
add di,25
mov ax,cx
call WRD_TO_HEX
mov al,bl
call BYTE_TO_HEX
mov [di-2],ax
```

```
mov dx,offset USER
mov ah,9h
int 21h
```

```
ret
OS_TYPE ENDP
```

```
MAIN PROC FAR
sub ax,ax
push ax
mov ax,DATA
mov ds,ax
```

```
call PC_TYPE
call OS_TYPE
```

```
xor AL,AL
mov AH,4Ch
int 21H
MAIN ENDP
```

```
CODE ENDS
```

END MAIN