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 заголовочных модулей

Студентка гр. 9381 _____ Москаленко Е.М.

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 загрузки их в основную память.

Функции и структуры данных.

Названия функций	Описание
TETR_TO_HEX	Перевод четырех младших битов регистре AL в 16-ричную цифру.
BYTE_TO_HEX	Перевод байта из AL в число 16-ной с.с. Символы записываются в регистры AL и AH.
WRD_TO_HEX	Перевод слова из AH в число в 16-ной с.с. Записывается в виде 4 символов по адресу из DI.
BYTE_TO_DEC	Перевод байта из AL в 16-ной с.с в число в 10-ной с.с.. Записывается по адресу, на который указывает SI (младшая цифра).
PRINT	Вывод строки на экран при помощи функции 9h прерывания 21h.
CHECK_PC	Процедура определения типа PC. Если тип не определен, то выводит строку с 16-ричной записью байта.

Выполнение работы.

1. Был написан текст исходного .COM модуля **os1_com.asm**, определяющий тип PC и версию системы. Программа выводит информацию о типе PC, версии системы, а также строки с серийным номером OEM и серийным номером пользователя. Полученный модуль был отлажен. В результате получен «хороший» .COM модуль и «плохой» .EXE модуль.

2. Далее был написан текст исходного модуля .EXE **os1_asm.asm** с теми же функциями, что и в **os1_com.asm.** Модуль был построен и отлажен. В результате получен «хороший» .EXE.
3. Далее было проведено сравнения исходных текстов для .COM и .EXE.
4. Файлы os1_com.com, os1_com.exe и os1_exe.exe были открыты в 16-ричном редакторе. Проведено их сравнение.
5. Загрузочный модуль .COM был исследован при помощи отладчика.
6. Модуль os1_exe.exe так же был исследован при помощи отладчика.

Последовательность действий.

Сначала вызывается процедура CHECK_PC, которая читает содержимое последнего байта ROM BIOS, сравнивает коды, определяет тип PC и выводит строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводится в символьную строку с помощью процедур BYTE_TO_HEX и TETR_TO_HEX и выводится на экран в виде соответствующего сообщения.

Затем при помощи функции 30h прерывания 21h получаем значение текущего номера DOS. В соответствующие строки записываем номер версии системы, серийный номер OEM и серийный номер пользователя.

Для вывода строк используется процедура PRINT, вызывающая функцию 9h прерывания 21h.

Результаты исследования проблем.

Отличия исходных текстов COM и EXE программ.

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

   COM-программы содержат единственный сегмент.

2) ***EXE-программа?***

   EXE-программы содержат несколько программных сегментов, включая сегмент кода, данных и стека.

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

Директива SEGMENT определяет начало сегмента.

Директива **ORG 100h** устанавливает значение программного счетчика в 100h, так как при загрузке COM-файла в память DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы только после этого блока. Все программы, которые компилируются в файлы типа COM, должны начинаться с этой директивы.

Также необходима директива **ASSUME** для того, чтобы сегмент данных и сегмент кода указывали на один общий сегмент.

Директива **END** необходима для завершения любой программы.

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

В COM-программе отсутствует таблица настроек, поэтому нельзя использовать команды вида mov <регистр>, seg <имя сегмента>.

Отличия форматов файлов COM и EXE модулей.

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

COM-файл состоит из одного сегмента, включающий в себя сегменты данных и кода. Код начинается с адреса 0h.

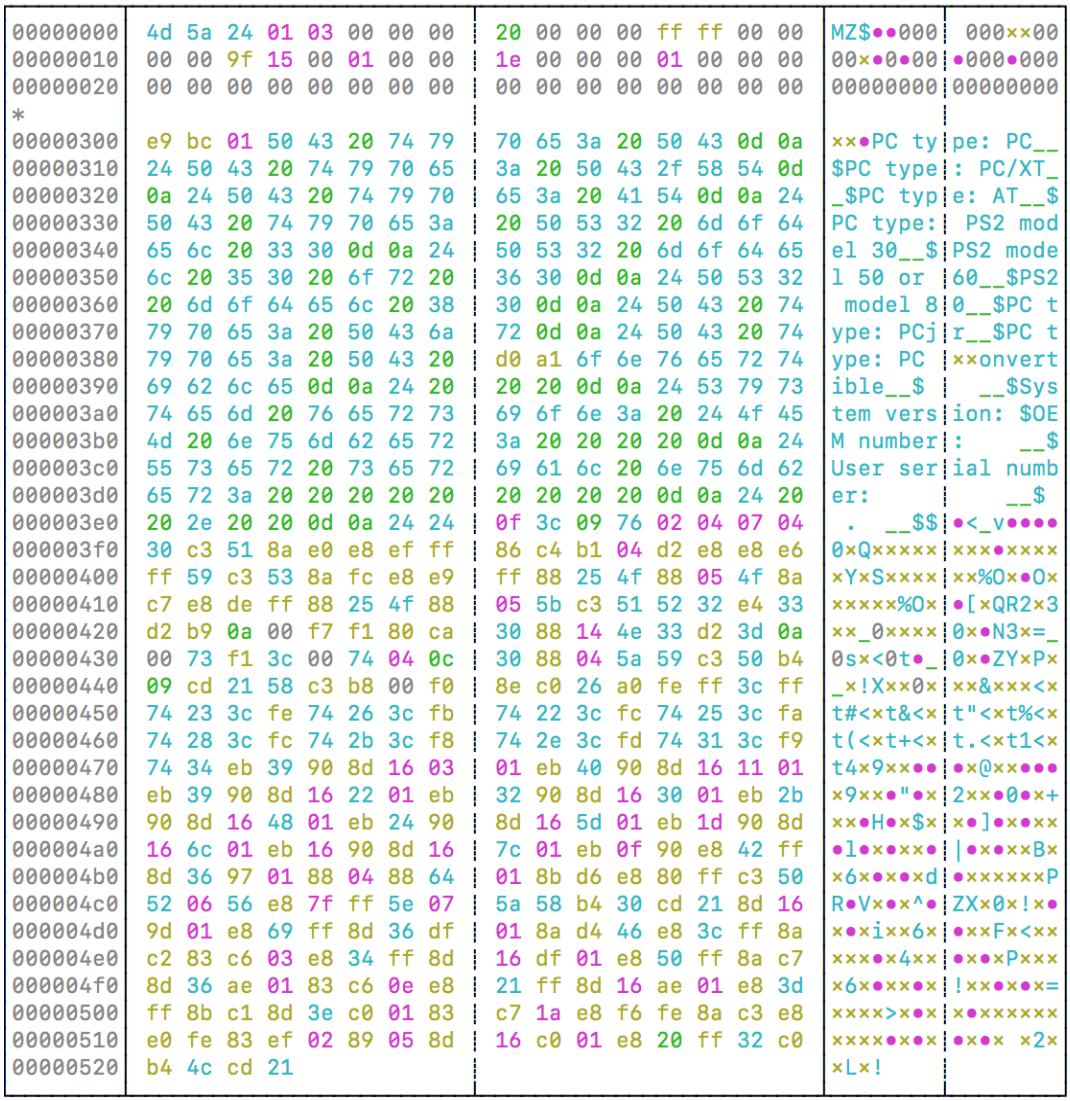
Рисунок 1. 16-ричный вид файла os1_com.asm

```
00000000  e9 bc 01 50 43 20 74 79  70 65 3a 20 50 43 0d 0a  ××•PC ty pe: PC__
00000010  24 50 43 20 74 79 70 65  3a 20 50 43 2f 58 54 0d  $PC type : PC/XT_
00000020  0a 24 50 43 20 74 79 70  65 3a 20 41 54 0d 0a 24  _$PC typ e: AT__$
00000030  50 43 20 74 79 70 65 3a  20 50 53 32 20 6d 6f 64  PC type:  PS2 mod
00000040  65 6c 20 33 30 0d 0a 24  50 53 32 20 6d 6f 64 65  el 30__$ PS2 mode
00000050  6c 20 35 30 20 6f 72 20  36 30 0d 0a 24 50 53 32  l 50 or  60__$PS2
00000060  20 6d 6f 64 65 6c 20 38  30 0d 0a 24 50 43 20 74   model 8 0__$PC t
00000070  79 70 65 3a 20 50 43 6a  72 0d 0a 24 50 43 20 74  ype: PCj r__$PC t
00000080  79 70 65 3a 20 50 43 20  d0 a1 6f 6e 76 65 72 74  ype: PC  ××onvert
00000090  69 62 6c 65 0d 0a 24 20  20 20 0d 0a 24 53 79 73  ible__$    __$Sys
000000a0  74 65 6d 20 76 65 72 73  69 6f 6e 3a 20 24 4f 45  tem vers ion: $OE
000000b0  4d 20 6e 75 6d 62 65 72  3a 20 20 20 20 0d 0a 24  M number :     __$
000000c0  55 73 65 72 20 73 65 72  69 61 6c 20 6e 75 6d 62  User ser ial numb
000000d0  65 72 3a 20 20 20 20 20  20 20 20 20 0d 0a 24 20  er:      __$
000000e0  20 2e 20 20 0d 0a 24 24  0f 3c 09 76 02 04 07 04   .  __$$ •<_v••••
000000f0  30 c3 51 8a e0 e8 ef ff  86 c4 b1 04 d2 e8 e8 e6  0×Q××××× ×××•××××
00000100  ff 59 c3 53 8a fc e8 e9  ff 88 25 4f 88 05 4f 8a  ×Y×S×××× ××%O×•O×
00000110  c7 e8 de ff 88 25 4f 88  05 5b c3 51 52 32 e4 33  ××××%O× •[×QR2×3
00000120  d2 b9 0a 00 f7 f1 80 ca  30 88 14 4e 33 d2 3d 0a  ××_0×××× 0×•N3×=_
00000130  00 73 f1 3c 00 74 04 0c  30 88 04 5a 59 c3 50 b4  0s×<0t•_ 0×•ZY×P×
00000140  09 cd 21 58 c3 b8 00 f0  8e c0 26 a0 fe ff 3c ff  _×!X××0× ××&×××<×
00000150  74 23 3c fe 74 26 3c fb  74 22 3c fc 74 25 3c fa  t#<×t&<× t"<×t%<×
00000160  74 28 3c fc 74 2b 3c f8  74 2e 3c fd 74 31 3c f9  t(<×t+<× t.<×t1<×
00000170  74 34 eb 39 90 8d 16 03  01 eb 40 90 8d 16 11 01  t4×9×••• •×@××•••
00000180  eb 39 90 8d 16 22 01 eb  32 90 8d 16 30 01 eb 2b  ×9×××"•× 2××•0•×+
00000190  90 8d 16 48 01 eb 24 90  8d 16 5d 01 eb 1d 90 8d  ××•H•×$× ××•]•×•××
000001a0  16 6c 01 eb 16 90 8d 16  7c 01 eb 0f 90 e8 42 ff  •l•×•××• |•×•××B×
000001b0  8d 36 97 01 88 04 88 64  01 8b d6 e8 80 ff c3 50  ×6×•×•×d •××××××P
000001c0  52 06 56 e8 7f ff 5e 07  5a 58 b4 30 cd 21 8d 16  R•V×••×^• ZX×0×!××
000001d0  9d 01 e8 69 ff 8d 36 df  01 8a d4 46 e8 3c ff 8a  ×••i××6× •××F×<××
000001e0  c2 83 c6 03 e8 34 ff 8d  16 df 01 e8 50 ff 8a c7  ×××•×4×× •×•×P×××
000001f0  8d 36 ae 01 83 c6 0e e8  21 ff 8d 16 ae 01 e8 3d  ×6×•××•× !××•×•×=
00000200  ff 8b c1 8d 3e c0 01 83  c7 1a e8 f6 fe 8a c3 e8  ××××>×•× ×•××××××
00000210  e0 fe 83 ef 02 89 05 8d  16 c0 01 e8 20 ff 32 c0  ××××•×•× •×•× ×2×
00000220  b4 4c cd 21                                      ×L×!
```

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

Код располагается с адреса 300h, как и данные. Это неправильно для модуля .EXE. С адреса 0h располгается заголовок и таблица настройки адресов, состоящая из длинных указателей (смещение: сегмент) на те слова в загрузочном модуле, которые содержат настраиваемые сегментные адреса.

Рисунок 2. 16-ричный вид «плохого» модуля .EXE os1_com.exe

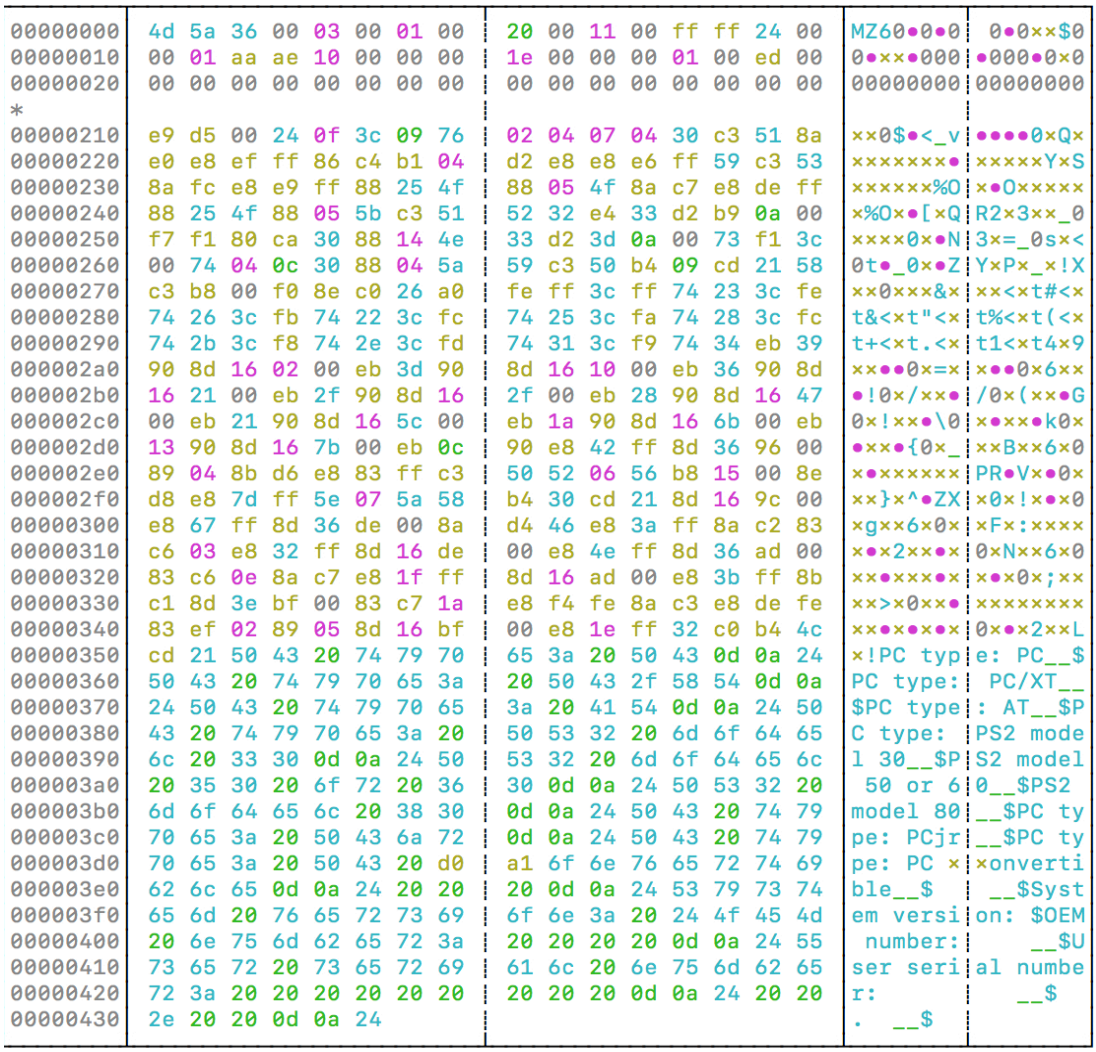


3) Какова структура файла «хорошего» EXE? Чем он отличается от файла «плохого» EXE?

«Хороший» .EXE модуль содержит несколько программных сегментов, включая сегмент кода, данных и стека, тогда как «плохой» содержит лишь один.

Также хороший .EXE в отличие от плохого не содержит директивы ORG 100h, выделяющей память под PSP (256 байт), поэтому код начинается с адреса 210h. Хороший .EXE модуль в начале содержит таблицу настроек.

Рисунок 3. 16-ричный вид «хорошего» модуля .EXE os1_exe.exe

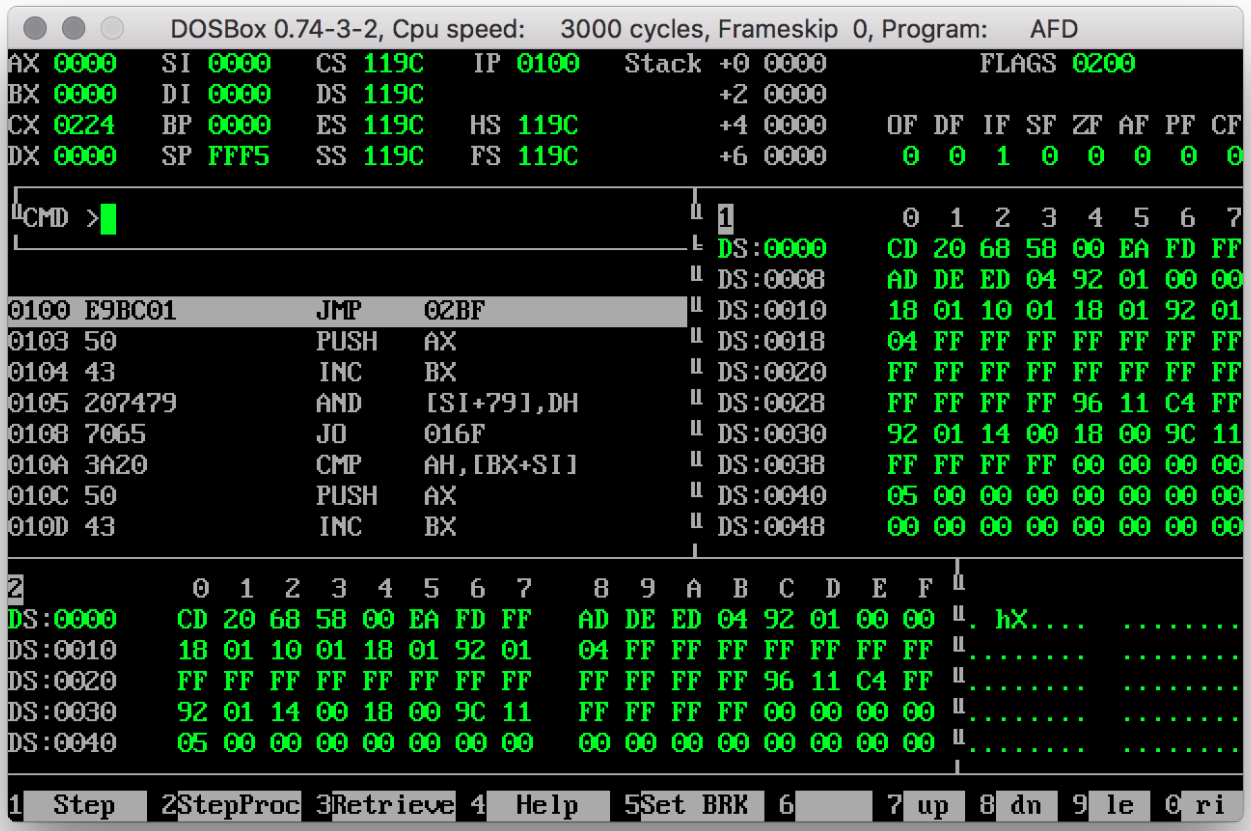


Загрузка COM модуля в основную память.

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

Образ COM-файла считывается с диска и помещается в память, начиная с PSP:0100h. Код располагается с адреса 100h.

Рисунок 4. Модуль os1_com.com в отладчике AFD.



2) Что располагается с адреса 0?

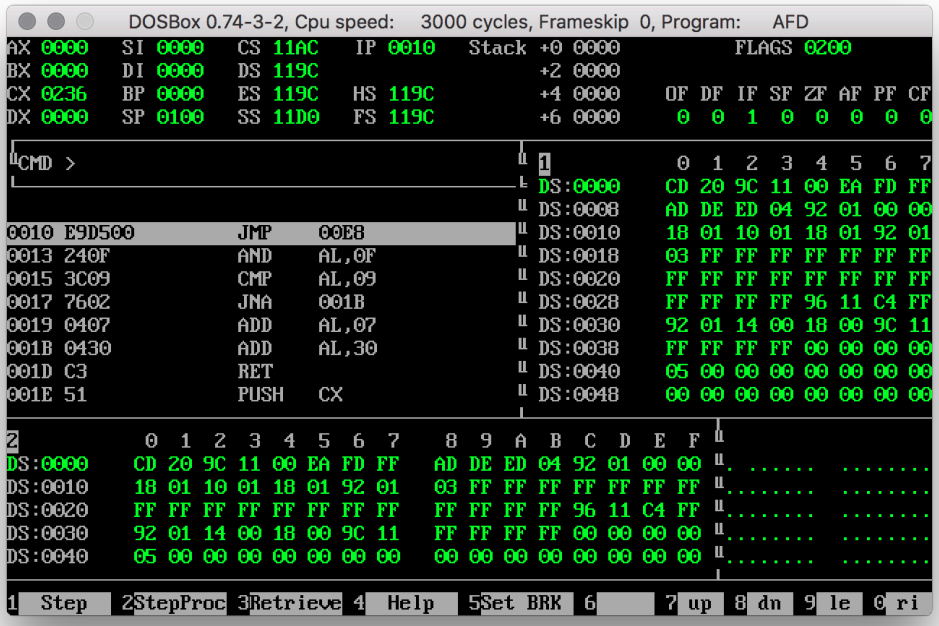
PSP (Program Segment Prefics) – специальная область оперативной памяти размером 256 (100h) байт.

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

CS, DS, ES и SS указывают на PSP.

4) Как определяется стек? Какую область памяти он занимает? Какие адреса?

Рисунок 5. Модуль os1_exe.exe в отладчике AFD.



Стек генерируется автоматически при создании COM-программы. SS – на начало (0h), регистр SP указывает на конец стека (FFFFh).

Загрузка «хорошего» EXE модуля в основную память.

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

EXE-файл загружается, начиная с адреса PSP:0100h. В процессе загрузки считывается информация заголовка EXE в начале файла и выполняется перемещение адресов сегментов.

Регистры CS, IP, SS и SP инициализированы значениями, указанными в заголовке EXE. DS и ES устанавливаются на начало сегмента PSP, SS– на начало сегмента стека, CS – на начало сегмента команд. В IP - смещение точки входа в программу.

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

На начало PSP.

3) ***Как определяется стек?***

Стек располагается в оперативной памяти в сегменте стека, и поэтому адресуется относительно сегментного регистра SS.

4) ***Как определяется точка входа?***

Точка входа определяется при помощи директивы END. В роли *необязательного операнда* здесь выступает *метка* (*или выражение*), определяющая адрес, с которого начинается выполнение программы (точка входа в программу).

Вывод.

В ходе лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов загрузки их в основную память.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл os1_com.asm

```
TESTPC SEGMENT

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

        ORG 100H

START:

        JMP BEGIN



Type_PC     db    'PC type:
PC',0DH,0AH,'$'

Type_XT     db    'PC type:
PC/XT',0DH,0AH,'$'

Type_AT     db    'PC type:
AT',0DH,0AH,'$'

Type_PS30   db    'PC type: PS2 model
30',0DH,0AH,'$'

Type_PS50   db    'PS2 model 50 or
60',0DH,0AH,'$'

Type_PS80   db    'PS2 model
80',0DH,0AH,'$'

Type_PCjr   db    'PC type:
PCjr',0DH,0AH,'$'

Type_PCCont db    'PC type: PC
Convertible',0DH,0AH,'$'

Type_Unknown db    '   ', 0DH, 0AH, '$'



System_version db   'System version: ',
'$'
```

```
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

    push CX
```

```
Number_OEM     db   'OEM number:     ', 0DH, 0AH, '$'

Number_User    db   'User serial number:         ', 0DH, 0AH, '$'

Number_Version db   '  .  ', 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 AL переводится в два символа 16с.с. числа в AX

    push CX

    mov AH,AL

    call TETR_TO_HEX

    xchg AL,AH

    mov CL,4

    shr AL,CL
```

```
    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

PRINT  PROC NEAR    ; вывод строки на экран

      push ax

      mov ah, 9h

      int 21h

      pop ax
```

```
    call TETR_TO_HEX ; в AL старшая
цифра

    pop CX ;в AH младшая

    ret

BYTE_TO_HEX ENDP



WRD_TO_HEX PROC near

;
```

```
      ret

PRINT ENDP



CHECK_PC PROC NEAR

      mov ax, 0F000h

      mov es, ax

      mov al, es:[0FFFEh]
;получаем байт

      cmp al, 0FFh

      je PC

      cmp al, 0FEh

      je XT

      cmp al, 0FBh

      je XT

      cmp al, 0FCh

      je AT

      cmp al, 0FAh

      je PS30

      cmp al, 0FCh

      je PS50

      cmp al, 0F8h

      je PS80

      cmp al, 0FDh

      je PCjr

      cmp al, 0F9h

      je PCCont

      jmp UNKNOWN
```

```
PC:
     lea dx, Type_PC
     jmp PRINT_STR

XT:
    lea dx, Type_XT
    jmp PRINT_STR

AT:
    lea dx, Type_AT
    jmp PRINT_STR

PS30:
    lea dx, Type_PS30
    jmp PRINT_STR

PS50:
    lea dx, Type_PS50
    jmp PRINT_STR
PS80:
    lea dx, Type_PS80
    jmp PRINT_STR

PCjr:
    lea dx, Type_PCjr
    jmp PRINT_STR
```

```
PCCont:
    lea dx, Type_PCCont
    jmp PRINT_STR

UNKNOWN:
    call BYTE_TO_HEX
    lea si, Type_Unknown
    mov [si], al
    mov [si+1], ah
    mov dx, si

PRINT_STR:
    call PRINT
    ret
CHECK_PC ENDP

BEGIN:
    push ax
    push dx
    push es
    push si
    call CHECK_PC
    pop si
    pop es
    pop dx
    pop ax
```

```
; выход в DOS
    mov AH, 30H
    int 21h

    lea dx, System_version
    call PRINT

    lea si, Number_Version
    mov dl, ah
    inc si
    call BYTE_TO_DEC
    mov al, dl
    add si, 3
    call BYTE_TO_DEC
    lea dx, Number_Version
    call PRINT

;номер OEM
    mov al, bh
    lea si, Number_OEM
    add si, 14
    call BYTE_TO_DEC
    lea dx, Number_OEM
    call PRINT

; номер пользователя
    mov    AX,CX
```

```
    lea di, Number_User
    add    di, 26
    call   WRD_TO_HEX

    mov al, bl
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax

    lea dx, Number_User
    call PRINT

    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
    END START ; конец модуля, START - точка входа
```

Файл os1_asm.asm

```
DOSSEG                                          ; Задание сегментов под ДОС
.MODEL  SMALL                                   ; Модель памяти-SMALL(Малая)
.STACK  100h
.DATA

Type_PC     db    'PC type: PC',0DH,0AH,'$'
Type_XT     db    'PC type: PC/XT',0DH,0AH,'$'
```

```
Type_AT     db    'PC type: AT',0DH,0AH,'$'
Type_PS30   db    'PC type: PS2 model 30',0DH,0AH,'$'
Type_PS50   db    'PS2 model 50 or 60',0DH,0AH,'$'
Type_PS80   db    'PS2 model 80',0DH,0AH,'$'
Type_PCjr   db    'PC type: PCjr',0DH,0AH,'$'
Type_PCCont db    'PC type: PC Convertible',0DH,0AH,'$'
Type_Unknown db    '   ', 0DH, 0AH, '$'

System_version db   'System version: ', '$'
Number_OEM     db   'OEM number:    ', 0DH, 0AH, '$'
Number_User    db   'User serial number:         ', 0DH, 0AH, '$'
Number_Version db   '  .  ', 0DH, 0AH, '$'
.CODE
START:
        JMP BEGIN
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 AL переводится в два символа 16с.с. 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
    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
   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
```

```
PRINT  PROC NEAR    ; вывод строки на экран
      push ax
      mov ah, 9h
      int 21H
      pop ax
      ret
PRINT ENDP

CHECK_PC PROC NEAR      ;проверка типа PC
      mov ax, 0F000h
      mov es, ax
      mov al, es:[0FFFEh] ;получаем байт

      cmp al, 0FFh
      je PC
      cmp al, 0FEh
      je XT
      cmp al, 0FBh
      je XT
      cmp al, 0FCh
      je AT
      cmp al, 0FAh
      je PS30
      cmp al, 0FCh
      je PS50
      cmp al, 0F8h
```

```
      je PS80
      cmp al, 0FDh
      je PCjr
      cmp al, 0F9h
      je PCCont
      jmp UNKNOWN

PC:
     lea dx, Type_PC
     jmp PRINT_STR

XT:
    lea dx, Type_XT
    jmp PRINT_STR

AT:
    lea dx, Type_AT
    jmp PRINT_STR

PS30:
    lea dx, Type_PS30
    jmp PRINT_STR

PS50:
    lea dx, Type_PS50
    jmp PRINT_STR
PS80:
```

```
    lea dx, Type_PS80
    jmp PRINT_STR

PCjr:
    lea dx, Type_PCjr
    jmp PRINT_STR

PCCont:
    lea dx, Type_PCCont
    jmp PRINT_STR

UNKNOWN:              ;неизвестный тип
    call BYTE_TO_HEX
    lea si, Type_Unknown
    mov [si], ax
    mov dx, si

PRINT_STR:
    call PRINT
    ret
CHECK_PC ENDP

BEGIN:
    push ax
    push dx
    push es
    push si
```

```
    mov  ax, @data       ; Загрузка в DS адреса начала
    mov  ds,ax           ; сегмента данных
    call CHECK_PC
    pop si
    pop es
    pop dx
    pop ax

    mov AH, 30H
    int 21h

    lea dx, System_version
    call PRINT

    lea si, Number_Version      ;  номер версии системы
    mov dl, ah
    inc si
    call BYTE_TO_DEC
    mov al, dl
    add si, 3             ; "перешагиваем" через точку
    call BYTE_TO_DEC
    lea dx, Number_Version
    call PRINT

    lea si, Number_OEM     ;номер OEM
    add si, 14
    mov al, bh
```

```
    call BYTE_TO_DEC
    lea dx, Number_OEM
    call PRINT

; номер пользователя
    mov    AX,CX
    lea di, Number_User
    add    di, 26
    call   WRD_TO_HEX

    mov al, bl
    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax

    lea dx, Number_User
    call PRINT

    xor AL,AL
    mov AH,4Ch
    int 21H
END START
    END START ; конец модуля, START - точка входа
```