**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. 8381 |  | Муковский Д.В. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2020

**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 компьютера представлено в таблице 1:

Таблица 1 – Идентификация типа компьютера

|  |  |
| --- | --- |
| Код | Тип компьютера |
| FF | Оригинальный IBM PC |
| FE | XT, Portable PC |
| FD | PCjr |
| FC | AT |
| FB | XT с памятью 640 К на мат. плате |
| FA | PS/2 модель 25 или 30 |
| F9 | Convertible PC |
| F8 | PS/2 модели 55SX, 70, 80 |
| 9A | Compaq XT, Compaq Plus |
| 30 | Sperry PC |
| 2D | Compaq PC, Compaq Deskpro |

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:CX – 24-битовый серийный номер пользователя.

**Постановка задачи.**

Требуется реализовать текст исходного .COM модуля, который определяет тип PC 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.

**Выполнение работы.**

Первым делом был написан исходный модуль .COM, который определяет тип PC и версию системы. Полученный модуль был скомпилирован, в результате чего был получен “плохой” .EXE модуль, а с помощью программы exe2bin получен “хороший” .COM модуль. После линковки было выведено предупреждение, что сегмент стека отсутствует, что представлено на рис. 1.

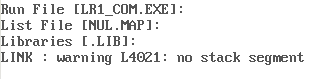


Рисунок 1 – Предупреждение об отсутствие стека

Результат выполнения «плохого» .EXE модуля представлен на рис. 2.

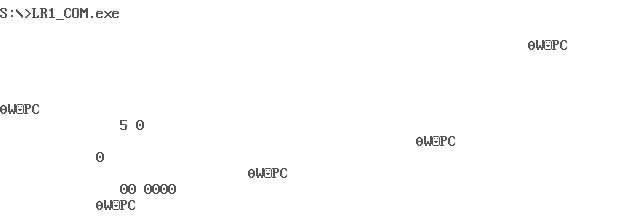


Рисунок 2 – Выполнение «плохого» .EXE модуля

Результат выполнения «хорошего» .COM модуля представлен на рис. 3.

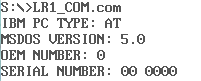


Рисунок 3 - Выполнение «хорошего» .COM модуля

Далее был написан исходный .EXE модуль. В результате компилирования был получен «хороший» .EXE модуль. Результат его выполнения представлен на рис. 4.

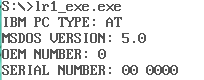


Рисунок 4 - Выполнение «хорошего» .EXE модуля

**Отличия исходных текстов COM и EXE программ.**

1. **Сколько сегментов должна содержать COM-программа?**

Один сегмент, в котором размещаются программные коды, данные и стек.

1. **EXE программа?**

Для программного кода, данных и стека предусмотрены отдельные сегменты.

1. **Какие директивы должны обязательно быть в тексте COM программы?**

Директива ORG 100h (смещение 100h), так как при загрузке СОМ-файла в память, DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы только после этого блока.

Директива ASSUME, ставящая в соответствие адрес сегмента программы сегментам кода и данных. Ошибки программы без данной директивы представлены на рис. 5.

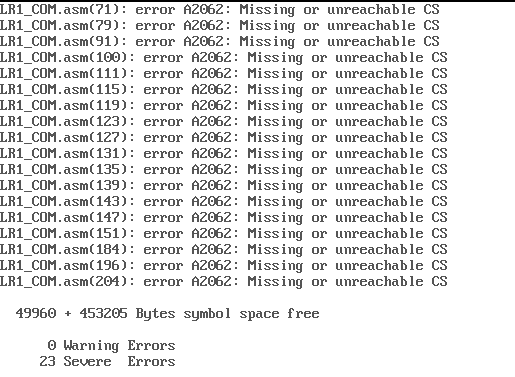


Рисунок 5 - .COM без директивы ASSUME

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

Нет, так как в отличие от EXE-программы, в которой существует таблица настроек, хранящая информацию о типе адресов и их местоположении в коде, COM-программа ею не располагает. Адреса сегментов определяются загрузчиком в момент запуска программы, а не в момент компиляции. Поэтому в COM-программах невозможно использовать команды вида: mov [регистр], seg [сегмент].

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

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

Вид файла COM в шестнадцатеричном формате представлен на рис. 6.

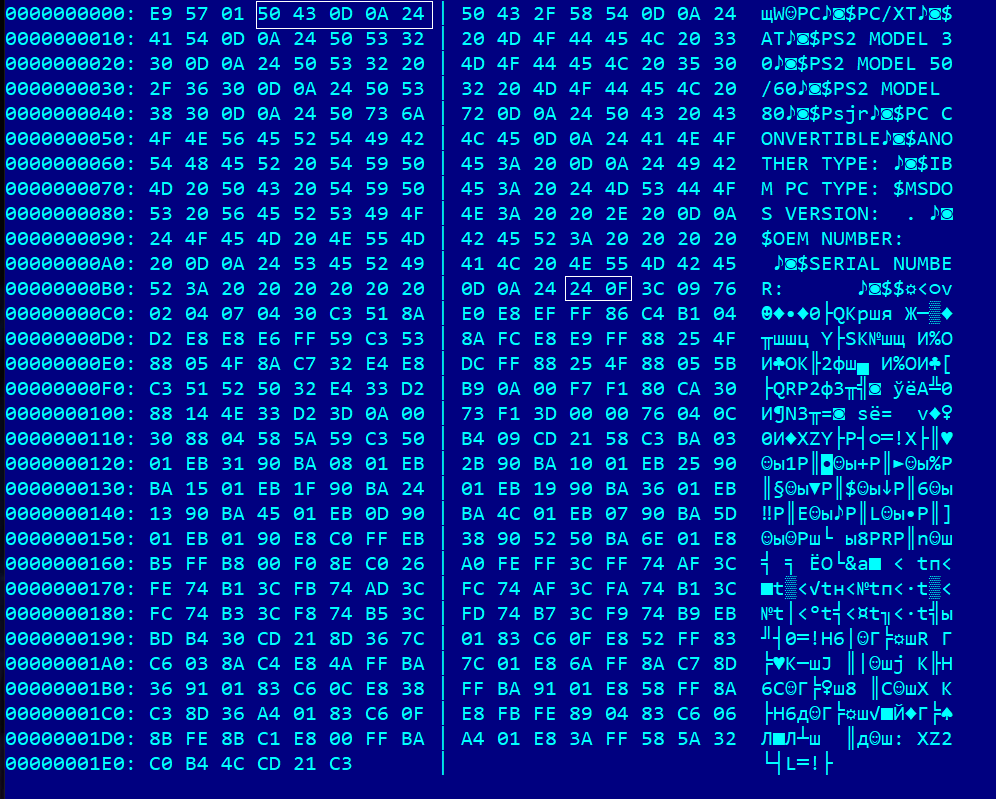
****

Рисунок 6 - .COM модуль в шестнадцатеричном виде

Данный файл состоит из одного сегмента, размер файла не может превышать 64 Кб. Код начинается с адреса 0h, но при запуске модуля устанавливается смещение в 100h.

Сначала идут байты данных, на рис.6 первым выделено слово «PC». А байты, отвечающие за код, начинаются с номера BC и на приведенном рисунке выделена первая команда процедуры TETR\_TO\_HEX (см. Приложение А).

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

Вид «плохого» EXE файла в шестнадцатеричном виде представлен на рис. 7.



Рисунок 7 – «плохой» .EXE модуль в шестнадцатеричном виде

В «плохом» .EXE файле код располагается с адреса 300h. С адреса 0h располагается таблица настроек. Также 100h резервируется командой org 100h.

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

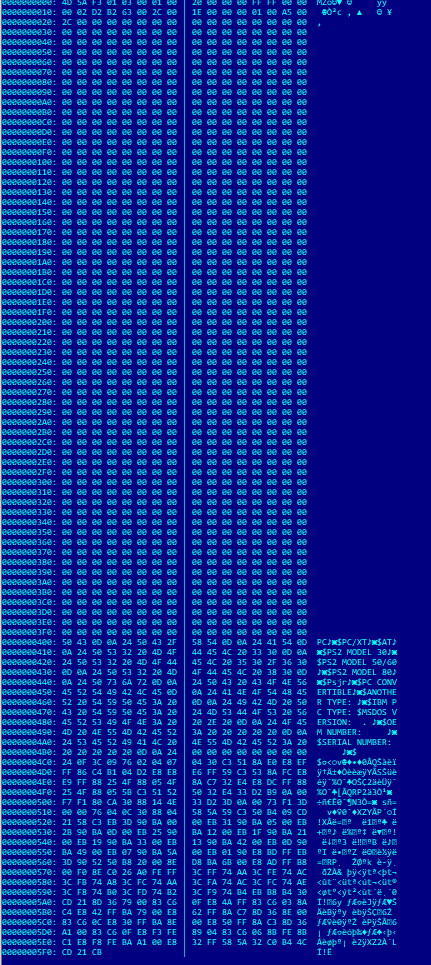


Рисунок 8 – «хороший» .EXE модуль в шестнадцатеричном виде

В «хорошем» EXE с нулевого адреса располагается таблица настроек. Также перед кодом выделена память под стек, в данном случае при размере стека 200h код располагается с адреса 400h. Еще одним отличием «хорошего» модуля является то, что в нем не резервируются дополнительно 100h, требуемые для PSP в .COM файле.

**Загрузка COM модуля в основную память**

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

Загрузка файла .COM в отладчике Turbo Debugger представлен на рис. 9.

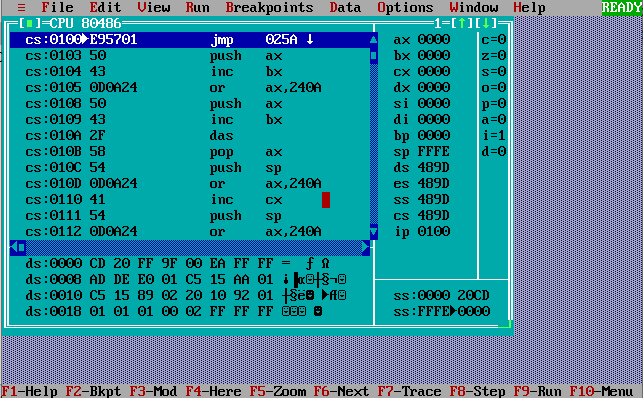


Рисунок 9 – Отладка .COM файла

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?**

С нулевого адреса располагается адрес начала PSP.

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

Сегментные регистры имеют значение 489Dh и указывают на 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 64 кб достаточен, то DOS устанавливает в регистре SP-адрес конца сегмента – FFFEh, что можно увидеть на рис. 9. Адреса расположены в диапазоне от FFFEh до 0000h.

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

Загрузка файла .EXE в отладчике Turbo Debugger представлен на рис. 10.

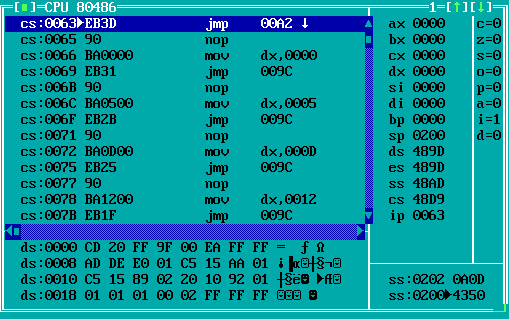
****

Рисунок 10 – Отладка .EXE файла

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

Для PSP и программы выделяется блок памяти. После запуска программы DS и ES указывают на начало PSP (489Dh), CS – на начало сегмента команд (48D9), а SS – на начало сегмента стека (48AD). IP имеет ненулевое значение, так как в программе есть дополнительные процедуры, которые расположены до основной.

**2. На что указывают регистры DS и ES?**

Изначально регистры указывают на начало PSP. Поэтому в начале программы для корректной работы регистр DS инициализируется адресом начала сегмента данных.

**3. Как определяется стек?**

Стек определяется в исходном коде при помощи директивы. STACK, а при исполнении в регистр SS записывается адрес начала сегмента стека, а в указатель стека SP- указатель на конец стека.

**4. Как определяется точка входа?**

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом директивы END, который называется точкой входа.

**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 ПРОГРАММЫ LR1\_COM.ASM**

LAB SEGMENT

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

ORG 100H

MAIN: JMP BEGIN

;данные

PC DB 'PC', 0DH, 0AH,'$'

PC\_XT DB 'PC/XT', 0DH, 0AH,'$'

AT\_ DB 'AT', 0DH, 0AH, '$'

PS2\_30 DB 'PS2 MODEL 30', 0DH, 0AH, '$'

PS2\_5060 DB 'PS2 MODEL 50/60', 0DH, 0AH, '$'

PS2\_80 DB 'PS2 MODEL 80', 0DH, 0AH, '$'

PCJR DB 'Psjr', 0DH, 0AH, '$'

PC\_CONVERTIBLE DB 'PC CONVERTIBLE', 0DH, 0AH, '$'

TYPE\_ANOTHER DB 'ANOTHER TYPE: ',0DH, 0AH, '$'

IBM\_PC\_NAME DB 'IBM PC TYPE: ', '$'

OS\_NAME DB 'MSDOS VERSION: . ', 0DH, 0AH, '$'

OEM\_NAME DB 'OEM NUMBER: ', 0DH, 0AH, '$'

SERIAL\_NAME DB 'SERIAL NUMBER: ', 0DH, 0AH, '$'

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 шест. 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 цифра

RET

BYTE\_TO\_HEX ENDP

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

WRD\_TO\_HEX PROC NEAR

;перевод в 16 с/с 16 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

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

;перевод в 10 с/с, si - адрес поля младшей цифры

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

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

PRINT PROC NEAR

PUSH AX

MOV AH, 09H

INT 21H

POP AX

RET

PRINT ENDP

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

PC\_WRITE:

MOV DX, OFFSET PC

JMP WRITE

PC\_XT\_WRITE:

MOV DX, OFFSET PC\_XT

JMP WRITE

AT\_WRITE:

MOV DX, OFFSET AT\_

JMP WRITE

PS2\_30\_WRITE:

MOV DX, OFFSET PS2\_30

JMP WRITE

PS2\_5060\_WRITE:

MOV DX, OFFSET PS2\_5060

JMP WRITE

PS2\_80\_WRITE:

MOV DX, OFFSET PS2\_80

JMP WRITE

PCJR\_WRITE:

MOV DX, OFFSET PCJR

JMP WRITE

PC\_CONVERTIBLE\_WRITE:

MOV DX, OFFSET PC\_CONVERTIBLE

JMP WRITE

PC\_ANOTHER\_WRITE:

MOV DX, OFFSET TYPE\_ANOTHER

JMP WRITE

WRITE:

CALL PRINT

JMP OS

BEGIN:

PUSH DX

PUSH AX

MOV DX, OFFSET IBM\_PC\_NAME

CALL PRINT

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

JMP PC\_ANOTHER\_WRITE

OS:

MOV AH, 30H ;дает номер версии dos al - основная версия, ah - номер модификации

INT 21H

LEA SI, OS\_NAME

ADD SI, 15

CALL BYTE\_TO\_DEC

ADD SI, 3

MOV AL, AH

CALL BYTE\_TO\_DEC

MOV DX, OFFSET OS\_NAME

CALL PRINT

OEM:

MOV AL, BH ;bh - серийный номер

LEA SI, OEM\_NAME

ADD SI, 12

CALL BYTE\_TO\_DEC

MOV DX, OFFSET OEM\_NAME

CALL PRINT

SERIAL:

MOV AL, BL

LEA SI, SERIAL\_NAME

ADD SI, 15

CALL BYTE\_TO\_HEX

MOV [SI], AX

ADD SI, 6

MOV DI, SI

MOV AX, CX

CALL WRD\_TO\_HEX

MOV DX, OFFSET SERIAL\_NAME

CALL PRINT

POP AX

POP DX

XOR AL, AL

MOV AH, 4CH

INT 21H

RET

LAB ENDS

END MAIN

**ПРИЛОЖЕНИЕ B**

**ИСХОДНЫЙ КОД ПРОГРАММЫ LR1\_EXE.ASM**

ASTACK SEGMENT STACK

DW 100h DUP(?)

ASTACK ENDS

;данные

DATA SEGMENT

PC DB 'PC', 0DH, 0AH,'$'

PC\_XT DB 'PC/XT', 0DH, 0AH,'$'

AT\_ DB 'AT', 0DH, 0AH, '$'

PS2\_30 DB 'PS2 MODEL 30', 0DH, 0AH, '$'

PS2\_5060 DB 'PS2 MODEL 50/60', 0DH, 0AH, '$'

PS2\_80 DB 'PS2 MODEL 80', 0DH, 0AH, '$'

PCJR DB 'Psjr', 0DH, 0AH, '$'

PC\_CONVERTIBLE DB 'PC CONVERTIBLE', 0DH, 0AH, '$'

TYPE\_ANOTHER DB 'ANOTHER TYPE: ',0DH, 0AH, '$'

IBM\_PC\_NAME DB 'IBM PC TYPE: ', '$'

OS\_NAME DB 'MSDOS VERSION: . ', 0DH, 0AH, '$'

OEM\_NAME DB 'OEM NUMBER: ', 0DH, 0AH, '$'

SERIAL\_NAME DB 'SERIAL NUMBER: ', 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 шест. 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 цифра

RET

BYTE\_TO\_HEX ENDP

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

WRD\_TO\_HEX PROC NEAR

;перевод в 16 с/с 16 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

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

;перевод в 10 с/с, si - адрес поля младшей цифры

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

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

PRINT PROC NEAR

PUSH AX

MOV AH, 09H

INT 21H

POP AX

RET

PRINT ENDP

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

MAIN PROC FAR

JMP BEGIN

PC\_WRITE:

MOV DX, OFFSET PC

JMP WRITE

PC\_XT\_WRITE:

MOV DX, OFFSET PC\_XT

JMP WRITE

AT\_WRITE:

MOV DX, OFFSET AT\_

JMP WRITE

PS2\_30\_WRITE:

MOV DX, OFFSET PS2\_30

JMP WRITE

PS2\_5060\_WRITE:

MOV DX, OFFSET PS2\_5060

JMP WRITE

PS2\_80\_WRITE:

MOV DX, OFFSET PS2\_80

JMP WRITE

PCJR\_WRITE:

MOV DX, OFFSET PCJR

JMP WRITE

PC\_CONVERTIBLE\_WRITE:

MOV DX, OFFSET PC\_CONVERTIBLE

JMP WRITE

PC\_ANOTHER\_WRITE:

MOV DX, OFFSET TYPE\_ANOTHER

JMP WRITE

WRITE:

CALL PRINT

JMP OS

BEGIN:

PUSH DX

PUSH AX

MOV AX, DATA

MOV DS, AX

MOV DX, OFFSET IBM\_PC\_NAME

CALL PRINT

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

JMP PC\_ANOTHER\_WRITE

OS:

MOV AH, 30H ;дает номер версии dos al - основная версия, ah - номер модификации

INT 21H

LEA SI, OS\_NAME

ADD SI, 15

CALL BYTE\_TO\_DEC

ADD SI, 3

MOV AL, AH

CALL BYTE\_TO\_DEC

MOV DX, OFFSET OS\_NAME

CALL PRINT

OEM:

MOV AL, BH ;bh - серийный номер

LEA SI, OEM\_NAME

ADD SI, 12

CALL BYTE\_TO\_DEC

MOV DX, OFFSET OEM\_NAME

CALL PRINT

SERIAL:

MOV AL, BL

LEA SI, SERIAL\_NAME

ADD SI, 15

CALL BYTE\_TO\_HEX

MOV [SI], AX

ADD SI, 6

MOV DI, SI

MOV AX, CX

CALL WRD\_TO\_HEX

MOV DX, OFFSET SERIAL\_NAME

CALL PRINT

POP AX

POP DX

XOR AL, AL

MOV AH, 4CH

INT 21H

RET

MAIN ENDP

CODE ENDS

END MAIN