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 загрузочных модулей

Студентка гр. 6383 _____ Михеева Е. Е.

Преподаватель _____ Губкин А.Ф.

Санкт-Петербург

2018

ПОСТАНОВКА ЗАДАЧИ

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

Интерфейс использованных функций управляющей программы.

Тип IBM PC хранится в байте по адресу 0E000:0EEEE11, 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
PSjr:	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

Описание программы

Программа выводит тип IBM PC, версию MS DOS, серийный номер OEM, 24-битовый серийный номер пользователя.

1. Таблица используемых процедур

Процедура	**Описание**
TETR_TO_HEX	Перевод десятичной цифры в код символа
BYTE_TO_HEX	Перевод байта в 16-ной с/с в символьный код
WRD_TO_HEX	Перевод слова в 16-ной с/с в символьный код
BYTE_TO_DEC	Перевод байта в 16-ной с/с в символьный код в 10-ной с/с
PRINT	Вывод строки на экран

PCTYPE	Определение кода типа PC

2. Таблица используемых структур

Структура	**Тип**	**Описание**
isPC	Строка, содержащая символы размером 1 байт	Данная строка выводится, в случае, если тип IBM PC PC
isPC_XT	Строка, содержащая символы размером 1 байт	Данная строка выводится, в случае, если тип IBM PC PC/XT
isPS2_30	Строка, содержащая символы размером 1 байт	Данная строка выводится, в случае, если тип IBM PC PS2
isPS2_80	Строка, содержащая символы размером 1 байт	Данная строка выводится, в случае, если тип IBM PC PS2_80
isPCjr	Строка, содержащая символы размером 1 байт	Данная строка выводится, в случае, если тип IBM PCjr
isPC_conv	Строка, содержащая символы размером 1 байт	Данная строка выводится, в случае, если тип IBM PC PC Convertible
TYPE_PC	Строка, содержащая символы размером 1 байт	Данная строка выводится, в случае, если нам неизвестен тип IBM PC. Следом за ней выводится код IBM PC в 16-ной с/с
ENDLINE	Строка, содержащая символы размером 1 байт	Содержит символы переноса и конца строки для корректного вывода строки TYPE_PC и кода IBM PC
DOS_V	Строка, содержащая символы размером 1 байт	Данная строка содержит номер версии DOS в 10-ной с/с
ENDL_DOS_V	Строка, содержащая символы размером 1 байт	Содержит символы переноса и конца строки для корректного вывода строки DOS_V и номера версии DOS

MOD_N	Строка, содержащая однобайтовые символы	Данная строка содержит номер модификации DOS в 10-ной с/с
END_MOD_N	Строка, содержащая символы размером 1 байт	Содержит символы переноса и конца строки для корректного вывода строки MOD_N и номера модификации DOS
OEM	Строка, содержащая символы размером 1 байт	Данная строка содержит серийный номер OEM в 16-ной с/с
ENDOEM	Строка, содержащая символы размером 1 байт	Содержит символы переноса и конца строки для корректного вывода строки OEM и серийного номера OEM
USERN	Строка, содержащая символы размером 1 байт	Данная строка содержит серийный номер пользователя в 16-ной с/с
USERNEND	Строка, содержащая символы размером 1 байт	Содержит символы переноса и конца строки для корректного вывода строки USERN и серийного номера OEM

3. Вывод программы:

Рис 1. Вывод программы.

Исследование .COM, хорошего и плохого .EXE файлов

1) Отличия исходных текстов COM и EXE программ
1.1) Сколько сегментов должна содержать COM-программа?
Ответ. COM-программа должна содержать 1 сегмент

1.2) EXE-программа?

Ответ. EXE-программа должна содержать 1 и более сегментов,но обычно выделяют сегменты под код, данные и стек

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

Ответ. assume - задает значения сегментных регистров при входе в программу, org - служит для резервирования заданного количества памяти от начального адреса под psp,segment - определяет начало и конец сегмента памяти

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

Ответ. Не все, нельзя использовать команды, использующие адреса сегмента. Например, mov ax,code. Это связано с тем, что для этих команд используются данные о сегментах, находящиеся в таблице релокации (relocation table). COM-модули, в отличии от .EXE модулей, такую таблицу не содержат. Relocation table содержит список перемещений, т.е. тех мест в образе файла в памяти, в которых необходимо учесть различие между указанным в заголовке базовым адресом загрузки и реальным адресом загрузки.

2) Отличия форматов файлов COM и EXE модулей

2.1) Какова структура файла COM? С какого адреса располагается код

Ответ. В .COM файле стек сгенерирован автоматически, место под psp выделено программистом с помощью директивы ORG. Код располагается с 0 байта

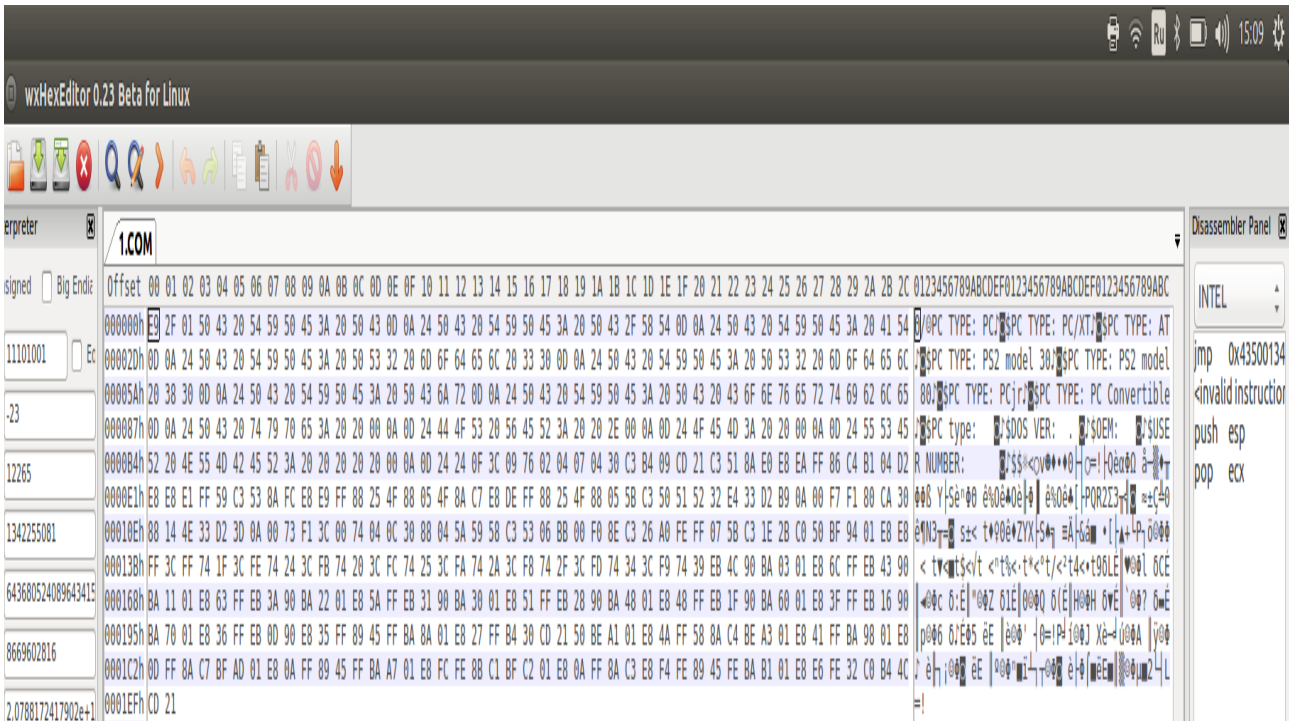

Рис. 2. COM-модуль

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

Ответ. Код начинается с 300h байта.

С 0 байта расположены mz — который идентифицирует принадлежность файла данному формату, а также таблица размещения сегментов

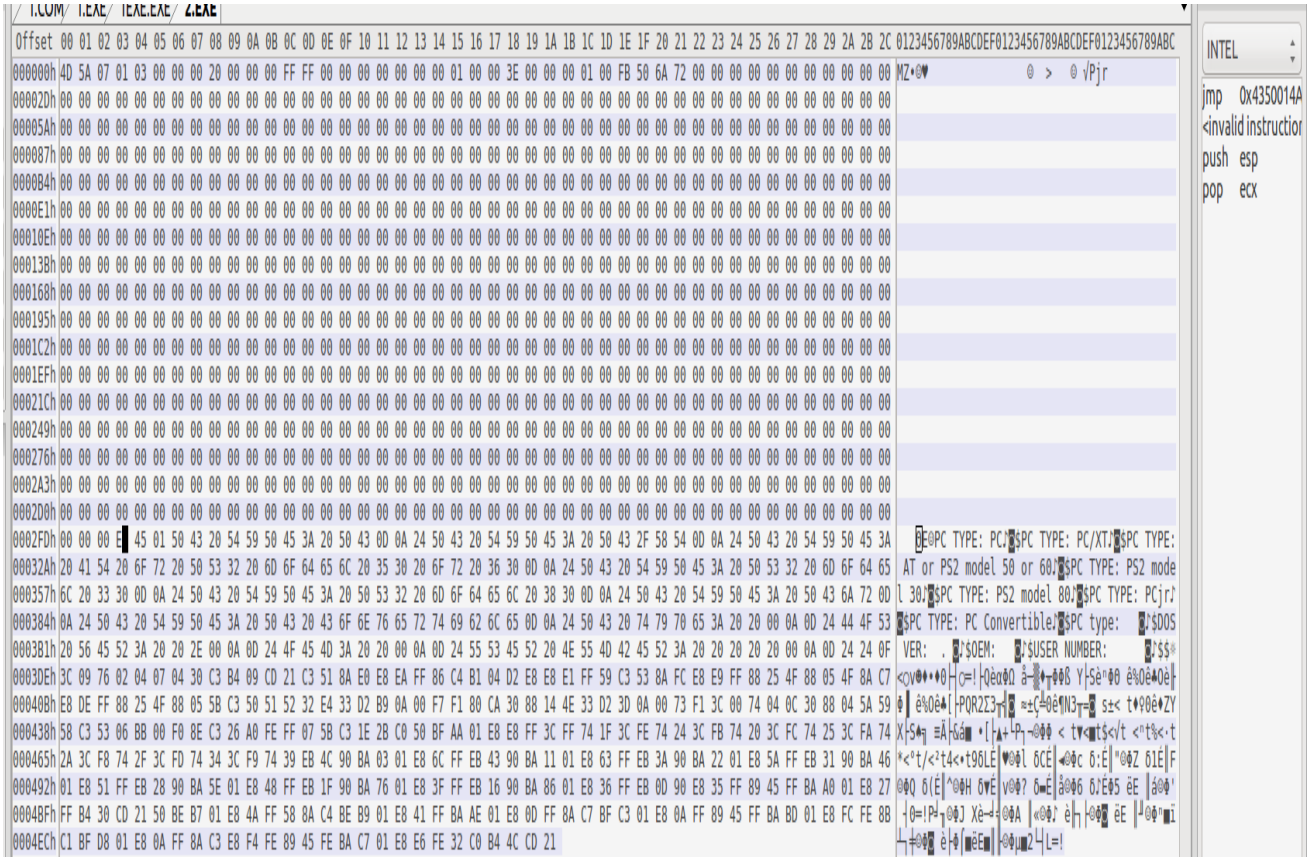

Рис. 3. «Плохой» .EXE-модуль

2.3) Какова структура файла «хорошего» EXE? Чем он отличается от файла «ПЛОХОГО» EXE?

Ответ. В «хорошем» .EXE файле выделены 3 сегмента: сегменты кода, данных, стека.

В «плохом» .EXE файле 1 сегмент

3) Загрузка COM модуля в основную память

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

Ответ. Код располагается С 256(100h) байта

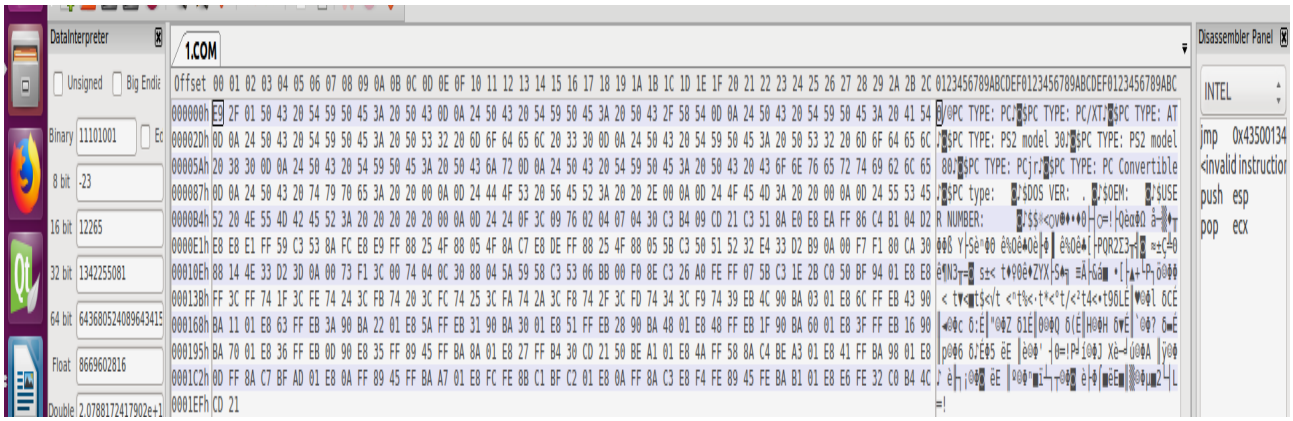

Рис. 4. COM-модуль

3.2) Что располагается с адреса 0?
Ответ. С 0 байта располагается PSP
3.3) Какие значения имеют сегментные регистры? На какие области памяти они
указывают?
Ответ. ds = cs = es = ss = 48DD — указывают на один сегмент PSP.
3.4) Как определяется стек? Какую область памяти он занимает? Какие адреса?
Ответ. Стек располакается в сегменте codeseg вершина стека находится по адресу fffe, занимает область памяти 48DD, адреса от FFFE до 0000

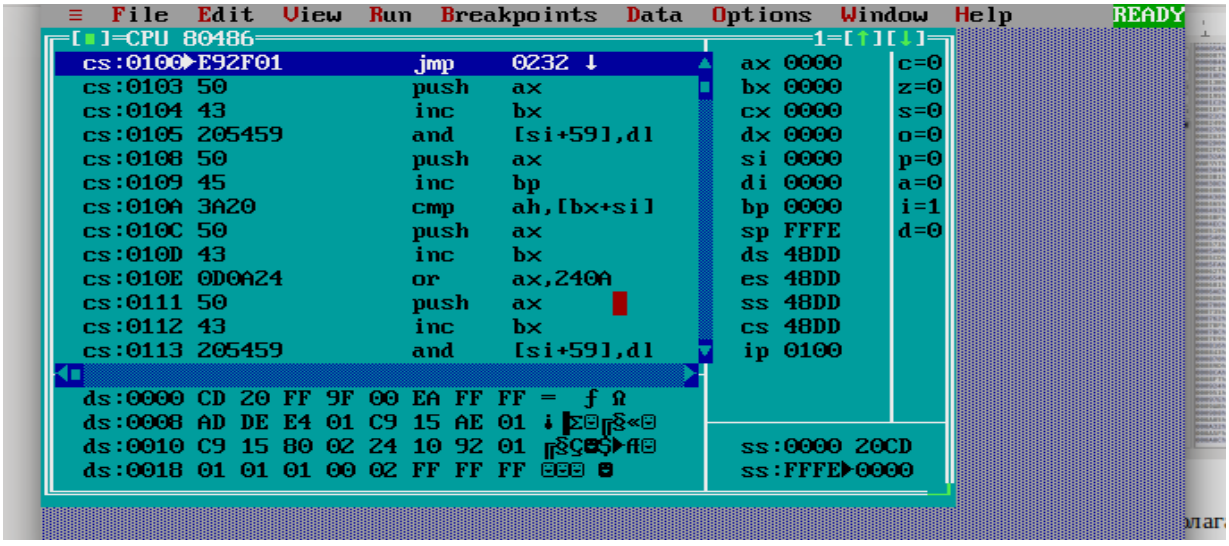

Рис. 5. COM-модуль

4) Загрузка «хорошего» EXE модуля в основную память
4.1) Как загружается «хороший» EXE? Какие значения имеют сегментные регистры?

Ответ. При загрузке регистры ds, ss, cs указывают на разные сегменты, программа начинается с адреса 006C
ds = es = 48DD, ss = 48ED? Cs = 48FD

4.2) На что указывают регистры ds и es?

Ответ. Регистры ds и es указывают на сегмент PSP.

4.3) Как определяется стек?

Ответ. Стек располагается в сегменте astack, вершина стека находится по адресу 0018, занимает область памяти 48ED, адреса от 0018 до 0000

4.4) Как определяется точка входа?

Ответ. Точка входа определяется по end main, адрес ip = 00C6

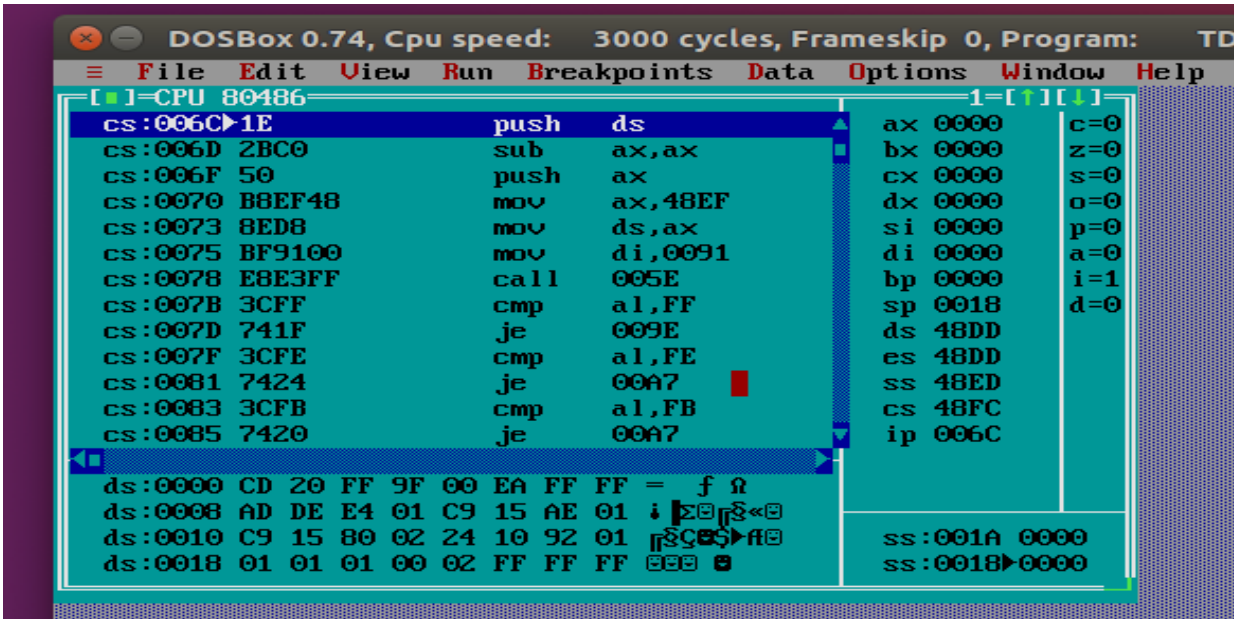

Рис. 6. «Хороший» .EXE-модуль

Выводы. В ходе работы была написана программа для определения типа IBM PC, версии DOS, серийного номера OEM, серийного номера пользователя. Также исследованы отличия .COM файлов от «плохого» и «хорошего» .EXE файлов.

Приложение А. Тексты программ

1. Текст программы .COM файла и «плохого» .EXE файла

```
CODESEG SEGMENT

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

ORG 100H

START: JMP BEGIN


isPC db 'PC TYPE: PC', 0dh, 0ah, '$';
isPC_XT db 'PC TYPE: PC/XT', 0dh, 0ah, '$';
isAT db 'PC TYPE: AT ', 0dh, 0ah, '$';
isPS2_30 db 'PC TYPE: PS2 model 30', 0dh, 0ah, '$';
isPS2_80 db 'PC TYPE: PS2 model 80', 0dh, 0ah, '$';
isPCjr db 'PC TYPE: PCjr', 0dh, 0ah, '$';
isPC_conv db 'PC TYPE: PC Convertible', 0dh, 0ah, '$';


TYPE_PC   db   'PC type:  '
ENDLINE DB 0, 0AH, 0DH,'$'

DOS_V     db   'DOS VER: '
ENDL_DOS_V DB 0, 0AH, 0DH,'$'

MOD_N     db   ' .'
END_MOD_N DB 0, 0AH, 0DH,'$'

OEM  db   'OEM:  '
ENDOEM DB 0, 0AH, 0DH,'$'

USERN     db   'USER NUMBER:     '
USERNEND DB 0, 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
```

```
;-------------------------------

PRINT PROC near

 mov  ah,9
 int  21h
 ret

PRINT 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 ;в AL старшая цифра

pop CX ;в AH младшая

ret

BYTE_TO_HEX ENDP

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

WRD_TO_HEX PROC near

;перевод в 16 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 байта в 10с/с, SI - адрес поля младшей цифры
; AL содержит исходный байт
push AX
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
pop  AX
ret
BYTE_TO_DEC ENDP
;-------------------------------
; TYPE PC
PCTYPE    PROC NEAR
push BX
push ES

mov  BX,0F000H
mov  ES,BX
mov  AL,ES:[0FFFEH]

pop  ES
pop  BX

ret
PCTYPE    ENDP


;;;;;;;;;;BEGIN
BEGIN:
```

```
push DS
sub AX,AX
push AX


; pc type
mov di, offset ENDLINE
call PCTYPE

; try to define pc type
 cmp al, 0ffh
        je mPC;

    cmp al, 0feh
        je mPC_XT;

    cmp al, 0fbh
        je mPC_XT;

    cmp al, 0fch
        je mAT;

    cmp al, 0fah
        je mPS2_30;


    cmp al, 0f8h
        je mPS2_80;

    cmp al, 0fdh
        je mPCjr;

    cmp al, 0f9h
        je mPC_conv;

    jmp mVMSD;

;---------if type pc defined----------
mPC:    lea dx, isPC;
        call PRINT
        jmp mVMSD;

mPC_XT: lea dx, isPC_XT;
        call PRINT
        jmp mVMSD;

mAT:    lea dx, isAT;
         call PRINT
        jmp mVMSD;
```

```
mPS2_30:    lea dx, isPS2_30;
             call PRINT
            jmp mVMSD;

mPS2_80:    lea dx, isPS2_80;
            call PRINT
            jmp mVMSD;

mPCjr:  lea dx, isPCjr;
         call PRINT
        jmp mVMSD;

mPC_conv:   lea dx, isPC_conv;
             call PRINT
            jmp mVMSD;

;if pc type undefined

call BYTE_TO_HEX
mov [di-1],ax
;mov [PH],ax
mov dx, offset TYPE_PC;
call PRINT
mVMSD:

;;;;DOS TYPE
mov  AH,30H
INT  21H


push ax
mov si, offset MOD_N
call BYTE_TO_DEC
pop ax
mov al,ah
mov si, offset END_MOD_N
call BYTE_TO_DEC
mov dx, offset  DOS_V;
call PRINT

;;;;oem

mov al,bh
mov di, offset ENDOEM
call BYTE_TO_HEX
mov [di-1],ax
mov dx, offset OEM;
call PRINT

;; 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;
call PRINT


;end of program
xor AL,AL

mov AH,4Ch

int 21H

CODESEG ENDS
END START
```

2. Текст программы «хорошего» .EXE файла

```
AStack    SEGMENT  STACK
          DW 12 DUP(?)    ;
AStack    ENDS


DATA      SEGMENT
isPC db 'PC TYPE: PC', 0dh, 0ah, '$';
isPC_XT db 'PC TYPE: PC/XT', 0dh, 0ah, '$';
isAT db 'PC TYPE: AT ', 0dh, 0ah, '$';
isPS2_30 db 'PC TYPE: PS2 model 30', 0dh, 0ah, '$';
isPS2_80 db 'PC TYPE: PS2 model 80', 0dh, 0ah, '$';
isPCjr db 'PC TYPE: PCjr', 0dh, 0ah, '$';
isPC_conv db 'PC TYPE: PC Convertible', 0dh, 0ah, '$';


TYPE_PC   db   'PC type:  '
ENDLINE DB 0, 0AH, 0DH,'$'
;label PH db at TYPE_PC

DOS_V     db   'DOS VER: '
;ENDL_DOS_V DB 0, 0AH, 0DH,'$'

MOD_N     db   ' .'
END_MOD_N DB 0, 0AH, 0DH,'$'

OEM  db   'OEM:  '
ENDOEM DB 0, 0AH, 0DH,'$'

USERN     db   'USER NUMBER:     '
```

```
USERNEND DB 0, 0AH, 0DH,'$'
DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:CODE, ES:NOTHING, 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

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

PRINT PROC near

 mov  ah,9
 int  21h
 ret

PRINT 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 ;в AL старшая цифра

pop CX ;в AH младшая
```

```
ret
BYTE_TO_HEX ENDP
;-------------------------------
WRD_TO_HEX PROC near
;перевод в 16 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 байта в 10с/с, SI - адрес поля младшей цифры
; AL содержит исходный байт
```

```
push AX
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
pop  AX
ret
BYTE_TO_DEC ENDP
;--------------------------------
; TYPE PC
PCTYPE    PROC NEAR
push BX
```

```
push ES


mov  BX,0F000H

mov  ES,BX

mov  AL,ES:[0FFFEH]


pop  ES

pop  BX


ret
PCTYPE    ENDP



;;;;;;;;;;;BEGIN


MAIN PROC FAR

push DS
sub AX,AX
push AX
mov   AX,DATA
mov   DS,AX

; pc type
mov di, offset ENDLINE
call PCTYPE

; try to define pc type
 cmp al, 0ffh
        je mPC;

    cmp al, 0feh
        je mPC_XT;

    cmp al, 0fbh
        je mPC_XT;

    cmp al, 0fch
        je mAT;

    cmp al, 0fah
        je mPS2_30;
```

```
    cmp al, 0f8h
        je mPS2_80;

    cmp al, 0fdh
        je mPCjr;

    cmp al, 0f9h
        je mPC_conv;

    jmp mVMSD;

;---------if type pc defined----------
mPC:    lea dx, isPC;
        call PRINT
        jmp mVMSD;

mPC_XT: lea dx, isPC_XT;
        call PRINT
        jmp mVMSD;

mAT:    lea dx, isAT;
         call PRINT
        jmp mVMSD;

mPS2_30:    lea dx, isPS2_30;
             call PRINT
            jmp mVMSD;

mPS2_80:    lea dx, isPS2_80;
            call PRINT
            jmp mVMSD;

mPCjr:  lea dx, isPCjr;
         call PRINT
        jmp mVMSD;

mPC_conv:   lea dx, isPC_conv;
             call PRINT
            jmp mVMSD;

;if pc type undefined

call BYTE_TO_HEX
mov [di-1],ax
;mov [PH],ax
mov dx, offset TYPE_PC;
call PRINT
mVMSD:

;;;;DOS TYPE
mov  AH,30H
INT  21H
```

```
push ax
mov si, offset MOD_N
call BYTE_TO_DEC
pop ax
mov al,ah
mov si, offset END_MOD_N
call BYTE_TO_DEC
mov dx, offset  DOS_V;
call PRINT

;;;;oem

mov al,bh
mov di, offset ENDOEM
call BYTE_TO_HEX
mov [di-1],ax
mov dx, offset OEM;
call PRINT

;; 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;
call PRINT


;end of program
xor AL,AL

mov AH,4Ch

int 21H
MAIN ENDP

CODE ENDS
END MAIN
```