**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по практической работе № 2**

**по дисциплине «Операционные системы»**

Тема: **Исследование структур загрузочных модулей**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9382 |  | Демин В.В. |
| 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 .СОМ модуля, который определяет тип РС и версию системы. Это довольно простая задача и для тех, кто уже имеет опыт программирования на ассемблере, это будет небольшой разминкой. Для тех, кто раньше не сталкивался с программированием на ассемблере, это неплохая задача для первого опыта.

За основу возьмите шаблон, приведенный в разделе «Основные сведения». Необходимые сведения о том, как извлечь требуемую информацию, представлены в следующем разделе.

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» .СОМ модуль, а также необходимо построить «плохой» .ЕХЕ, полученный из исходного текста для .СОМ модуля.

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

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

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .СОМ и файл «плохого» .ЕХЕ в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» .ЕХЕ и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов СОМ и ЕХЕ модулей».

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

Шаг 6. Откройте отладчик TD.EXE и загрузите «хороший» .ЕХЕ. Ответьте на контрольные вопросы «Загрузка «хорошего» ЕХЕ 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.

Необходимые сведения для составления программы

Тип IBM PC хранится в байте по адресу 0F000:0FFFEh, в предпоследнем байте ROM BIOS. Соответствие кода и типа в таблице:

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:CX - 24-битовый серийный номер пользователя.

**Выполнение работы.**

1.Была написан модуль .com , который определяет тип PC и версию системы. Был использован шаблон из методички, в которм были прописаны функции для перевода бинарного числа в десятичное, шестнадцатиричное число в строку из шестнадцатиричных цифр. Эти функции использовались, чтобы вывести на экран в нужном формате тип PC и версию системы, так как этот результат записывался в регистры. А для определения типа PC, мы взяли готовые названия типов и в зависимости от значения в AL, выводили на экран соответсвующие наименования.

2. После чего данная программа была переписана для модуля .exe.

Запуск модуля .COM

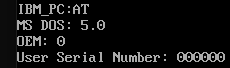


Рис.1 – «Хороший» .COM модуль

Запуск плохого .EXE

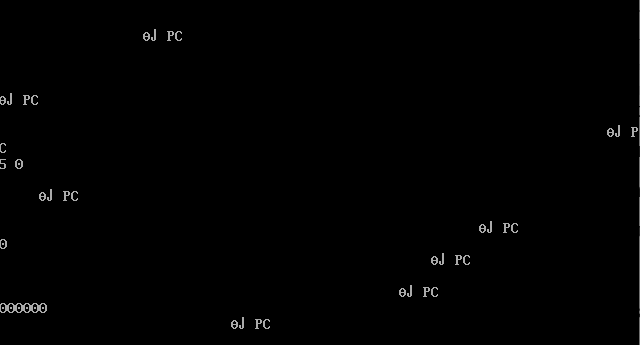


Рис.2 – «Плохой» EXE модуль

Запуск хорошоего .EXE

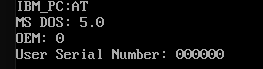
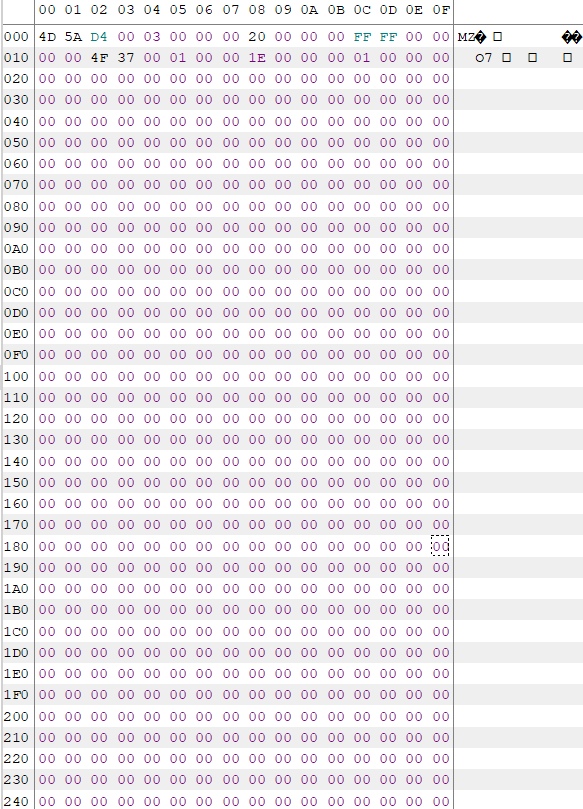


Рис. 3 – «Хороший» EXE модуль

Шаг 4.



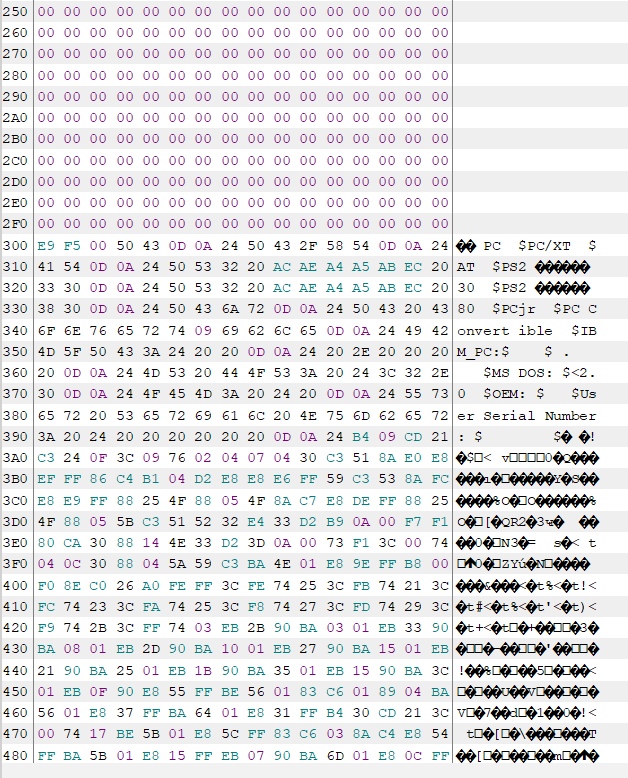


Рис.4 Файл загрузочного модуля «Плохого» .EXE в 16-ном виде

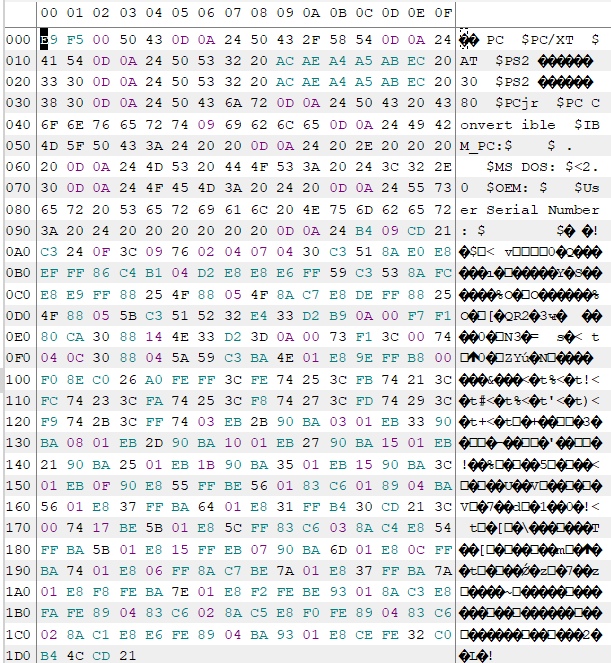
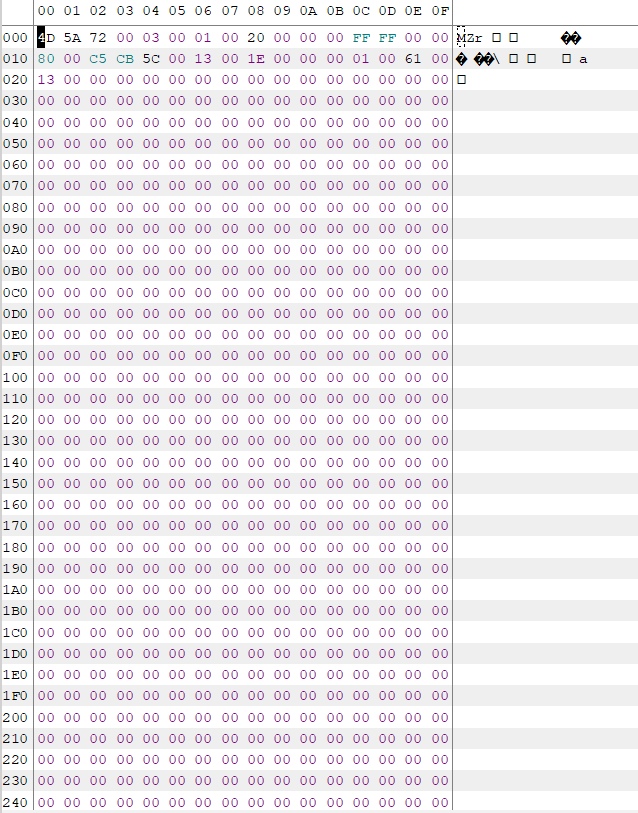


Рис.5 Файл загрузочного модуля .COM в 16-ном виде



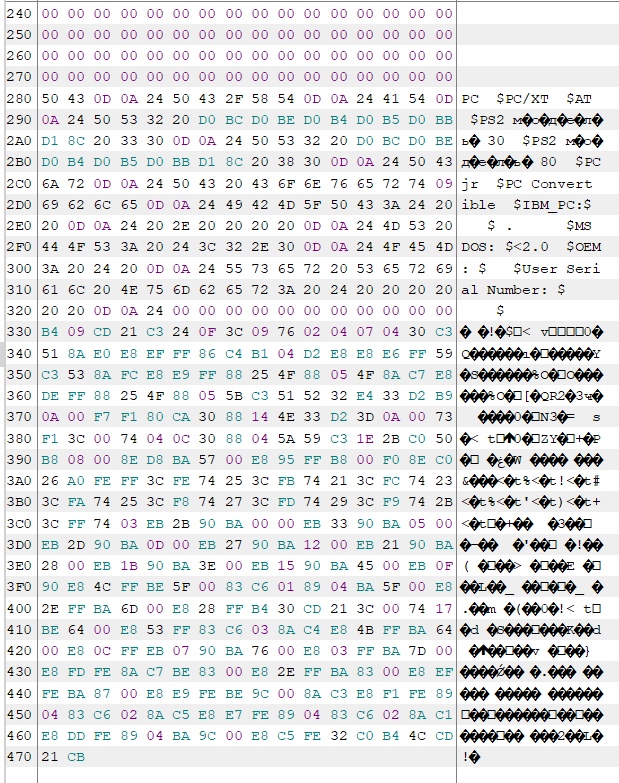


Рис.6 Файл загрузочного модуля «Хорошего» .EXE в 16-ном виде

Шаг 5

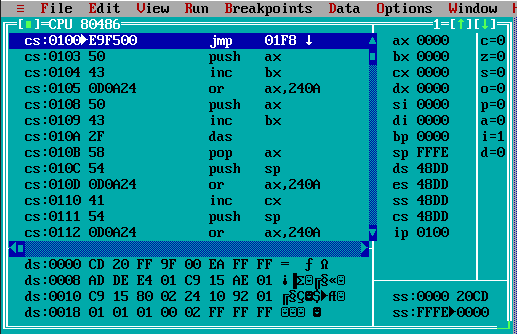


Рис. 7 отладчик TD.EXE для .COM

Шаг 6

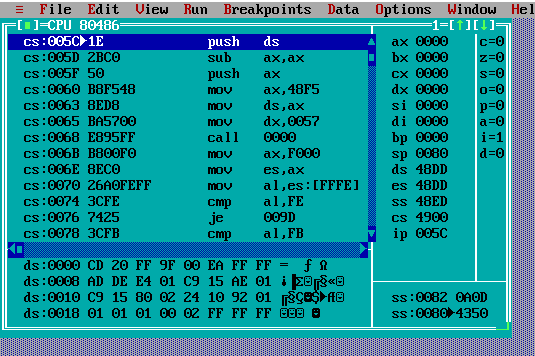


Рис. 8 отладчик TD.EXE для .EXE

**Выводы.**

Были исследованы модули .COM и .EXE, рассмотрены из различия в исходных текстах и различия готовых модулей. Также были рассмотрены способы загрузки модулей в основную память.

ПРИЛОЖЕНИЕ А

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

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

1. Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать один сегмент.

1. EXE-программа?

Один сегмент и более.

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

ORG 100h – обязательная директива для COM-программы, так как при загрузке модуля в оперативную память, сначала грузится PSP размером в 100h. И директива нужна, чтобы задать смещение для адресации.

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

Например, нельзя использовать:

mov ax, DATA

Где DATA – адрес сегмента кода

**Отличия форматов файлов .COM и.EXE программ:**

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

Файл .COM состоит из одного сегмента, его максимальный размер – 64 кб. В этот сегмент входит сегмент кода и сегмент данных. Он начинается с адреса 0h.

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

В «плохом» EXE данные и код содержатся в одном сегменте.

С адреса 0h идёт таблица настроек (Relocation table). Код располагается с адреса 300h.

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

В «хорошем» EXE данные, стек и код разделены по сегментам. Код располагается с адреса 200h в отличие от 300h в «плохом» EXE.

**Загрузка COM модуля в основную память:**

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

Код располагается с адреса 100h. При загрузки модуля COM. Сегментные регистры будут указывать на начало PSP.

1. Что располагается с адреса 0?

PSP

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

Регистры CS, DS, ES и SS указывают на PSP, имеют значения 48DD.

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

Стек генерируется автоматически при загрузке COM-программы. SS – на начало – 0h, SP – на конец – FFFEh, адрес расположен в диапазоне 0h – FFFEh.

**Загрузка «хорошего» EXE модуля в основную 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 команд. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END.

CS = 4900

SS = 48ED

ES = 48DD

DS = 48DD

1. На что указывают регистры DS и ES?

Регистры DS и ES указывают на начало сегмента PSP.

1. Как определяется стек?

При помощи директивы .STACK, также задается размер стека. Регистр SS будет указывать на начало сегмента стека, а SP на конец.

1. Как определяется точка входа?

Точка входа определяется при помощи директивы END.

ПРИЛОЖЕНИЕ Б

КОД ПРОГРАММЫ

Lab1com.asm: TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

STRING\_PC db 'PC',0DH,0AH,'$'

STRING\_PC\_XT db 'PC/XT',0DH,0AH,'$'

STRING\_AT db 'AT',0DH,0AH,'$'

STRING\_PS2\_30 db 'PS2 модель 30',0DH,0AH,'$'

STRING\_PS2\_80 db 'PS2 модель 80',0DH,0AH,'$'

STRING\_PCjr db 'PCjr',0DH,0AH,'$'

STRING\_PC\_CON db 'PC Convert ible',0DH,0AH,'$'

STRING\_IMB db 'IBM\_PC:','$'

STRING\_ANOTHER db ' ',0DH,0AH,'$'

STRING\_VERSION\_DOS db ' . ',0DH,0AH,'$'

STRING\_VERSION\_DOS\_TEXT db 'MS DOS: ','$'

STRING\_0 db '<2.0',0DH,0AH,'$'

STRING\_OEM\_TEXT db 'OEM: ', '$'

STRING\_OEM db ' ',0DH,0AH,'$'

STRING\_USER\_TEXT db 'User Serial Number: ', '$'

STRING\_USER db ' ',0DH,0AH,'$'

PRINT PROC near

mov AH, 09h ;Номер функции 09h

int 21h

ret

PRINT 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

;byte AL translate in two symbols on 16cc numbers in AX

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

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

WRD\_TO\_HEX PROC near

;translate in 16cc a 16 discharge number

;in AL - number, DI - the address of the last symbol

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

;translate in 10cc, SI - the adress of the field of younger digit

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

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

BEGIN:

ibm\_pc:

mov DX,offset STRING\_IMB

call PRINT

MOV AX,0F000H ;указывает ES на ПЗУ

MOV ES,AX ;

MOV AL,ES:[0FFFEH] ;получаем байт

CMP AL,0FEH

je PC\_XT

CMP AL,0FBH

je PC\_XT

CMP AL,0FCH

je vAT

CMP AL,0FAH

je PS2\_30

CMP AL,0F8H

je PS2\_80

CMP AL,0FDH

je PCjr

CMP AL,0F9H

je PC\_CON

CMP AL,0FFH

je PC

jmp Another

PC:

mov DX, offset STRING\_PC

jmp PRINT\_IBM\_PC

PC\_XT:

mov DX, offset STRING\_PC\_XT

jmp PRINT\_IBM\_PC

vAT:

mov DX,offset STRING\_AT

jmp PRINT\_IBM\_PC

PS2\_30:

mov DX,offset STRING\_PS2\_30

jmp PRINT\_IBM\_PC

PS2\_80:

mov DX,offset STRING\_PS2\_80

jmp PRINT\_IBM\_PC

PCjr:

mov DX,offset STRING\_PCjr

jmp PRINT\_IBM\_PC

PC\_CON:

mov DX,offset STRING\_PC\_CON

jmp PRINT\_IBM\_PC

Another:

call BYTE\_TO\_HEX

mov si, offset STRING\_ANOTHER

add si, 1

mov [si], ax

mov dx,offset STRING\_ANOTHER

PRINT\_IBM\_PC:

call PRINT

MS\_DOS:

mov dx, offset STRING\_VERSION\_DOS\_TEXT

call PRINT

MOV AH,30h

INT 21h

cmp AL,0

je NULL\_VERSION

mov si, offset STRING\_VERSION\_DOS

call BYTE\_TO\_DEC

add si,3

mov AL,AH

call BYTE\_TO\_DEC

MOV dx,offset STRING\_VERSION\_DOS

call PRINT

jmp OEM

NULL\_VERSION:

mov dx, offset STRING\_0

call PRINT

OEM:

mov dx,offset STRING\_OEM\_TEXT

call PRINT

mov AL,BH

mov si,offset STRING\_OEM

call BYTE\_TO\_DEC

mov dx, offset STRINg\_OEM

call PRINT

USER\_SERIES:

mov dx,offset STRING\_USER\_TEXT

call PRINT

mov si, offset STRING\_USER

mov al, bl

call byte\_to\_hex

mov [si], ax

add si, 2

mov al, ch

call byte\_to\_hex

mov [si], ax

add si, 2

mov al, cl

call byte\_to\_hex

mov [si], ax

mov dx, offset STRING\_USER

call PRINT

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START

LAB1EXE.ASM: AStack SEGMENT STACK

DW 64 DUP(?) ; Žâ¢®¤¨âáï 12 á«®¢ ¯ ¬ïâ¨

AStack ENDS

DATA SEGMENT

STRING\_PC db 'PC',0DH,0AH,'$'

STRING\_PC\_XT db 'PC/XT',0DH,0AH,'$'

STRING\_AT db 'AT',0DH,0AH,'$'

STRING\_PS2\_30 db 'PS2 модель 30',0DH,0AH,'$'

STRING\_PS2\_80 db 'PS2 модель 80',0DH,0AH,'$'

STRING\_PCjr db 'PCjr',0DH,0AH,'$'

STRING\_PC\_CON db 'PC Convert ible',0DH,0AH,'$'

STRING\_IMB db 'IBM\_PC:','$'

STRING\_ANOTHER db ' ',0DH,0AH,'$'

STRING\_VERSION\_DOS db ' . ',0DH,0AH,'$'

STRING\_VERSION\_DOS\_TEXT db 'MS DOS: ','$'

STRING\_0 db '<2.0',0DH,0AH,'$'

STRING\_OEM\_TEXT db 'OEM: ', '$'

STRING\_OEM db ' ',0DH,0AH,'$'

STRING\_USER\_TEXT db 'User Serial Number: ', '$'

STRING\_USER db ' ',0DH,0AH,'$'

DATA ENDS

CODE SEGMENT

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

PRINT PROC near

mov AH, 09h ;Номер функции 09h

int 21h

ret

PRINT 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

;byte AL translate in two symbols on 16cc numbers in AX

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

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

WRD\_TO\_HEX PROC near

;translate in 16cc a 16 discharge number

;in AL - number, DI - the address of the last symbol

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

;translate in 10cc, SI - the adress of the field of younger digit

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

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

Main PROC FAR

push DS

sub AX,AX

push AX

mov AX,DATA

mov DS,AX

ibm\_pc:

mov DX,offset STRING\_IMB

call PRINT

MOV AX,0F000H ;указывает ES на ПЗУ

MOV ES,AX ;

MOV AL,ES:[0FFFEH] ;получаем байт

CMP AL,0FEH

je PC\_XT

CMP AL,0FBH

je PC\_XT

CMP AL,0FCH

je vAT

CMP AL,0FAH

je PS2\_30

CMP AL,0F8H

je PS2\_80

CMP AL,0FDH

je PCjr

CMP AL,0F9H

je PC\_CON

CMP AL,0FFH

je PC

jmp Another

PC:

mov DX, offset STRING\_PC

jmp PRINT\_IBM\_PC

PC\_XT:

mov DX, offset STRING\_PC\_XT

jmp PRINT\_IBM\_PC

vAT:

mov DX,offset STRING\_AT

jmp PRINT\_IBM\_PC

PS2\_30:

mov DX,offset STRING\_PS2\_30

jmp PRINT\_IBM\_PC

PS2\_80:

mov DX,offset STRING\_PS2\_80

jmp PRINT\_IBM\_PC

PCjr:

mov DX,offset STRING\_PCjr

jmp PRINT\_IBM\_PC

PC\_CON:

mov DX,offset STRING\_PC\_CON

jmp PRINT\_IBM\_PC

Another:

call BYTE\_TO\_HEX

mov si, offset STRING\_ANOTHER

add si, 1

mov [si], ax

mov dx,offset STRING\_ANOTHER

PRINT\_IBM\_PC:

call PRINT

MS\_DOS:

mov dx, offset STRING\_VERSION\_DOS\_TEXT

call PRINT

MOV AH,30h

INT 21h

cmp AL,0

je NULL\_VERSION

mov si, offset STRING\_VERSION\_DOS

call BYTE\_TO\_DEC

add si,3

mov AL,AH

call BYTE\_TO\_DEC

MOV dx,offset STRING\_VERSION\_DOS

call PRINT

jmp OEM

NULL\_VERSION:

mov dx, offset STRING\_0

call PRINT

OEM:

mov dx,offset STRING\_OEM\_TEXT

call PRINT

mov AL,BH

mov si,offset STRING\_OEM

call BYTE\_TO\_DEC

mov dx, offset STRINg\_OEM

call PRINT

USER\_SERIES:

mov dx,offset STRING\_USER\_TEXT

call PRINT

mov si, offset STRING\_USER

mov al, bl

call byte\_to\_hex

mov [si], ax

add si, 2

mov al, ch

call byte\_to\_hex

mov [si], ax

add si, 2

mov al, cl

call byte\_to\_hex

mov [si], ax

mov dx, offset STRING\_USER

call PRINT

xor AL,AL

mov AH,4Ch

int 21H

ret

Main ENDP

CODE ENDS

END Main