**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. 6382 |  | Сергеев А.Д. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2019

**Цель работы.**

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

**Основные теоретические положения.**

Тип IBM PC хранится в байте по адресу 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Соответствия кодов и типов представлены в таблице 1.

Таблица 1 — Соответствие кодов и типов.

|  |  |
| --- | --- |
| Тип 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 MS DOS следует воспользоваться функцией 30H прерывания 21H.Входным параметром является номер функции в AH:

mov ah, 30h

int 21h

Выходными параметрами являются:

AL – номер основной версии. Если 0, то < 2.0

AH – номер модификации

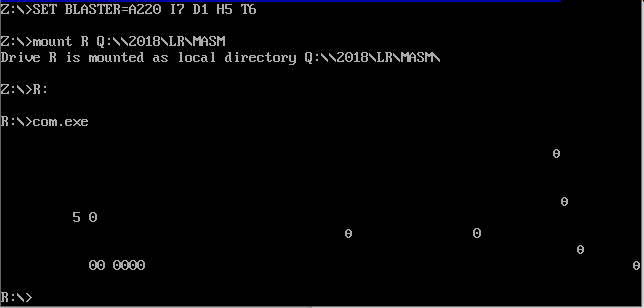
BH – серийный номер OEM (Original Equipment Manufacturer)

BL:CH – 24-битовый серийный номер пользователя

**Порядок выполнения работы.**

Написание работы производилось на базе операционной системы Windows 10 в редакторе Atom. Отладка и тестирование проводились в эмуляторе DOSBOX.

Был написан исходный код .COM модуля, он был собран с помощью программ masm и link. В результате был получен «плохой» .EXE модуль, который был преобразован в «хороший» .COM модуль при помощи программы exe2bin. Результат выполнения «плохого» .EXE модуля представлен на рисунке 1.

Рисунок 1 — вывод «плохого» .EXE модуля

Результат выполнения «хорошего» .COM модуля представлен на рисунке 2.

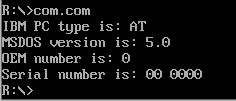


Рисунок 2 — вывод «хорошего» .EXE модуля

Был написан исходный код .EXE модуля, в результате его отладки и сборки был получен «хороший» .EXE модуль. Результат его работы представлен на рисунке 3.

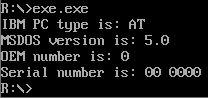


Рисунок 3 — результат работы «хорошего» .EXE модуля

**Ответы на контрольные вопросы.**

1. **Отличия исходных кодов .COM и .EXE программ**

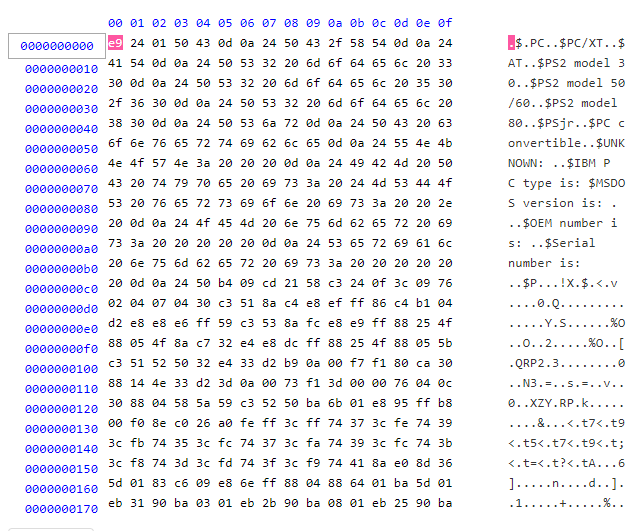
В отличии от .EXE программы, которая может содержать несколько сегментов (при использовании различных моделей памяти), .COM программа содержит только один сегмент (модель памяти tiny). Также в .EXE программе должен содержаться сегмент стека — в случае, если он не был создан программистом, операционная система создаст его автоматически.

В тексте .COM программы обязательно должны быть директивы org, которая используется для резервирования первых нескольких байт для префикса программного сегмента (PSP) и устанавливает адрес начала выполнения программы, и assume, которая переопределяет системные регистры (в данном случае необходимо, чтобы регистр кода CS и регистр данных DS указывали на единственный сегмент программы).

В .COM программе можно использовать не все форматы команд. Так как таблица настройки с информацией о местоположении адресов отсутствует, нельзя использовать команды, связанные с адресом сегмента, он не известен до загрузки сегмента в память.

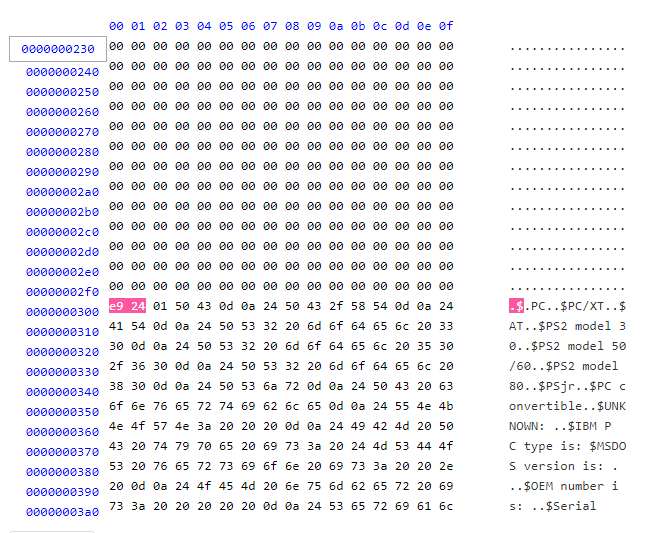
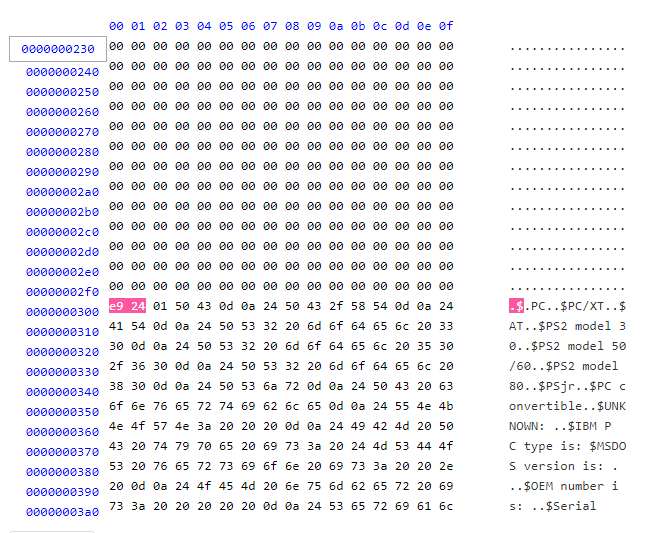
1. **Отличия форматов файлов COM и EXE модулей**

На рисунке 4 представлен вид .COM файла в шестнадцатеричном виде.

Рисунок 4 — .COM файл в шестнадцатеричном виде

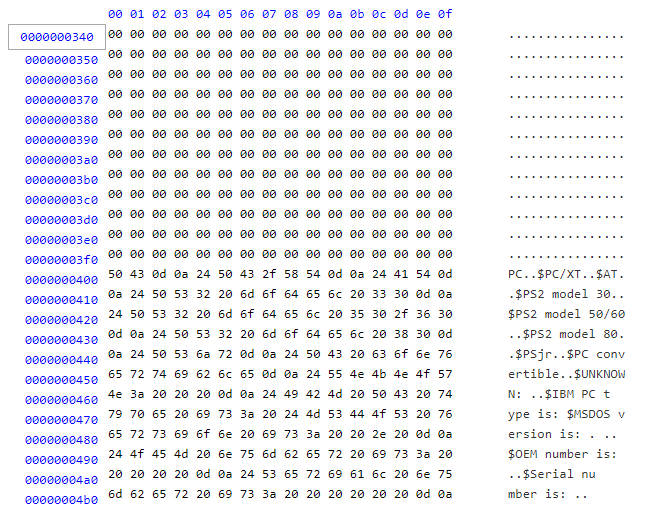
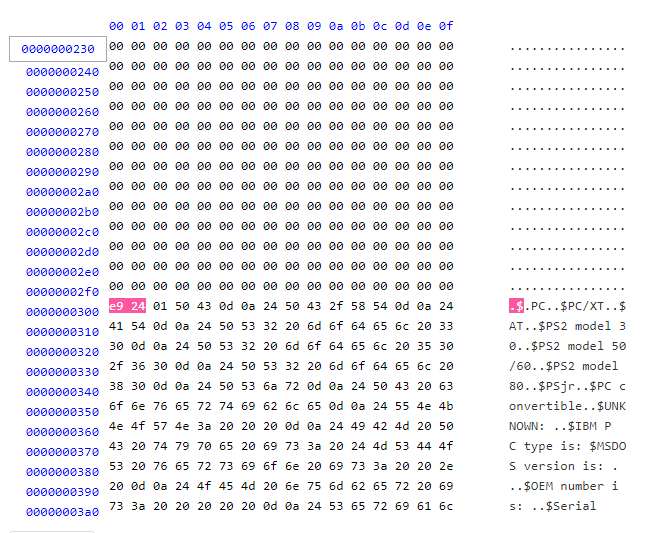
Можно заметить, что текст программы начинается с нулевого адреса, секция данных заканчивается на 0b0, на 0c0 начинается код. Между ними есть 6 пустых байт.

На рисунке 5 представлен вид «плохого» .EXE файла в шестнадцатеричном виде (с адреса начала кода).

Рисунок 5 — «плохой» .EXE файл в шестнадцатеричном виде

С нулевого адреса в файле находится информация для загрузчика, сегменты должны начаться с адреса 200h, но в данном случае начинаются с 300h, так как дополнительные 100h байтов зарезервировано командой org 100h.

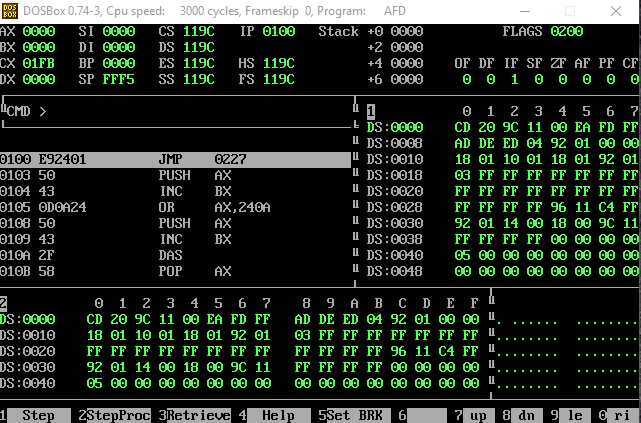
На рисунке 6 представлен вид «хорошего» .EXE файла в шестнадцатеричном виде (с адреса начала кода).

Рисунок 6 — «хороший» .EXE файл в шестнадцатеричном виде

В отличии от «плохого» .EXE файла, «хороший» содержит в себе сегмент стека, который также начинается с адреса 200h и занимает 100h двойных байтов. Таким образом сегмент данных и кода начинаются по адресу 400h.

1. **Загрузка COM модуля в основную память**

На рисунке 7 представлен вид запущенного .COM модуля в отладчике afd.

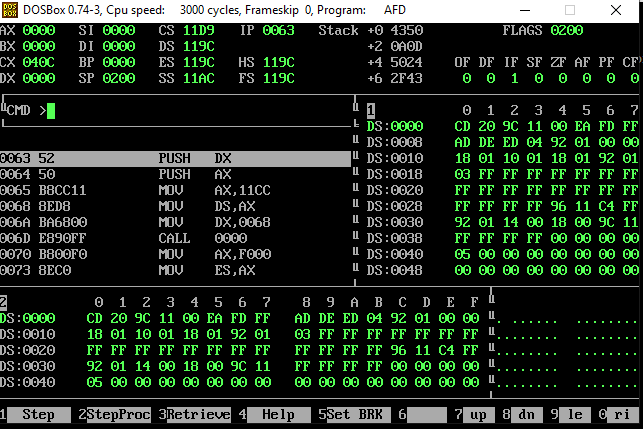
Рисунок 6 — вид запущенного COM модуля в отладчике afd

Все регистры указывают на нулевой адрес, он же - начало PSP, а код и данные располагаются по адресу 100h. Первая команда осуществляет переход к коду.

Стек в .COM модуле создается автоматически и указывает на FFF5 (регистр SP). Адреса в нем нумеруются от больших к меньшим, то есть стек занимает всю память, оставшуюся от 64кб (максимальный размер COM модуля в памяти) после загрузки данных и кода.

1. **Загрузка «хорошего» EXE модуля в основную память**

На рисунке 8 представлен вид запущенного «хорошего» .EXE модуля в отладчике afd.

Рисунок 7 — вид запущенного «хорошего» .EXE модуля в отладчике afd

После запуска регистры DS и ES указывают на PSP, а CS и SS – на начало сегмента кода и стека соответственно. Необходимость в начале работы программы загрузки в DS сегмента данных — одна из отличительных особенностей кода «хорошего» .EXE модуля.

В данном случае стек был объявлен явно, создан, и в начале работы программы его адрес хранится в регистре SS. Если бы он объявлен не был, он был бы создан автоматически так же, как при загрузке .COM модуля.

Точка входа в программу определяется оператором END, который находится в конце кода. Операндом может служить функция или метка, адрес которой в во время загрузки программы в память помещается в регистр IP.

**Вывод.**

В ходе выполнения лабораторной работы были изучены структурные различия COM и EXE модулей, а также различия их загрузки в память.

**Приложение А**

**Исходный код программы com.asm**

TESTPC SEGMENT

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

org 100H

START: jmp BEGIN

PC\_TYPE db 'PC', 0DH, 0AH,'$'

PC\_XT\_TYPE db 'PC/XT', 0dh, 0ah,'$'

AT\_TYPE db 'AT', 0dh, 0ah, '$'

PS2\_30\_TYPE db 'PS2 model 30', 0dh, 0ah, '$'

PS2\_5060\_TYPE db 'PS2 model 50/60', 0dh, 0ah, '$'

PS2\_80\_TYPE db 'PS2 model 80', 0dh, 0ah, '$'

PCJR\_TYPE db 'PSjr', 0dh, 0ah, '$'

PC\_CONVERTIBLE db 'PC convertible', 0dh, 0ah, '$'

PC\_UNKNOWN db 'UNKNOWN: ', 0dh, 0ah, '$'

IBM\_PC\_NAME db 'IBM PC type is: ', '$'

OS\_NAME db 'MSDOS version is: . ', 0dh, 0ah, '$'

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

SERIAL\_NAME db 'Serial number is: ', 0dh, 0ah, '$'

PRINT\_STRING PROC near

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT\_STRING ENDP

;--------------------------------------------------------------------------------

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 al, ah

call TETR\_TO\_HEX

xchg al, ah

mov cl, 4

shr al, cl

call TETR\_TO\_HEX

pop cx

ret

BYTE\_TO\_HEX ENDP

;--------------------------------------------------------------------------------

WRD\_TO\_HEX PROC near

push bx

mov bh, ah

call BYTE\_TO\_HEX

mov [di], ah

dec di

mov [di], al

dec di

mov al, bh

xor ah, ah

call BYTE\_TO\_HEX

mov [di], ah

dec di

mov [di], al

pop bx

ret

WRD\_TO\_HEX ENDP

;--------------------------------------------------------------------------------

BYTE\_TO\_DEC PROC near

push cx

push dx

push ax

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 ax, 00h

jbe end\_l

or al, 30h

mov [si], al

end\_l: pop ax

pop dx

pop cx

ret

BYTE\_TO\_DEC ENDP

BEGIN:

push dx

push ax

mov dx, offset IBM\_PC\_NAME

call PRINT\_STRING

mov ax, 0F000H

mov es, ax

mov al, es:[0FFFEH]

cmp al, 0FFh

je PC\_WRITE

cmp al, 0FEh

je PC\_XT\_WRITE

cmp al, 0FBh

je PC\_XT\_WRITE

cmp al, 0FCh

je AT\_WRITE

cmp al, 0FAh

je PS2\_30\_WRITE

cmp al, 0FCh

je PS2\_5060\_WRITE

cmp al, 0F8h

je PS2\_80\_WRITE

cmp al, 0FDh

je PCJR\_WRITE

cmp al, 0F9H

je PC\_CONVERTIBLE\_WRITE

mov ah, al

lea si, PC\_UNKNOWN

add si, 9

call BYTE\_TO\_HEX

mov [si], al

mov [si+1], ah

mov dx, offset PC\_UNKNOWN

jmp TYPE\_WRITE

PC\_WRITE:

mov dx, offset PC\_TYPE

jmp TYPE\_WRITE

PC\_XT\_WRITE:

mov dx, offset PC\_XT\_TYPE

jmp TYPE\_WRITE

AT\_WRITE:

mov dx, offset AT\_TYPE

jmp TYPE\_WRITE

PS2\_30\_WRITE:

mov dx, offset PS2\_30\_TYPE

jmp TYPE\_WRITE

PS2\_5060\_WRITE:

mov dx, offset PS2\_5060\_TYPE

jmp TYPE\_WRITE

PS2\_80\_WRITE:

mov dx, offset PS2\_80\_TYPE

jmp TYPE\_WRITE

PCJR\_WRITE:

mov dx, offset PCJR\_TYPE

jmp TYPE\_WRITE

PC\_CONVERTIBLE\_WRITE:

mov dx, offset PC\_CONVERTIBLE

jmp TYPE\_WRITE

TYPE\_WRITE:

call PRINT\_STRING

OS\_INFO\_GET:

mov ah, 30h

int 21h

OS\_VERSION\_SET:

lea si, OS\_NAME

add si, 18

call BYTE\_TO\_DEC

add si, 3

mov al, ah

call BYTE\_TO\_DEC

OS\_VERSION\_WRITE:

mov dx, offset OS\_NAME

call PRINT\_STRING

OEM\_SET:

mov al, BH

lea si, OEM\_NAME

add si, 15

call BYTE\_TO\_DEC

OEM\_WRITE:

mov dx, offset OEM\_NAME

call PRINT\_STRING

SERIAL\_SET:

mov al, bl

lea si, SERIAL\_NAME

add si, 18

call BYTE\_TO\_HEX

mov [si], ax

add si, 6

mov di, si

mov ax, cx

call WRD\_TO\_HEX

SERIAL\_WRITE:

mov dx, offset SERIAL\_NAME

call PRINT\_STRING

ENDING:

pop ax

pop dx

xor al, al

mov ah, 4ch

int 21h

ret

TESTPC ENDS

END START

**Приложение Б**

**Исходный код программы  exe.asm**

AStack SEGMENT STACK

DW 100h DUP(?)

AStack ENDS

DATA SEGMENT

PC\_TYPE db 'PC', 0DH, 0AH,'$'

PC\_XT\_TYPE db 'PC/XT', 0dh, 0ah,'$'

AT\_TYPE db 'AT', 0dh, 0ah, '$'

PS2\_30\_TYPE db 'PS2 model 30', 0dh, 0ah, '$'

PS2\_5060\_TYPE db 'PS2 model 50/60', 0dh, 0ah, '$'

PS2\_80\_TYPE db 'PS2 model 80', 0dh, 0ah, '$'

PCJR\_TYPE db 'PSjr', 0dh, 0ah, '$'

PC\_CONVERTIBLE db 'PC convertible', 0dh, 0ah, '$'

PC\_UNKNOWN db 'UNKNOWN: ', 0dh, 0ah, '$'

IBM\_PC\_NAME db 'IBM PC type is: ', '$'

OS\_NAME db 'MSDOS version is: . ', 0dh, 0ah, '$'

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

SERIAL\_NAME db 'Serial number is: ', 0dh, 0ah, '$'

DATA ENDS

CODE SEGMENT

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

PRINT\_STRING PROC near

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT\_STRING ENDP

;--------------------------------------------------------------------------------

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 al, ah

call TETR\_TO\_HEX

xchg al, ah

mov cl, 4

shr al, cl

call TETR\_TO\_HEX

pop cx

ret

BYTE\_TO\_HEX ENDP

;--------------------------------------------------------------------------------

WRD\_TO\_HEX PROC near

push bx

mov bh, ah

call BYTE\_TO\_HEX

mov [di], ah

dec di

mov [di], al

dec di

mov al, bh

xor ah, ah

call BYTE\_TO\_HEX

mov [di], ah

dec di

mov [di], al

pop bx

ret

WRD\_TO\_HEX ENDP

;--------------------------------------------------------------------------------

BYTE\_TO\_DEC PROC near

push cx

push dx

push ax

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 ax, 00h

jbe end\_l

or al, 30h

mov [si], al

end\_l: pop ax

pop dx

pop cx

ret

BYTE\_TO\_DEC ENDP

Main PROC far

push dx

push ax

mov ax, DATA ; diff!

mov ds, ax ; diff!

mov dx, offset IBM\_PC\_NAME

call PRINT\_STRING

mov ax, 0F000H

mov es, ax

mov al, es:[0FFFEH]

cmp al, 0FFh

je PC\_WRITE

cmp al, 0FEh

je PC\_XT\_WRITE

cmp al, 0FBh

je PC\_XT\_WRITE

cmp al, 0FCh

je AT\_WRITE

cmp al, 0FAh

je PS2\_30\_WRITE

cmp al, 0FCh

je PS2\_5060\_WRITE

cmp al, 0F8h

je PS2\_80\_WRITE

cmp al, 0FDh

je PCJR\_WRITE

cmp al, 0F9H

je PC\_CONVERTIBLE\_WRITE

mov ah, al

lea si, PC\_UNKNOWN

add si, 9

call BYTE\_TO\_HEX

mov [si], al

mov [si+1], ah

mov dx, offset PC\_UNKNOWN

jmp TYPE\_WRITE

PC\_WRITE:

mov dx, offset PC\_TYPE

jmp TYPE\_WRITE

PC\_XT\_WRITE:

mov dx, offset PC\_XT\_TYPE

jmp TYPE\_WRITE

AT\_WRITE:

mov dx, offset AT\_TYPE

jmp TYPE\_WRITE

PS2\_30\_WRITE:

mov dx, offset PS2\_30\_TYPE

jmp TYPE\_WRITE

PS2\_5060\_WRITE:

mov dx, offset PS2\_5060\_TYPE

jmp TYPE\_WRITE

PS2\_80\_WRITE:

mov dx, offset PS2\_80\_TYPE

jmp TYPE\_WRITE

PCJR\_WRITE:

mov dx, offset PCJR\_TYPE

jmp TYPE\_WRITE

PC\_CONVERTIBLE\_WRITE:

mov dx, offset PC\_CONVERTIBLE

jmp TYPE\_WRITE

TYPE\_WRITE:

call PRINT\_STRING

OS\_INFO\_GET:

mov ah, 30h

int 21h

OS\_VERSION\_SET:

lea si, OS\_NAME

add si, 18

call BYTE\_TO\_DEC

add si, 3

mov al, ah

call BYTE\_TO\_DEC

OS\_VERSION\_WRITE:

mov dx, offset OS\_NAME

call PRINT\_STRING

OEM\_SET:

mov al, BH

lea si, OEM\_NAME

add si, 15

call BYTE\_TO\_DEC

OEM\_WRITE:

mov dx, offset OEM\_NAME

call PRINT\_STRING

SERIAL\_SET:

mov al, bl

lea si, SERIAL\_NAME

add si, 18

call BYTE\_TO\_HEX

mov [si], ax

add si, 6

mov di, si

mov ax, cx

call WRD\_TO\_HEX

SERIAL\_WRITE:

mov dx, offset SERIAL\_NAME

call PRINT\_STRING

ENDING:

pop ax

pop dx

xor al, al

mov ah, 4ch

int 21h

ret

Main ENDP

CODE ENDS

END Main