**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. 8383 |  | Мололкин К.А. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2020

**Цель работы.**

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

**Ход работы.**

Был написан текст исходного **.COM** модуля, который определяет тип PC и версию системы. Для этого программа читает содержимое предпоследнего байта ROM BIOS и, в соответствии с табл. 1, определяет тип PC.

Таблица 1 - Соответствие кода и тип PC

|  |  |
| --- | --- |
| Тип IBM PC | Код |
| 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 системы требуется воспользоваться функцией 30H прерывания 21H, затем из регистра AL считать номер основной версии, из AH номер модификации, из ВН серийный номер ОЕМ, а из регистров BL:CX 24-битовый серийные номер пользователя. Код программы представлен в приложении А.

Из полученного кода был получен «хороший» .COM модуль, а также «плохой» .EXE. Следующим шагом был написан текст исходного .EXE модуля, который выполняет те же функции, что и .COM полученный ранее.

**Ответы на контрольные вопросы**

**Отличия исходных текстов COM и EXE программ**

1) Сколько сегментов должна содержать COM-программа?

СОМ – программа должна содержать один сегмент с кодом и данными.

2) Сколько сегментов должна содержать EXE-программа?

EXE-программа может содержать любое количество сегментов

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

В тексте COM-программы обязательно должны быть директивыorg 100h для резервирования памяти PSP и assume для инициализации регистров.

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

В COM-программах нельзя использовать команды обращения к сегментам, потому что в COM-программах только один сегмент.

**Отличие форматов файлов COM и EXE модулей**

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

COM файл из одного сегмента, в котором располагается код и данные, код располагается с адреса 100h.

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

С нулевого адреса «плохого» EXE файла располагается заголовок и таблица настроек, затем с адреса 300h начинается единственный сегмент кода. На рис. 1 и 2 представлен шестнадцатеричный вид файла.

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

«Хороший» EXE в начале файла содержит задержит заголовок и стек, код начинается с 240h, в отличие от «плохого», у которого код начинается с 300h, а также «плохой» EXE содержит только один сегмент. На рис. 3 и 4 представлен шестнадцатеричный вид «хорошего» EXE.

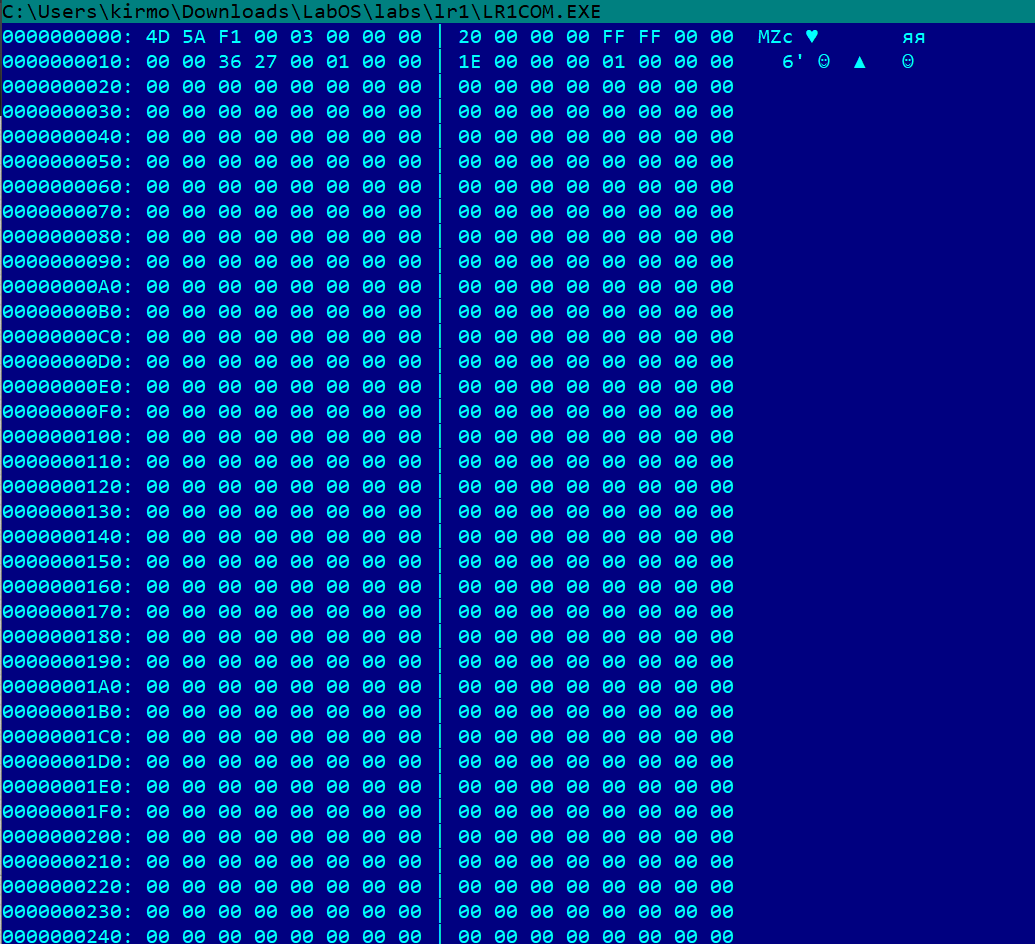


Рисунок 1 – начало «плохого» COM файла



Рисунок 2 – конец «плохого» COM файла

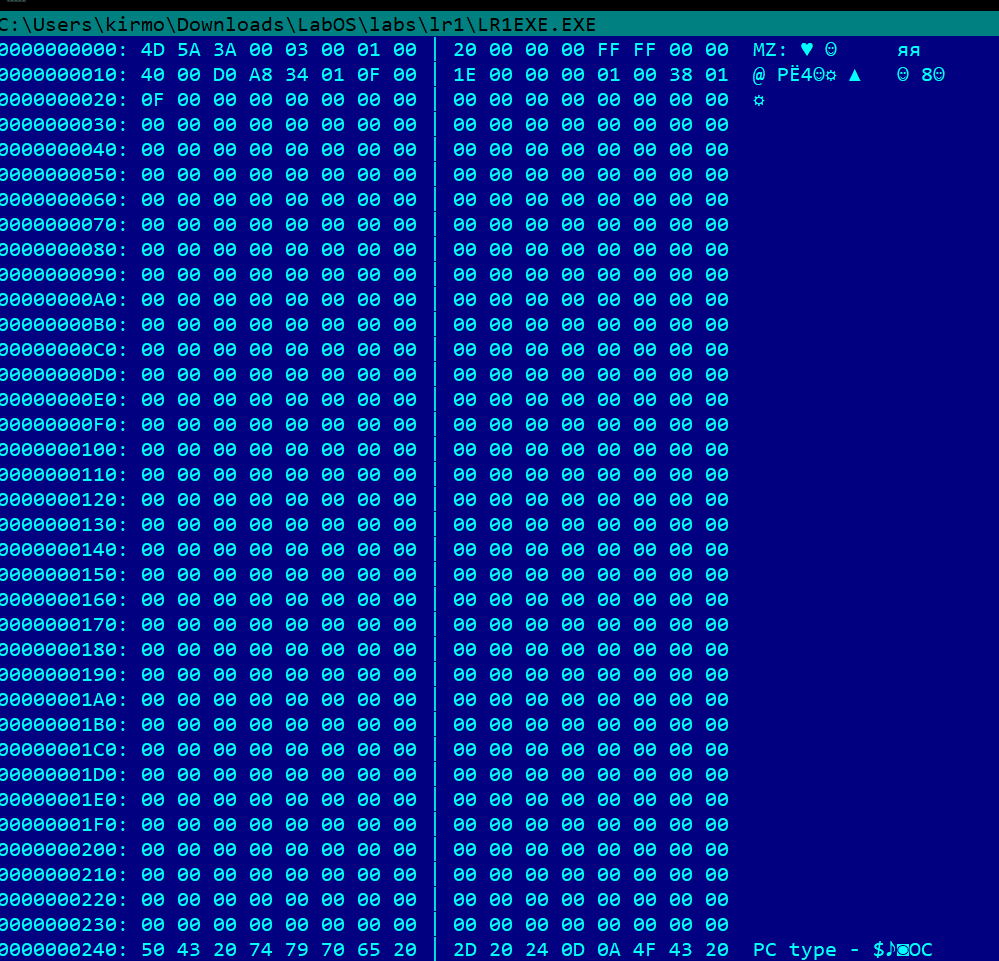


Рисунок 3 – начало «хорошего» EXE файла

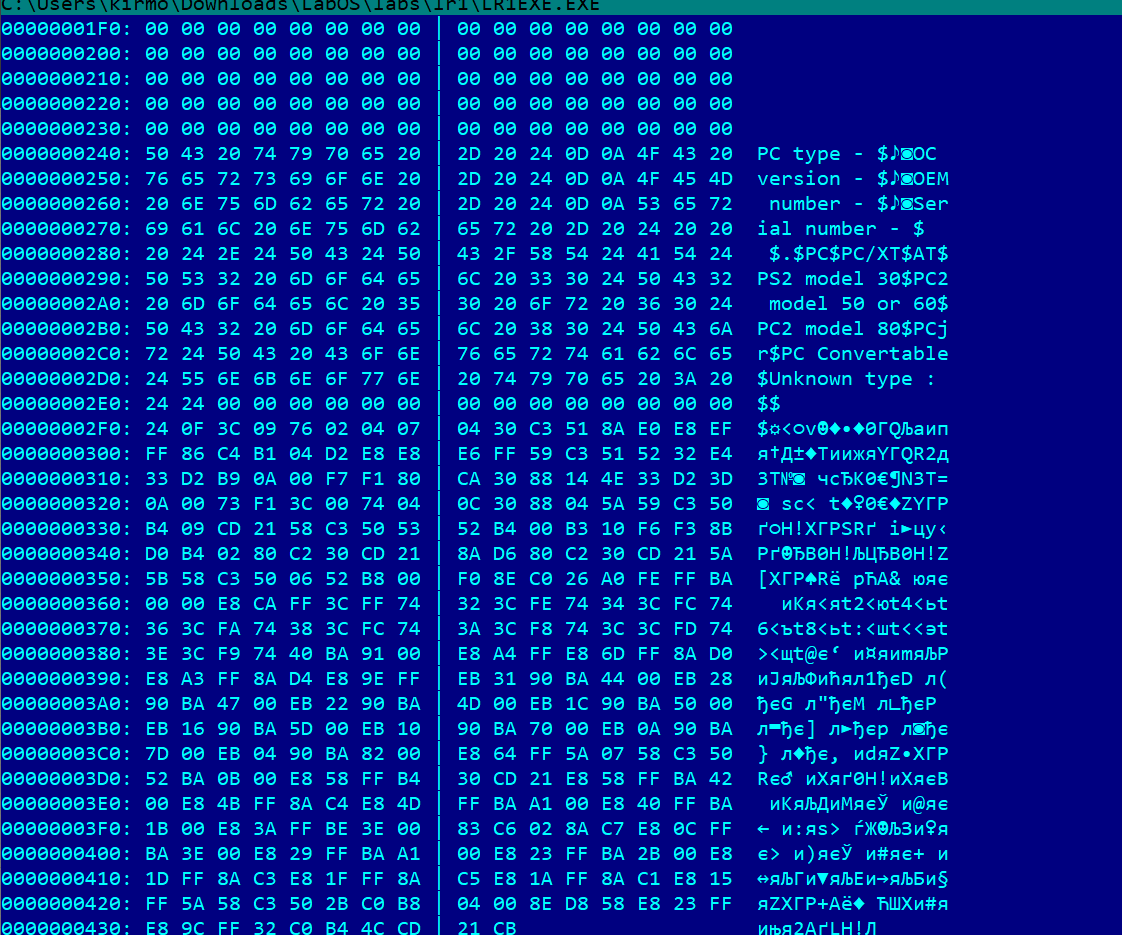


Рисунок 4 –конец «хорошего» EXE файла

**Загрузка COM модуля в основную память**

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

Код располагается с адреса 100h, до него располагается PSP. Данные, код и PSP расположены в одном сегменте.

2) Что располагается с адреса 0?

С адреса 0h располагается PSP.

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

Все сегментные регистры имеют значение 48DD и указывают на начало PSP. На рис. 5 представлен COM модуль, загруженный в отладчик.

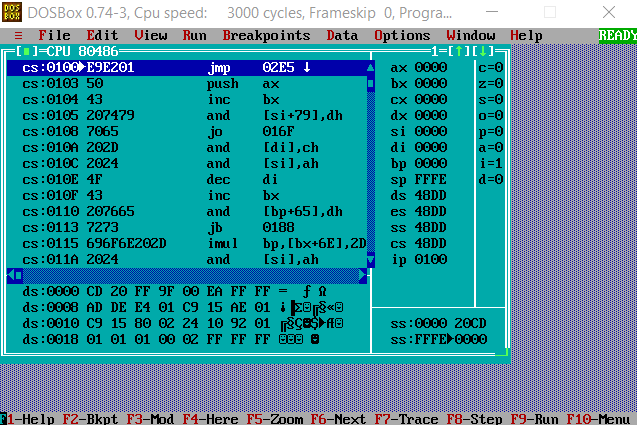


Рисунок 5 – COM модуль открытый в отладчике

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

Регистр сегмента стека при запуске программы равен FFFEh, стек занимает всю пространство не занятое PSP и кодом программы.

**Загрузка "хорошего" EXE модуля в основную память**

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

Регистры DS и ES имеют значение 48DD, регистр CS 48ED, а SS 48FC сегмента стека. На рис. 6 представлен EXE модуль, загруженный в отладчик.

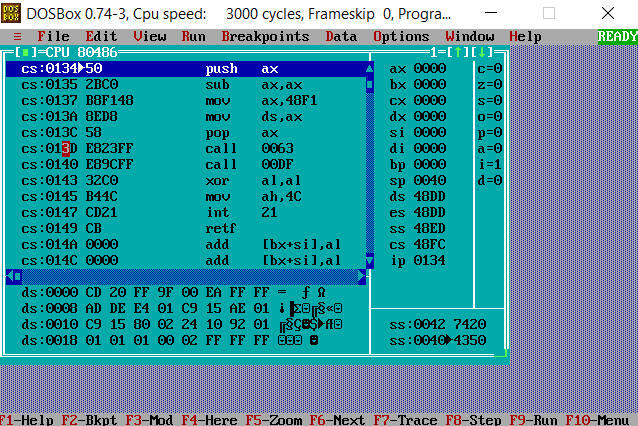


Рисунок 6 – Рисунок 5 – EXE модуль открытый в отладчике

2) На что указывают регистры DS и ES?

Регистры DS и ES указывают на начало PSP.

3) Как определяется стек?

Сегмент стека может выделятся с помощью директивы stack либо автоматически в соответствии с моделью памяти.

4) Как определяется точка входа?

Точка входа определяется началом сегмента кода, либо с помощью директивы END.

**Выводы.**

В ходе работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, а также структуры файлов загрузочных модулей и способов их загрузки в основную память.

Приложение А

**Lr1com.asm**

LR1 SEGMENT

ASSUME CS:LR1, DS:LR1, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

;DATA

PC\_TYPE db "PC type - $"

OC\_VERSION db "OC version - $"

OEM\_NUM db "OEM number - $"

S\_NUM db "Serial number - $"

OEM db " $"

DOT db ".$"

T\_PC db 'PC' , 0DH, 0AH, '$'

T\_XT db 'PC/XT' , 0DH, 0AH, '$'

T\_AT db 'AT' , 0DH, 0AH, '$'

T\_PS2\_30 db 'PS2 model 30' , 0DH, 0AH, '$'

T\_PS2\_50 db 'PC2 model 50 or 60' , 0DH, 0AH, '$'

T\_PS2\_80 db 'PC2 model 80' , 0DH, 0AH, '$'

T\_PCJR db 'PCjr' , 0DH, 0AH, '$'

T\_PC\_C db 'PC Convertable' , 0DH, 0AH, '$'

T\_UNKNOWN db "Unknown type : $"

ENTER\_SYMB db 0DH, 0AH, '$'

;PROCEDURES

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

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

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\_l

or AL,30h

mov [SI],AL

end\_l:

pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

PRINT\_STRING PROC near

push AX

mov ah, 09h

int 21h

pop AX

ret

PRINT\_STRING ENDP

PRINT\_SYMBOL PROC near

push AX

push BX

push DX

mov AH, 0

mov BL, 16

div BL

mov DX, AX

mov AH, 02h

add DL, '0'

int 21h

mov DL, DH

add DL, '0'

int 21h

pop DX

pop BX

pop AX

ret

PRINT\_SYMBOL ENDP

PRINT\_PC\_TYPE PROC near

push AX

push ES

push DX

mov AX, 0F000h

mov ES, AX

mov AL, ES:[0FFFEh]

mov dx, offset PC\_TYPE

call PRINT\_STRING

cmp AL, 0FFh

je pc\_t

cmp AL, 0FEh

je xt\_t

cmp AL, 0FCh

je at\_t

cmp AL, 0FAh

je ps2\_30\_t

cmp AL, 0FCh

je ps2\_50\_t

cmp AL, 0F8h

je ps2\_80\_t

cmp AL, 0FDh

je pcjr\_t

cmp AL, 0F9h

je pc\_c\_t

mov dx, offset T\_UNKNOWN

call PRINT\_STRING

call BYTE\_TO\_HEX

mov dl, al

call PRINT\_SYMBOL

mov dl, ah

call PRINT\_SYMBOL

jmp p\_out

pc\_t:

mov dx, offset T\_PC

jmp print\_end

xt\_t:

mov dx, offset T\_XT

jmp print\_end

at\_t:

mov dx, offset T\_AT

jmp print\_end

ps2\_30\_t:

mov dx, offset T\_PS2\_30

jmp print\_end

ps2\_50\_t:

mov dx, offset T\_PS2\_50

jmp print\_end

ps2\_80\_t:

mov dx, offset T\_PS2\_80

jmp print\_end

pcjr\_t:

mov dx, offset T\_PCJR

jmp print\_end

pc\_c\_t:

mov dx, offset T\_PC\_C

print\_end:

call PRINT\_STRING

p\_out:

pop DX

pop ES

pop AX

ret

PRINT\_PC\_TYPE ENDP

PRINT\_OC\_VERSION Proc near

push ax

push dx

mov dx, offset OC\_VERSION

call PRINT\_STRING

mov ah, 30h

int 21h

call PRINT\_SYMBOL

mov dx, offset DOT

call PRINT\_STRING

mov al, ah

;add dl, '0'

call PRINT\_SYMBOL

mov dx, offset ENTER\_SYMB

call PRINT\_STRING

mov dx, offset OEM\_NUM

call PRINT\_STRING

mov si, offset OEM

add si, 2

mov al, bh

call BYTE\_TO\_DEC

mov dx, offset OEM

call PRINT\_STRING

mov dx, offset ENTER\_SYMB

call PRINT\_STRING

mov dx, offset S\_NUM

call PRINT\_STRING

mov al, bl

call PRINT\_SYMBOL

mov al, ch

call PRINT\_SYMBOL

mov al, cl

call PRINT\_SYMBOL

pop dx

pop ax

ret

PRINT\_OC\_VERSION ENDP

BEGIN:

call PRINT\_PC\_TYPE

call PRINT\_OC\_VERSION

xor AL, AL

mov AH, 4Ch

int 21h

LR1 ENDS

END START

Приложение Б

**Lr1exe.asm**

ASTACK SEGMENT STACK

DW 20h DUP(?)

ASTACK ENDS

DATA SEGMENT

PC\_TYPE db "PC type - $"

OC\_VERSION db 13, 10, "OC version - $"

OEM\_NUM db 13, 10, "OEM number - $"

S\_NUM db 13, 10, "Serial number - $"

OEM db " $"

DOT db ".$"

T\_PC db "PC$"

T\_XT db "PC/XT$"

T\_AT db "AT$"

T\_PS2\_30 db "PS2 model 30$"

T\_PS2\_50 db "PC2 model 50 or 60$"

T\_PS2\_80 db "PC2 model 80$"

T\_PCJR db "PCjr$"

T\_PC\_C db "PC Convertable$"

T\_UNKNOWN db "Unknown type : $"

ENTER\_SYMB db "$"

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:ASTACK

;PROCEDURES

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

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

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\_l

or AL,30h

mov [SI],AL

end\_l:

pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

PRINT\_STRING PROC near

push AX

mov ah, 09h

int 21h

pop AX

ret

PRINT\_STRING ENDP

PRINT\_SYMBOL PROC near

push AX

push BX

push DX

mov AH, 0

mov BL, 16

div BL

mov DX, AX

mov AH, 02h

add DL, '0'

int 21h

mov DL, DH

add DL, '0'

int 21h

pop DX

pop BX

pop AX

ret

PRINT\_SYMBOL ENDP

PRINT\_PC\_TYPE PROC near

push AX

push ES

push DX

mov AX, 0F000h

mov ES, AX

mov AL, ES:[0FFFEh]

mov dx, offset PC\_TYPE

call PRINT\_STRING

cmp AL, 0FFh

je pc\_t

cmp AL, 0FEh

je xt\_t

cmp AL, 0FCh

je at\_t

cmp AL, 0FAh

je ps2\_30\_t

cmp AL, 0FCh

je ps2\_50\_t

cmp AL, 0F8h

je ps2\_80\_t

cmp AL, 0FDh

je pcjr\_t

cmp AL, 0F9h

je pc\_c\_t

mov dx, offset T\_UNKNOWN

call PRINT\_STRING

call BYTE\_TO\_HEX

mov dl, al

call PRINT\_SYMBOL

mov dl, ah

call PRINT\_SYMBOL

jmp p\_out

pc\_t:

mov dx, offset T\_PC

jmp print\_end

xt\_t:

mov dx, offset T\_XT

jmp print\_end

at\_t:

mov dx, offset T\_AT

jmp print\_end

ps2\_30\_t:

mov dx, offset T\_PS2\_30

jmp print\_end

ps2\_50\_t:

mov dx, offset T\_PS2\_50

jmp print\_end

ps2\_80\_t:

mov dx, offset T\_PS2\_80

jmp print\_end

pcjr\_t:

mov dx, offset T\_PCJR

jmp print\_end

pc\_c\_t:

mov dx, offset T\_PC\_C

print\_end:

call PRINT\_STRING

p\_out:

pop DX

pop ES

pop AX

ret

PRINT\_PC\_TYPE ENDP

PRINT\_OC\_VERSION Proc near

push ax

push dx

mov dx, offset OC\_VERSION

call PRINT\_STRING

mov ah, 30h

int 21h

call PRINT\_SYMBOL

mov dx, offset DOT

call PRINT\_STRING

mov al, ah

;add dl, '0'

call PRINT\_SYMBOL

mov dx, offset ENTER\_SYMB

call PRINT\_STRING

mov dx, offset OEM\_NUM

call PRINT\_STRING

mov si, offset OEM

add si, 2

mov al, bh

call BYTE\_TO\_DEC

mov dx, offset OEM

call PRINT\_STRING

mov dx, offset ENTER\_SYMB

call PRINT\_STRING

mov dx, offset S\_NUM

call PRINT\_STRING

mov al, bl

call PRINT\_SYMBOL

mov al, ch

call PRINT\_SYMBOL

mov al, cl

call PRINT\_SYMBOL

pop dx

pop ax

ret

PRINT\_OC\_VERSION ENDP

MAIN PROC FAR

push ax

sub AX,AX

mov AX,DATA

mov DS,AX

pop ax

call PRINT\_PC\_TYPE

call PRINT\_OC\_VERSION

xor AL, AL

mov AH, 4Ch

int 21h

ret

MAIN ENDP

CODE ENDS

END MAIN