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 загрузочных модулей

Студент гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Губкин А.Ф.

Санкт-Петербург
2022

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.

1. Написать текст исходного .COM модуля, который определяет тип PC и версию системы. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM.

2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в шаге 1 и отладить его. Таким образом, будет получен «хороший» .EXE.

3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на вопросы «Отличия исходных текстов COM и EXE программ».

4. Запустить FAR и открыть файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем открыть файл загрузочного модуля «хорошего» .EXE и сравнить его с предыдущими файлами. Ответить на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

5. Открыть отладчик TD.EXE и загрузить CO. Ответить на контрольные вопросы «Загрузка COM модуля в основную память». Представить в отчете план загрузки модуля .COM в основную память.

6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE в основную память».

7. Оформить отчет в соответствии с требованиями. Привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей — в отладчике.

Выполнение работы.

1. Файл *lb1_exe.exe* представляет собой «хороший» .exe модуль, исходный код которого содержится в файле *lb1_exe.asm*. В коде используем принцип разделения, разбивая данные, код, стэк на соответствующие сегменты. Передаём начало выполнения программы главной процедуре. *DS* указывает на сегмент данных, в котором хранятся строки для дальнейшего вывода на экран.

В файле описаны следующие процедуры:

print_function – отвечает за вывод строк (смещение находится в *DX*) с помощью прерывания *21h* и его функции *09h*.

TETR_TO_HEX, BYTE_TO_HEX, WRD_TO_HEX, BYTE_TO_DEC – процедуры, взятые из шаблона в методических указаниях. Отвечают за перевод бинарных чисел в системы счисления 16-ти и 10тиричную.

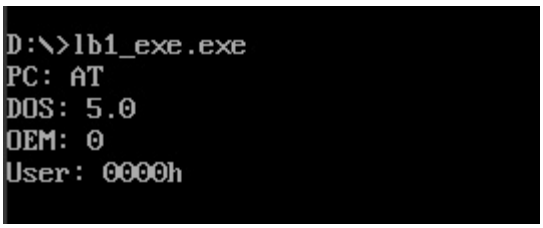
PC_proc – отвечает за определение и вывод типа *PC*. Из последнего байта ROM BIOS считывается тип *PC* (в *AL*), затем значение в *AL* пошагово сравнивается со значениями, соответствующими типам *PC*. При совпадении совершается прыжок к метке *print*, где вызываемая процедура *print_function* печатает соответствующую строку, содержащую тип *PC*.

DOS_proc – отвечает за определение и вывод версии *MS DOS*, информация о которой достаётся с помощью прерывании *21h* и его функции *30h*. Полученное значение переводится в строку и выводится на экран с поясняющим комментарием, хранящимся в сегменте данных.

OEM_proc и *User_proc* – отвечают за определение и вывод серийных номеров OEM и пользователя соответственно. Работают аналогично *DOS_proc,* только информация хранится не в *AH* и *AL,* а в *BH, BL* и *CX.*

В процедуре *begin вызывается PC_proc, DOS_proc, OEM_proc* и *User_proc*, а затем с помощью прерывания *21h* и его функции *4ch* всё завершается.

Рис. 1 - Результат работы «хорошего» *.exe* модуля *lb1_exe:*



2. Файл *lb1_com.com* представляет собой «хороший» .com модуль, исходный код которого содержится в файле *lb1_com.asm.* В отличие от кода для .exe модуля здесь объявляется единственный сегмент. Директива *org 100h* используется чтобы установить на конец *PSP - CS:IP*. Все процедуры написаны аналогично файлу *lb1_exe.asm.*

Рис. 2 - Результат работы «хорошего» *.com* модуля *lb1_com*:

Рис. 3 - Результат работы «плохого» *.exe* модуля *lb1_com:*

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?

Один.

2. EXE-программа?

Несколько (необязательно).

3. Какие директивы должны быть обязательно в тексте COM-программы? Org 100h - чтобы установить *CS:IP* на конец *PSP.* ASSUME CS:SEG, DS:SEG.

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

Нет. В модуле нет таблицы настройки, а адреса сегментных регистров устанавливаются при запуске, следовательно команды с указанием сегментов применять нельзя.

Отличия форматов файлов .com и .exe модулей:

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

Исходный код включает в себя единственный сегмент. В его начале определяются все сегментные регистры. Код с нулевого адреса.

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

Такой модуль содержит только один сегмент. Код располагается с адреса 300h. С адреса 0 идут заголовок, таблица настройки и смещение.

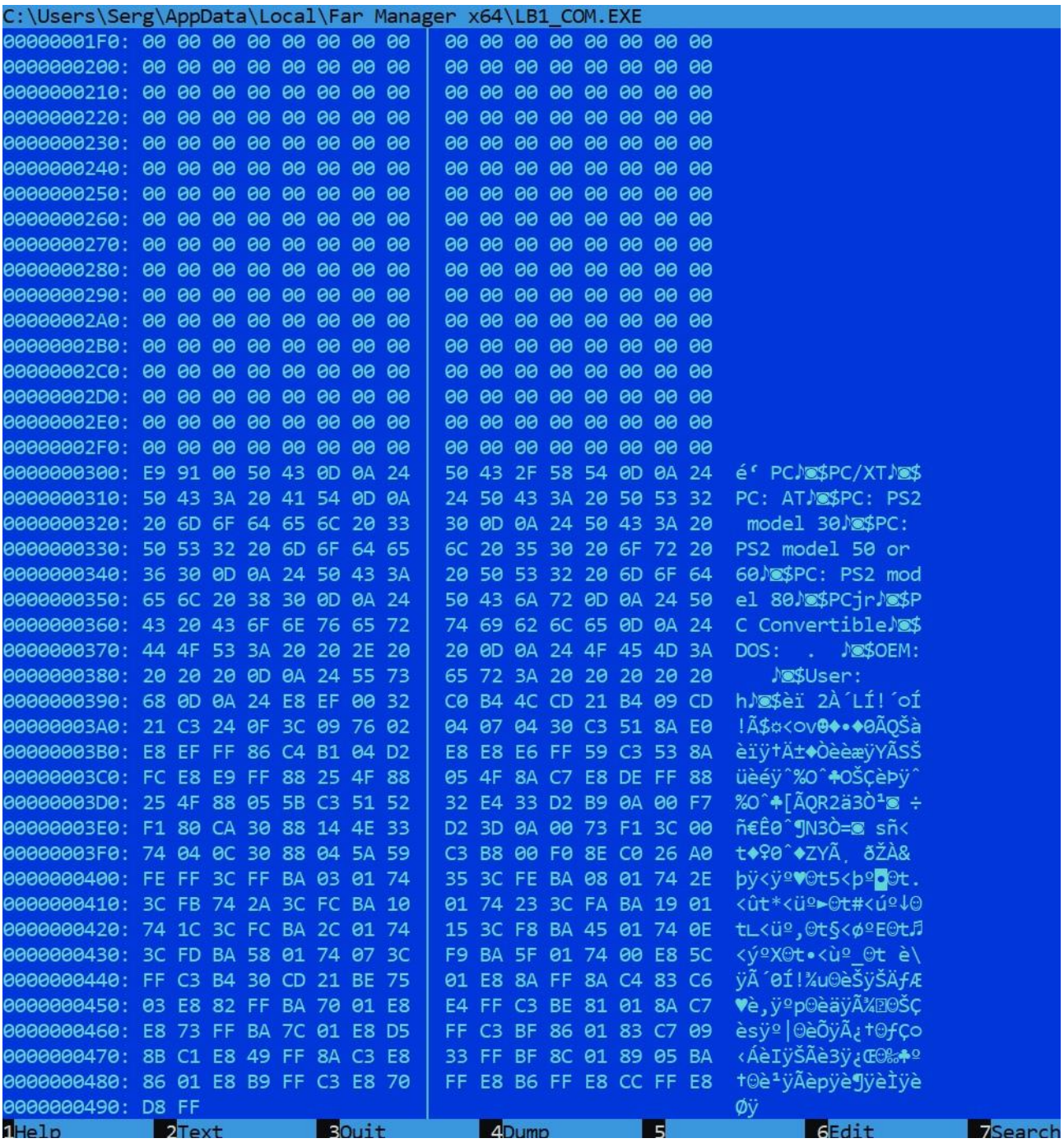


Рис. 4– «Плохой» .exe модуль

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

Такой модуль структурирован так, что имеет несколько сегментов. В начале модуля так же находится заголовок и таблица настройки, после чего идут сами сегменты в порядке, в котором они обозначены в коде.

Рис. 5 – «Хороший» .exe модуль

Загрузка .com модуля в основную память:

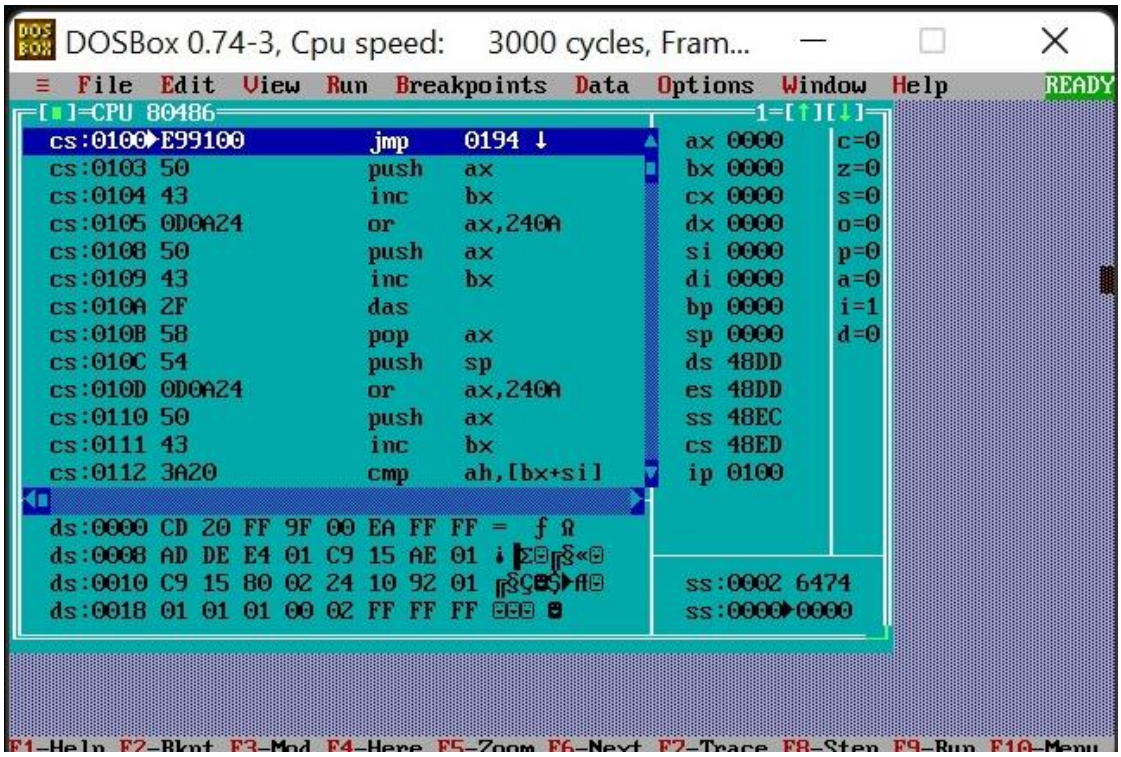


Рис. 6 – Загруженный .com модуль

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

Свободный сегмент выделяется в основной памяти, первые 256 байтов которого занимает PSP. После PSP идёт сама программа.

С адреса 48DD:0100 (рис. 7).

2. Что располагается с адреса 0?

PSP.

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

48DD (рис. 7). На начало PSP.

4. Как определяется стэк? Какую область памяти он занимает? Какие адреса?

Стэк занимает весь сегмент программы (64 Кб памяти). Начало сегмента SS (48DD), последний адрес SP (FFFE).

Адреса 48DD:0000 – 48DD:FFFE.

Загрузка «хорошего» .exe модуля в основную память:

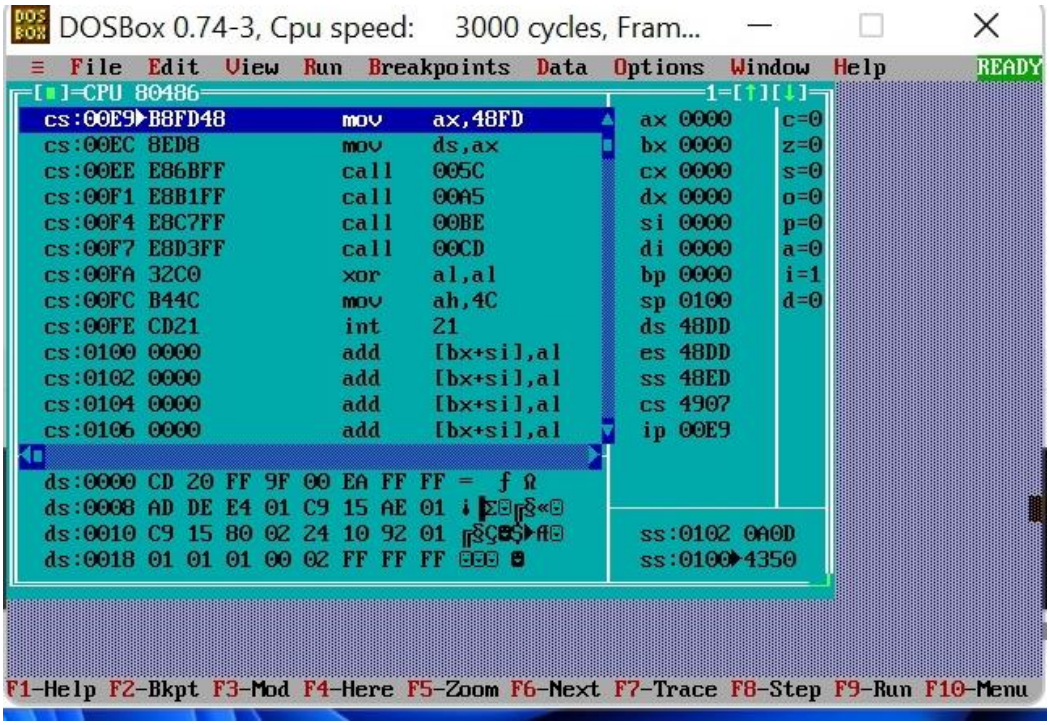


Рис. 7 – Загруженный .exe модуль

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

В память загружается PSP, следом .exe модуль в соответствии с заголовком. Значение см. рис. 8.

2. На что указывают DS и ES?

На начало PSP.

2. Как определяется стэк?

Стэк занимает и ограничивается отдельным сегментом, на границы которого указывают соответственно SS и SP.

3. Как определяется точка входа?

С помощью директивы END.

Выводы.

Были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А.

Исходный код.

lb1_exe.asm:

```
AStack SEGMENT STACK
    DW 128 DUP (?)
AStack ENDS

data SEGMENT

PCvar1 DB 'PC',0DH,0AH,'$'
PCvar2 DB 'PC/XT',0DH,0AH,'$'
PCvar3 DB 'PC: AT',0DH,0AH,'$'
PCvar4 DB 'PC: PS2 model 30',0DH,0AH,'$'
PCvar5 DB 'PC: PS2 model 50 or 60',0DH,0AH,'$'
PCvar6 DB 'PC: PS2 model 80',0DH,0AH,'$'
PCvar7 DB 'PCjr',0DH,0AH,'$'
PCvar8 DB 'PC Convertible',0DH,0AH,'$'
DOS DB 'DOS:  .  ',0DH,0AH,'$'
OEM DB 'OEM:   ',0DH, 0AH, '$'
USER DB 'User:     h',0DH, 0AH, '$'

data ENDS

main SEGMENT
    ASSUME CS:main, DS:data, SS:AStack


print_function PROC
    mov ah, 09h
    int 21h
    ret
print_function 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
      ;байт в AL переводится в два символа
      ;шестнадцатиричного 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
      ;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 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


PC_proc PROC
    mov AX, 0F000h
    mov ES, AX
    mov AL, ES:[0FFFEh]

    cmp AL, 0ffh
    mov DX, offset PCvar1
    je print

    cmp AL, 0feh
    mov DX, offset PCvar2
    je print

    cmp AL, 0fbh
    je print
```

```
    cmp AL, 0fch
    mov DX, offset PCvar3
    je print

    cmp AL, 0fah
    mov DX, offset PCvar4
    je print

    cmp AL, 0fch
    mov DX, offset PCvar5
    je print

     cmp AL, 0f8h
    mov DX, offset PCvar6
    je print

    cmp AL, 0fdh
    mov DX, offset PCvar7
    je print

    cmp AL, 0f9h
    mov DX, offset PCvar8
    je print

print:
    call print_function
    ret
PC_proc ENDP


DOS_proc PROC
    mov AH, 30h
    int 21h
    mov SI, offset DOS + 5
    call BYTE_TO_DEC
    mov AL, AH
    mov SI, offset DOS + 8
    call BYTE_TO_DEC
    mov DX, offset DOS
```

```
    call print_function
    ret
DOS_proc ENDP

OEM_proc PROC
    mov SI, offset OEM + 5
    mov AL, BH
    call BYTE_TO_DEC
    mov DX, offset OEM
    call print_function
    ret
OEM_proc ENDP

User_proc PROC
   mov DI, offset User
    add DI, 9
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    mov DI, offset User + 6
    mov [DI], AX
    mov DX, offset User
    call print_function
    ret
User_proc ENDP

begin PROC FAR
    mov AX, data
    mov DS, AX
    call PC_proc
    call DOS_proc
    call OEM_proc
    call User_proc
    xor AL, AL
    mov AH, 4Ch
    int 21h
begin ENDP
```

```
main ENDS
END begin
```

lb1_com.asm:

```
code SEGMENT
    ASSUME CS:code, DS:code
org 100H
begin:
    jmp opening

data:

PCvar1 DB 'PC',0DH,0AH,'$'
PCvar2 DB 'PC/XT',0DH,0AH,'$'
PCvar3 DB 'PC: AT',0DH,0AH,'$'
PCvar4 DB 'PC: PS2 model 30',0DH,0AH,'$'
PCvar5 DB 'PC: PS2 model 50 or 60',0DH,0AH,'$'
PCvar6 DB 'PC: PS2 model 80',0DH,0AH,'$'
PCvar7 DB 'PCjr',0DH,0AH,'$'
PCvar8 DB 'PC Convertible',0DH,0AH,'$'
DOS DB 'DOS:  .  ',0DH,0AH,'$'
OEM DB 'OEM:   ',0DH, 0AH, '$'
USER DB 'User:     h',0DH, 0AH, '$'

opening:
    call continued
    xor AL, AL
    mov AH, 4Ch
    int 21h


print_function PROC
    mov ah, 09h
    int 21h
    ret
print_function 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
      ;байт в AL переводится в два символа
      ;шестнадцатиричного 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
      ;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 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


PC_proc PROC
    mov AX, 0F000h
    mov ES, AX
    mov AL, ES:[0FFFEh]

    cmp AL, 0ffh
    mov DX, offset PCvar1
    je print

    cmp AL, 0feh
    mov DX, offset PCvar2
    je print
```

```
    cmp AL, 0fbh
    je print

    cmp AL, 0fch
    mov DX, offset PCvar3
    je print

    cmp AL, 0fah
    mov DX, offset PCvar4
    je print

    cmp AL, 0fch
    mov DX, offset PCvar5
    je print

     cmp AL, 0f8h
    mov DX, offset PCvar6
    je print

    cmp AL, 0fdh
    mov DX, offset PCvar7
    je print

    cmp AL, 0f9h
    mov DX, offset PCvar8
    je print

print:
    call print_function
    ret
PC_proc ENDP


DOS_proc PROC
    mov AH, 30h
    int 21h
    mov SI, offset DOS + 5
    call BYTE_TO_DEC
    mov AL, AH
```

```
    mov SI, offset DOS + 8
    call BYTE_TO_DEC
    mov DX, offset DOS
    call print_function
    ret
DOS_proc ENDP

OEM_proc PROC
    mov SI, offset OEM + 5
    mov AL, BH
    call BYTE_TO_DEC
    mov DX, offset OEM
    call print_function
    ret
OEM_proc ENDP

User_proc PROC
   mov DI, offset User
    add DI, 9
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    mov DI, offset User + 6
    mov [DI], AX
    mov DX, offset User
    call print_function
    ret
User_proc ENDP

continued PROC
    call PC_proc
    call DOS_proc
    call OEM_proc
    call User_proc
continued ENDP

code ENDS
END begin
```