**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Операционные системы»**

Тема: “Исследование структур заголовочных модулей”

|  |  |  |
| --- | --- | --- |
| Студент гр. 7381 |  | Ильясов А.В. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2019

**Цель работы.**

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

**Основные теоретические положения.**

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 переводится в символьную строку, содержащую запись шестнадцатеричного числа и выводится на экран в виде соответствующего сообщения.

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

|  |  |
| --- | --- |
| РС | FF |
| РС/XT | FE, FB |
| AT | FC |
| PS2 модель 30 | FA |
| PS2 модель 50 или 60 | FC |
| PS2 модель 80 | F8 |
| PCjr | FD |
| PC Convertible | F9 |

Таблица 1 – таблица соответствия кода и типа IBM PC

Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате хх.уу, где хх – номер основной версии, а уу – номер модификации в десятичной системе счисления, формировать строки с серийным номером ОЕМ и серийным номером пользователя. Полученные строки выводятся на экран.

Для определения версии MS DOS следует воспользоваться функцией 30Н прерывания 21Н. входным параметром является номер функции в АН:

MOV AH, 30h

INT 21H

Выходными параметрами являются:

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

AH – номер модификации

ВН – серийный номер ОЕМ

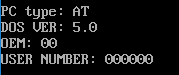
BL:CX – 24-битовый серийный номер пользователя.

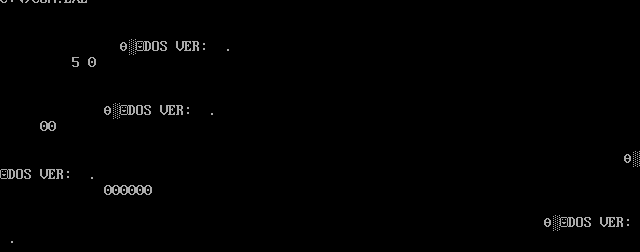
**Выполнение работы.**

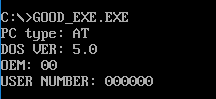
Написан текст исходного .COM модуля в соответствии с заданием. Были написаны процедуры получения типа IBM PC, версии системы, ОЕМ и серийного номера пользователя. Также были добавлены процедуры перевода кодов в различные системы счисления из методических указаний. Исходный код программы представлен в приложении А.

Написан текст исходного .EXE модуля с тем же функционалом. Исходный код программы представлен в приложении Б.

Ниже представлены результаты выполнения программы.

Рисунок 1 – результат работы .COM модуля

Рисунок 2 – результат работы «плохого» .ЕХЕ модуля

Рисунок 3 – результат работы «хорошего» .ЕХЕ модуля

**Выводы.**

Исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов загрузки в основную память. Реализована программа на языке ассемблера позволяющая определить тип IBM PC и тип системы.

**Ответы на контрольные вопросы.**

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

1. Сколько сегментов должна содержать COM-программа?

Ответ: один сегмент кода.

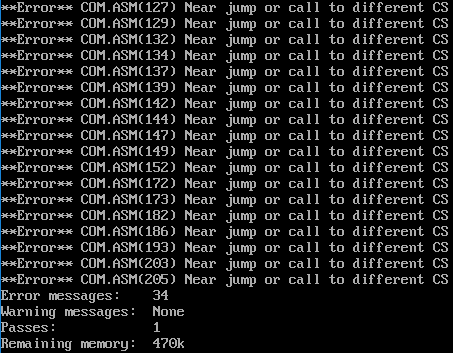
1. EXE-программа?

Ответ: «хорошая» .ЕХЕ-программа содержит 3 сегмента: сегмент стека, сегмент данных и сегмент кода.

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

Ответ: должна присутствовать директива ORG 100h для выделения памяти под PSP.

Дополнение ответа: также необходимо привязать каждый регистр к нужному сегменту. За это отвечает директива ASSUME. Если закомментировать строку с этой директивой, то компилятор выдаст ошибку «near jump or call to different cs» (см. рис. 4). Стоит заметить, что ошибки появляются в месте вызовов функций, то есть, компилятор не может понять из какого сегмента брать вызываемые функции.

рисунок 4 – результат компиляции с закомментированной директивой ASSUME

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

Ответ: т.к. в COM-программе все сегментные регистры определяются в момент запуска программы, а не в момент компиляции, то невозможно использование команды вида mov <регистр>, seg <имя сегмента>.

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

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

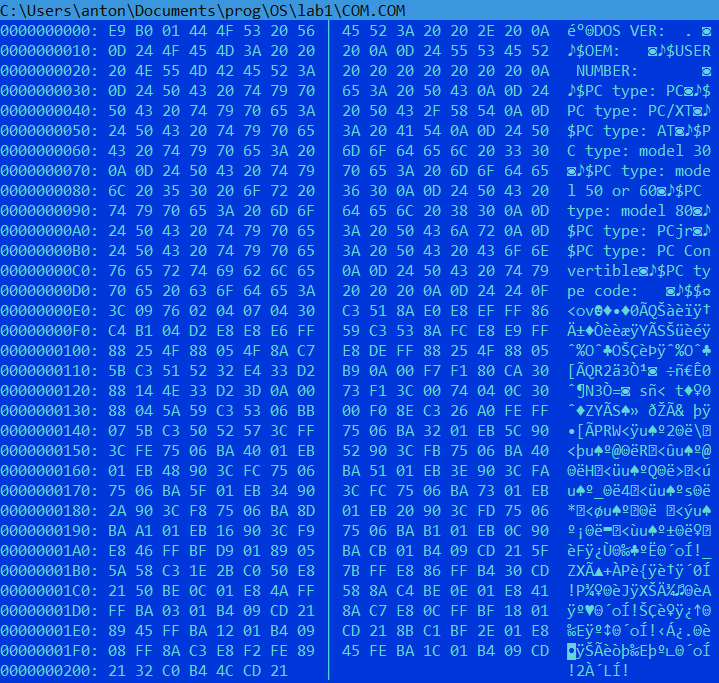
Ответ: COM-файл содержит данные и машинные команды. Код начинается с адреса 0h.

Рисунок 4 – hex-представление .COM файла

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

Ответ: в «плохом» файле EXE данные и код содержатся в одном сегменте. Код располагается с адреса 300h, 200h байт занимает заголовок, и еще на 100h сдвигает команда ORG 100h. С адреса 0h идёт таблица настроек.

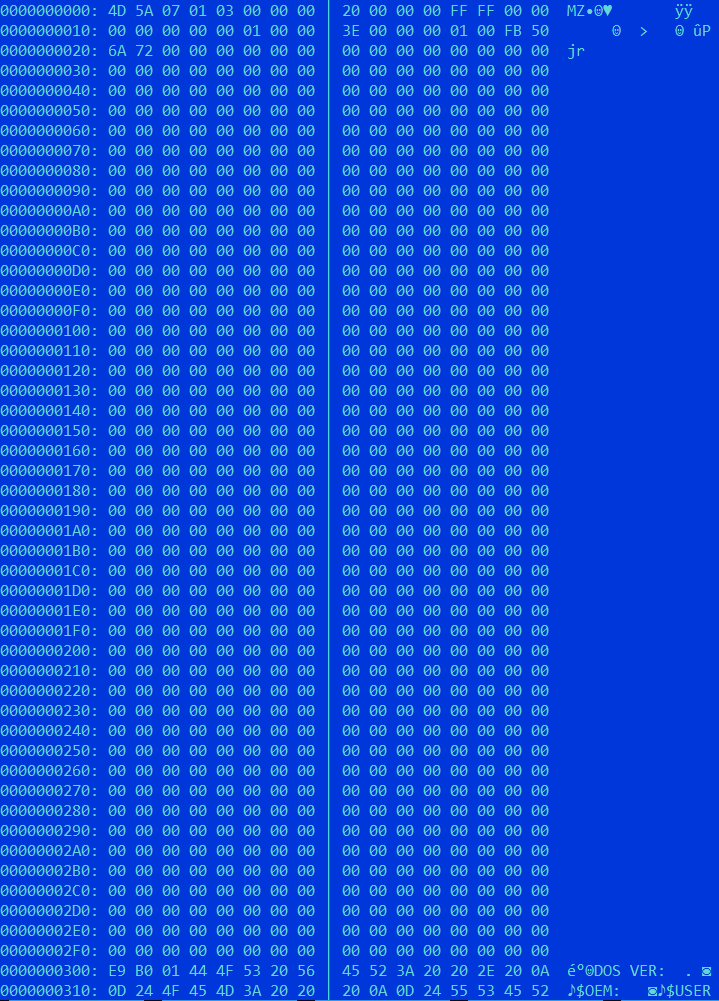
Рисунок 5 – hex-представление «плохого» .ЕХЕ файла

Рисунок 6 – hex-представление «плохого» .ЕХЕ файла(продолжение)

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

Ответ: в «хорошем» код, стек и данные выделены в отдельные сегменты. Код программы начинается также с адреса 300h, только в отличие от «плохого» файла, 100h байтов выделено под стек.

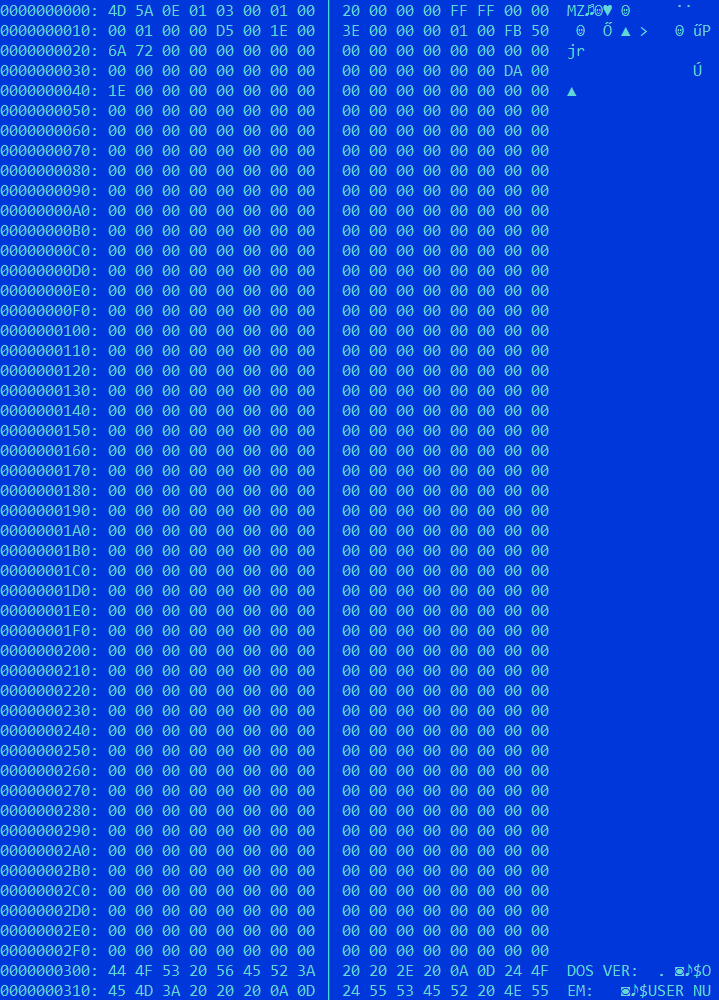
Рисунок 7 – hex-представление «хорошего» .ЕХЕ файла

Рисунок 8 – hex-представление «плохого» .ЕХЕ файла(продолжение)

**Загрузка COM модуля в основную память.**

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

Ответ:

1. Определяется сегментный адрес свободного участка памяти достаточного для размещения программы размера;

2. Создается блок памяти для PSP и программы (сегмент: 0000h – PSP; сегмент: 0100h – программа). В поля PSP заносятся соответствующие значения;

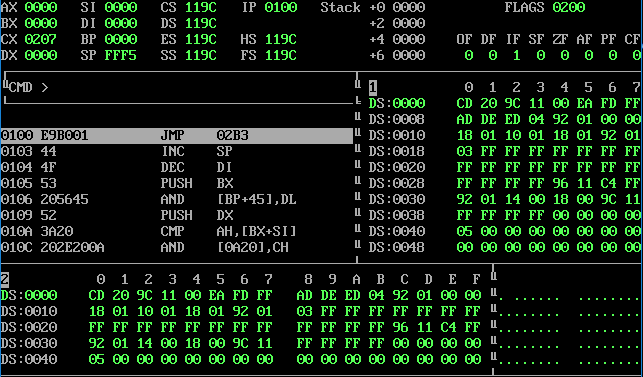
3. Загружается COM-файл с адреса PSP: 0100h;

4. Значение регистра AX устанавливается в соответствии с параметрами командной строки;

5. Регистры DS, ES и SS устанавливаются на сегмент PSP;

6. Регистр SP устанавливается на конец сегмента;

7. Происходит запуск программы с адреса PSP:0100h.

Рисунок 9 – состояние регистров перед запуском COM-программы

1. Что располагается с адреса 0?

Ответ: первые 100h байт занимает PSP.

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

Ответ: все сегментные регистры указывают на начало PSP.

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

Ответ: стек занимает весь сегмент COM-программы, его начало находится в конце сегмента, на него указывает регистр SP = FFF5. SS указывает на начало сегмента.

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

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

Ответ:

1. Определяется сегментный адрес свободного участка памяти, размер которого достаточен для размещения программы;

2. Создается блок памяти для PSP. В поля PSP заносятся соответствующие значения;

3. Определяется начальный сегмент. Загрузочный модуль считывается в начальный сегмент;

4. Таблица настройки считывается в рабочую память;

5. Определяются значения сегментных регистров:

SS = 11AC – сегмент стека;

CS = 11CA – сегмент кода;

DS = 119С – сегмент данных.

1. На что указывают регистры DS и ES?

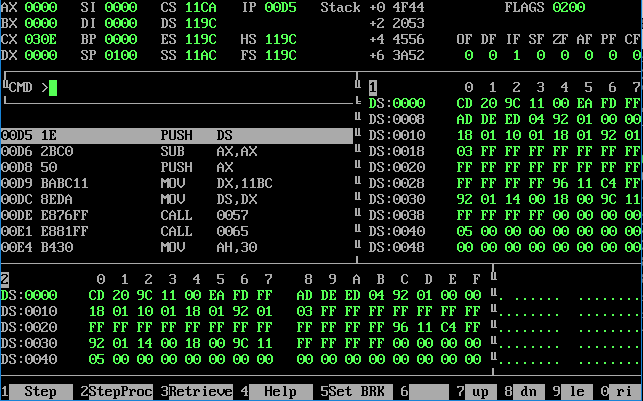
Ответ: регистры указывают на начало PSP.

1. Как определяется стек?

Ответ: при исполнении в регистр SS записывается адрес начала сегмента стека, а в SP – его вершины.

1. Как определяется точка входа?

Ответ: точка входа в программу определяется с помощью директивы END. После этой директивы указывается адрес, куда переходит программа при запуске.

Рисунок 10 - состояние регистров перед запуском EXE-программы

ПРИЛОжение а

исходный код .Сom модуля

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

DOS\_V db 'DOS VER: '

DOS\_F db ' .'

END\_DOS\_V db ' ', 0AH, 0DH,'$'

OEM db 'OEM: '

ENDOEM db ' ', 0AH, 0DH,'$'

USERN db 'USER NUMBER: '

USERNEND db ' ', 0AH, 0DH,'$'

TYPE\_PC db 'PC type: PC', 0AH, 0DH,'$'

TYPE\_PC\_XT db 'PC type: PC/XT', 0AH, 0DH,'$'

TYPE\_AT db 'PC type: AT', 0AH, 0DH,'$'

TYPE\_PS2\_M30 db 'PC type: model 30', 0AH, 0DH,'$'

TYPE\_PS2\_M50\_60 db 'PC type: model 50 or 60', 0AH, 0DH,'$'

TYPE\_PS2\_M80 db 'PC type: model 80', 0AH, 0DH,'$'

TYPE\_PSjr db 'PC type: PCjr', 0AH, 0DH,'$'

TYPE\_PC\_CONV db 'PC type: PC Convertible', 0AH, 0DH,'$'

TYPE\_UNKNOWN db 'PC type code: '

TYPE\_CODE db ' ', 0AH, 0DH, '$'

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 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 10 с/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\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

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

GET\_PC\_CODE PROC near

push BX

push ES

mov BX,0F000H

mov ES,BX

mov AL,ES:[0FFFEH]

pop ES

pop BX

ret

GET\_PC\_CODE ENDP

PRINT\_PC\_TYPE PROC near

push AX

push DX

push DI

T0: cmp AL, 0FFh

jne T1

mov DX, offset TYPE\_PC;

jmp print

T1: cmp AL, 0FEh

jne T2

mov DX, offset TYPE\_PC\_XT;

jmp print

T2: cmp AL, 0FBh

jne T3

mov DX, offset TYPE\_PC\_XT;

jmp print

T3: cmp AL, 0FCh

jne T4

mov DX, offset TYPE\_AT;

jmp print

T4: cmp AL, 0FAh

jne T5

mov DX, offset TYPE\_PS2\_M30;

jmp print

T5: cmp AL, 0FCh

jne T6

mov DX, offset TYPE\_PS2\_M50\_60;

jmp print

T6: cmp AL, 0F8h

jne T7

mov DX, offset TYPE\_PS2\_M80;

jmp print

T7: cmp AL, 0FDh

jne T8

mov DX, offset TYPE\_PSjr

jmp print

T8: cmp AL, 0F9h

jne T9

mov DX, offset TYPE\_PC\_CONV

jmp print

T9:

call BYTE\_TO\_HEX

mov DI, OFFSET TYPE\_CODE

mov [DI], AX

mov DX, offset TYPE\_UNKNOWN;

print:

mov AH, 09h

int 21h

pop DI

pop DX

pop AX

ret

PRINT\_PC\_TYPE ENDP

BEGIN:

push DS

sub AX, AX

push AX

; PC TYPE

call GET\_PC\_CODE

call PRINT\_PC\_TYPE

mov AH,30H

INT 21H

; DOS

push AX

mov SI, offset DOS\_F

call BYTE\_TO\_DEC

pop AX

mov AL, AH

mov SI, offset END\_DOS\_V

call BYTE\_TO\_DEC

mov DX, offset DOS\_V;

mov AH, 09h

int 21h

; OEM

mov AL, BH

call BYTE\_TO\_HEX

mov DI, offset ENDOEM

mov [DI-1], AX

mov DX, offset OEM;

mov AH, 09h

int 21h

; USER NUMBER

mov AX, CX

mov DI, offset USERNEND

call WRD\_TO\_HEX

mov AL, BL

call BYTE\_TO\_HEX

mov [DI-2], AX

mov DX, offset USERN

mov AH, 09h

int 21h

; выход в DOS

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START

ПРИЛОжение Б

исходный код .EXE модуля

SSEG SEGMENT stack

db 100 dup(?)

SSEG ENDS

DATA SEGMENT

DOS\_V db 'DOS VER: '

DOS\_F db ' .'

END\_DOS\_V db ' ', 0AH, 0DH,'$'

OEM db 'OEM: '

ENDOEM db ' ', 0AH, 0DH,'$'

USERN db 'USER NUMBER: '

USERNEND db ' ', 0AH, 0DH,'$'

TYPE\_PC db 'PC type: PC', 0AH, 0DH,'$'

TYPE\_PC\_XT db 'PC type: PC/XT', 0AH, 0DH,'$'

TYPE\_AT db 'PC type: AT', 0AH, 0DH,'$'

TYPE\_PS2\_M30 db 'PC type: model 30', 0AH, 0DH,'$'

TYPE\_PS2\_M50\_60 db 'PC type: model 50 or 60', 0AH, 0DH,'$'

TYPE\_PS2\_M80 db 'PC type: model 80', 0AH, 0DH,'$'

TYPE\_PSjr db 'PC type: PCjr', 0AH, 0DH,'$'

TYPE\_PC\_CONV db 'PC type: PC Convertible', 0AH, 0DH,'$'

TYPE\_UNKNOWN db 'PC type code: '

TYPE\_CODE db ' ', 0AH, 0DH, '$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:CODE, ES:DATA, SS:SSEG

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 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 10 с/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\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

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

GET\_PC\_CODE PROC near

push BX

push ES

mov BX,0F000H

mov ES,BX

mov AL,ES:[0FFFEH]

pop ES

pop BX

ret

GET\_PC\_CODE ENDP

PRINT\_PC\_TYPE PROC near

push AX

push DX

push DI

T0: cmp AL, 0FFh

jne T1

mov DX, offset TYPE\_PC;

jmp print

T1: cmp AL, 0FEh

jne T2

mov DX, offset TYPE\_PC\_XT;

jmp print

T2: cmp AL, 0FBh

jne T3

mov DX, offset TYPE\_PC\_XT;

jmp print

T3: cmp AL, 0FCh

jne T4

mov DX, offset TYPE\_AT;

jmp print

T4: cmp AL, 0FAh

jne T5

mov DX, offset TYPE\_PS2\_M30;

jmp print

T5: cmp AL, 0FCh

jne T6

mov DX, offset TYPE\_PS2\_M50\_60;

jmp print

T6: cmp AL, 0F8h

jne T7

mov DX, offset TYPE\_PS2\_M80;

jmp print

T7: cmp AL, 0FDh

jne T8

mov DX, offset TYPE\_PSjr

jmp print

T8: cmp AL, 0F9h

jne T9

mov DX, offset TYPE\_PC\_CONV

jmp print

T9:

call BYTE\_TO\_HEX

mov DI, OFFSET TYPE\_CODE

mov [DI], AX

mov DX, offset TYPE\_UNKNOWN;

print:

mov AH, 09h

int 21h

pop DI

pop DX

pop AX

ret

PRINT\_PC\_TYPE ENDP

START:

push DS

sub AX, AX

push AX

mov DX, DATA

mov DS, DX

; PC TYPE

call GET\_PC\_CODE

call PRINT\_PC\_TYPE

mov AH,30H

INT 21H

; DOS

push AX

mov SI, offset DOS\_F

call BYTE\_TO\_DEC

pop AX

mov AL, AH

mov SI, offset END\_DOS\_V

call BYTE\_TO\_DEC

mov DX, offset DOS\_V;

mov AH, 09h

int 21h

; OEM

mov AL, BH

call BYTE\_TO\_HEX

mov DI, offset ENDOEM

mov [DI-1], AX

mov DX, offset OEM;

mov AH, 09h

int 21h

; USER NUMBER

mov AX, CX

mov DI, offset USERNEND

call WRD\_TO\_HEX

mov AL, BL

call BYTE\_TO\_HEX

mov [DI-2], AX

mov DX, offset USERN

mov AH, 09h

int 21h

; выход в DOS

xor AL,AL

mov AH,4Ch

int 21H

CODE ENDS

END START