

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

Охотникова Г.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

Задание.

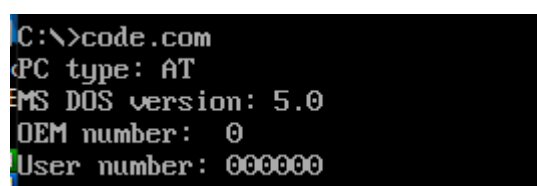
1. Написать текст исходного .COM модуля, который определяет тип РС и версию системы. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM.
2. Написать текст исходного .EXE модуля, который выполняет те же функции,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

При выполнении данной лабораторной работы в шаблон из методических указаний были добавлены три процедуры: одна — выводит строку на экран, вторая — проверяет информацию о типе ПК и выводит ее на экран, третья — о типе ОС и так же выводит ее на экран.

Для того, чтобы построить «хороший» COM модуль, был написан файл с исходным кодом формата asm, а путем преобразования из него был получен «плохой» EXE модуль.

Пример запуска «хорошего» COM модуля на рисунке 1:



```
C:\>code.com
PC type: AT
MS DOS version: 5.0
OEM number: 0
User number: 000000
```

Рисунок 1

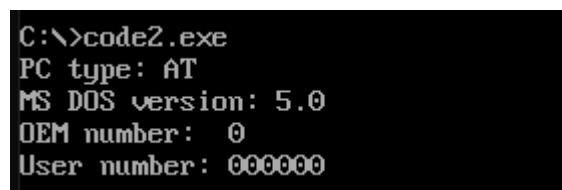
Пример запуска «плохого» EXE модуля на рисунке 2:



```
C:\>code.exe
PC type: PC
MS DOS version: 5.0
OEM number: 0
User number: 000000
PC type: PC
PC type: PC
PC type: PC
PC type: PC
```

Рисунок 2

Для «хорошего» EXE модуля был написан файл с исходным кодом, который отличается от «хорошего» COM модуля разметкой сегментов. Пример запуска на рисунке 3:



```
C:\>code2.exe
PC type: AT
MS DOS version: 5.0
OEM number: 0
User number: 000000
```

Рисунок 3

Исходный программный код см. в приложении А.

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?

Ответ: Один.

2. EXE-программа?

Ответ: Больше или равно одного.

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

Ответ: директива `org 100h`, которая смещает адресацию на 256 байт (размер PSP), директива `ASSUME` для того, чтобы компилятор распознал, к какому сегментному регистру каждый сегмент.

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

Ответ: Нет. Так как в COM модуле отсутствует таблица настроек, в которой содержатся описания адресов, зависящие от размещения модуля в операционной системе, нельзя использовать команды `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 состоит из одного сегмента. Адресация начинается с `0h`, но при загрузке программы произойдет смещение на PSP, который равен `100h`. (см. рисунок 4)

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

Что располагается с адреса `0`?

Ответ: «Плохой» EXE состоит из одного сегмента. Код будет начинаться с адреса `300h`. До этого будет находиться заголовок и relocation table (что занимает 512 байт), а также смещение `100h` (256 байт). (см. рисунок 5)

[illegible]

```
C:\Users\Okhot\OneDrive\Рабочий стол\ЛЭТИ\lab1\C0DE.EXE
0000000190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000300: F9 F7 01 58 43 20 74 79 70 65 3A 20 50 43 20 0D é+@PC type: PC
0000000310: 0A 24 50 43 20 74 79 70 65 3A 20 50 43 2F 58 43 ¤$PC type: PC/XT
0000000320: 20 0D 0A 24 50 43 20 74 79 70 65 3A 20 41 54 20 ¤$PC type: AT
0000000330: 0D 0A 24 50 43 20 74 79 70 65 3A 20 50 53 32 20 ¤$PC type: PS2
0000000340: 6D 6F 64 65 6C 20 33 38 20 0D 0A 24 50 43 20 74 model 30 ¤$PC t
0000000350: 79 70 65 3A 20 50 53 32 20 0D 6F 64 65 6C 20 35 pe: PS2 model 5
0000000360: 30 20 6F 72 20 36 30 20 0D 0A 24 50 43 20 74 79 or 60 ¤$PC ty
0000000370: 70 65 3A 20 50 53 32 20 6D 6F 64 65 6C 20 38 30 pe: PS2 model 80
0000000380: 20 0D 0A 24 50 43 20 74 79 70 65 3A 20 50 53 43 ¤$PC type: PSC
0000000390: 6A 72 20 0D 0A 24 50 43 20 74 79 70 65 3A 20 50 jr ¤$PC type: P
00000003A0: 43 20 43 6F 6E 76 65 72 74 69 62 6C 65 20 0D 0A C Convertible ¤$
00000003B0: 24 40 53 20 44 4F 53 20 76 65 72 73 69 6F 6E 3A ¤$MS DOS version:
00000003C0: 20 20 2E 20 20 0D 0A 24 4F 45 40 20 6E 75 6D 62 er: ¤$OEM numb
00000003D0: 65 72 3A 20 20 20 0D 0A 24 55 73 65 72 20 0E 75 er: ¤$User nu
00000003E0: 6D 62 65 72 3A 20 20 20 20 20 20 20 0D 0A 24 mber: ¤$¸$
00000003F0: 0F 3C 09 76 02 04 07 04 30 C3 51 8A 0E EF FF ¤cov0+...0AQSaëÿ
0000000400: 86 CA B1 04 D2 E8 E8 E6 FF 59 C3 58 BA FC E8 E9 +Ad+0eayvASûëé
0000000410: FF 88 25 4F 88 05 4F 8A C7 E8 DE FF 88 25 4F 88 y"%0+0Sçpy%"0
0000000420: 05 58 C3 51 52 32 F4 3B D2 09 0A 0D F7 F1 80 CA ¤[AR2ã3!¤ ±ñf
```

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

Ответ: «Хороший» EXE состоит из нескольких сегментов. Начинается с заголовка и relocation table. Затем расположены сами сегменты в том порядке, в котором они определены в коде (сегмент стека, сегмент данных, сегмент кода). (см. рисунок 6)

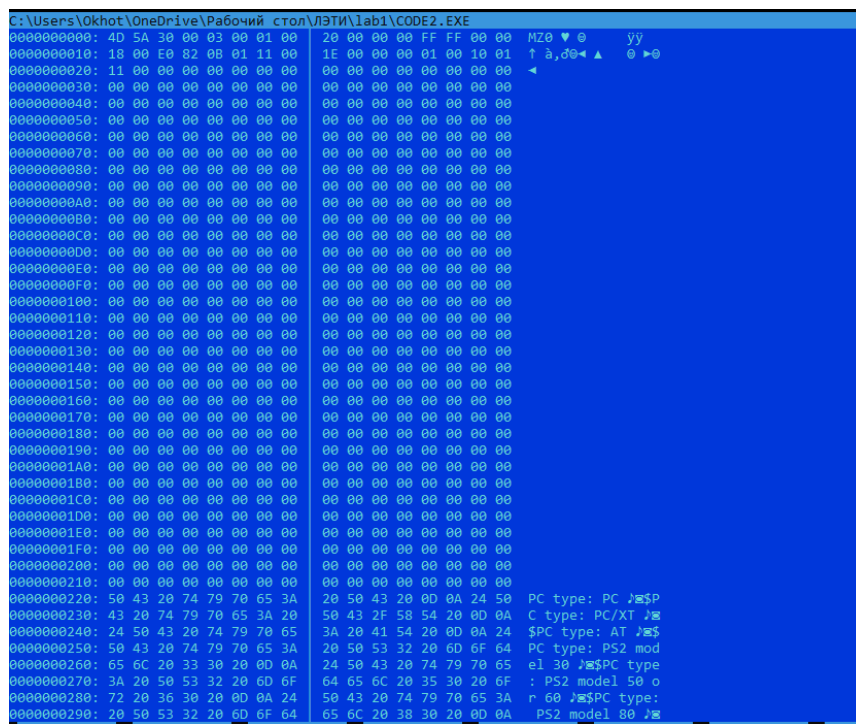


Рисунок 6 — "хороший" EXE модуль

Загрузка .com модуля в основную память:

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

Ответ: В основной памяти выделяется свободный сегмент. Загружается PSP, который занимает первые 256 байт, а затем записывает код программы. Он располагается с адреса CS:0100 = 48DD:0100.

2. Что располагается с адреса 0?

Ответ: PSP.

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

Ответ: Все сегментные регистры указывают на начало и имеют значение 48DD.

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

Ответ: Под стек отведен весь сегмент, в который загружена программа. Стэк занимает адреса 48DD:0000 – 48DD:FFFE. SS=48DD указывает на

начало сегмента, SP=FFFE указывает на последний адрес сегмента, который кратен двум.

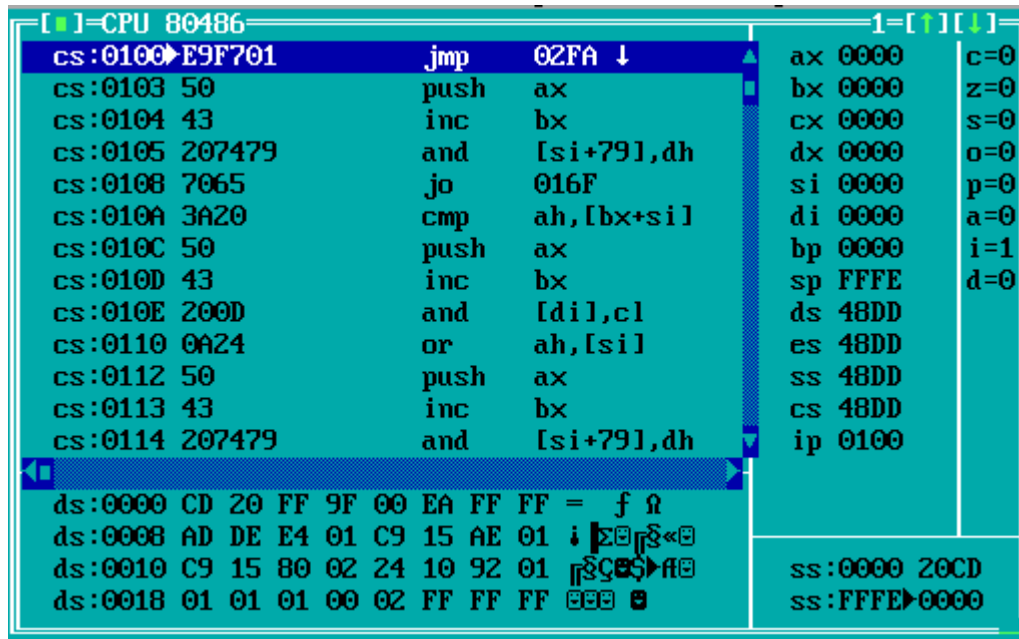


Рисунок 7 – «Хороший» COM модуль в отладчике

Загрузка «хорошего» .exe модуля в основную память:

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

Ответ: В первую очередь, загружается PSP, после которого загружается EXE модуль в соответствии с информацией в заголовке. DS=ES=48DD, CS=4907, SS=48ED.

2. На что указывают DS и ES?

Ответ: На начало PSP.

3. Как определяется стэк?

Ответ: В коде описывается стэковый сегмент. SS указывает на начало сегмента стэка, а SP – на конец стэка.

4. Как определяется точка входа?

Ответ: Точка входа определяется при помощи директивы END.

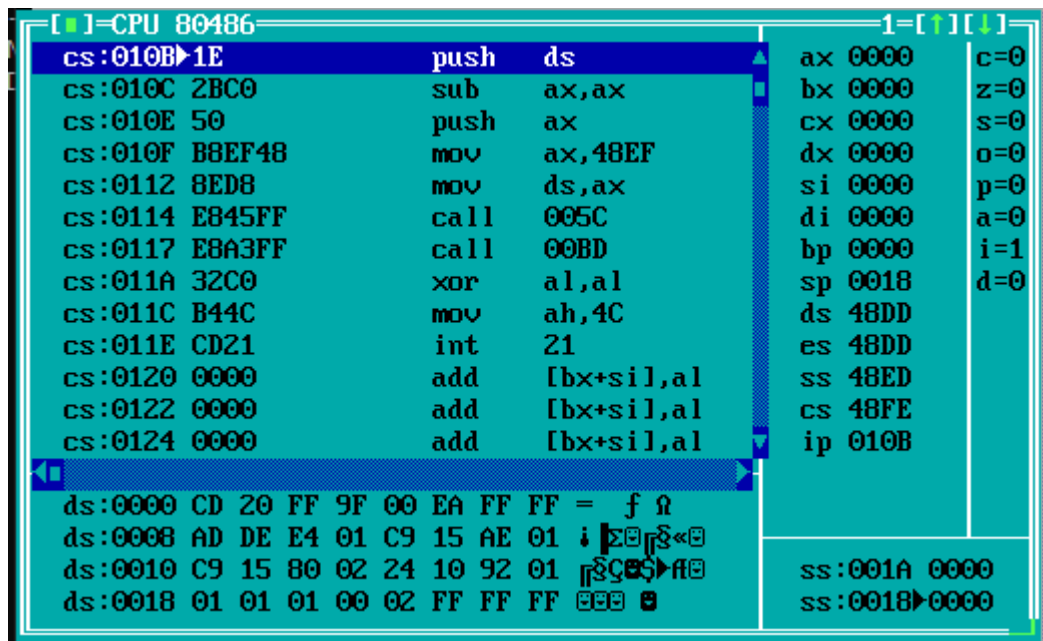


Рисунок 8 — «Хороший» EXE модуль в отладчике

Выводы.

При выполнении данной лабораторной работы были изучены принципы организации EXE и COM модулей, отличия между ними. Также написана программа, выводящая на экран информацию о версии DOS.

ПРИЛОЖЕНИЕ А.

Название файла: code.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
; ДАННЫЕ

PC db 'PC type: PC ', 0DH, 0AH, '$'
PC_XT db 'PC type: PC/XT ', 0DH, 0AH, '$'
AT db 'PC type: AT ', 0DH, 0AH, '$'
PS230 db 'PC type: PS2 model 30 ', 0DH, 0AH, '$'
PS25060 db 'PC type: PS2 model 50 or 60 ', 0DH, 0AH, '$'
PS280 db 'PC type: PS2 model 80 ', 0DH, 0AH, '$'
PCjr db 'PC type: PSCjr ', 0DH, 0AH, '$'
PC_convert db 'PC type: PC Convertible ', 0DH, 0AH, '$'
DOS_vs db 'MS DOS version: . ', 0DH, 0AH, '$'
OEM_num db 'OEM number: ', 0DH, 0AH, '$'
USER_num db 'User number: ', 0DH, 0AH, '$'

;STRING db 'Значение регистра AX= ', 0DH, 0AH, '$'

;ПРОЦЕДУРЫ
;-----
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 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 10с/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
;-----

PRINT_STRING PROC near
mov AH, 09h
int 21h
ret
PRINT_STRING ENDP

PCTYPE_CHECKING PROC near
mov AX, 0F000h
mov ES, AX
mov AL, ES:[0FFFEh] ;получаем байт

cmp AL, 0FFh
je pc_type

cmp AL, 0FEh
je pc_xt_type

cmp AL, 0FDh
je pcjr_type

cmp AL, 0FCh
je at_type

cmp AL, 0FCh
je ps2_5060

```

```

cmp AL, 0FBh
je pc_xt_type

cmp AL, 0FAh
je ps2_30

cmp AL, 0F8h
je ps2_80

cmp AL, 0F9h
je pc_conv

pc_type:
mov DX, offset PC
jmp end_proc

pc_xt_type:
mov DX, offset PC_XT
jmp end_proc

pcjr_type:
mov DX, offset PCjr
jmp end_proc

at_type:
mov DX, offset AT
jmp end_proc

ps2_5060:
mov DX, offset PS25060
jmp end_proc

ps2_80:
mov DX, offset PS280
jmp end_proc

ps2_30:
mov DX, offset PS230
jmp end_proc

```

```
pc_conv:
mov DX, offset PC_convert
jmp end_proc
```

```
end_proc:
call PRINT_STRING
ret
```

```
PCTYPE_CHECKING ENDP
```

```
OS_CHECKING PROC near
mov AH, 30h
int 21h
push AX
```

```
mov SI, offset DOS_vs
add si, 16
call BYTE_TO_DEC
pop AX
add SI, 3
mov AL, AH
call BYTE_TO_DEC
mov DX, offset DOS_vs
call PRINT_STRING
```

```
mov SI, offset OEM_num
add SI, 13
mov AL, BH
call BYTE_TO_DEC
mov DX, offset OEM_num
call PRINT_STRING
```

```
mov DI, offset USER_num
add DI, 18
mov AX, CX
call WRD_TO_HEX
mov AL, BL
call BYTE_TO_HEX
```

```

mov DI, offset USER_num
add DI, 13
mov [DI], AX
mov DX, offset USER_num
call PRINT_STRING
ret

end_proc2:
ret

OS_CHECKING ENDP
; КОД
BEGIN:
    call PCTYPE_CHECKING
    call OS_CHECKING
    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
    END START ;конец модуля, START - точка входа

```

Название файла: code2.asm

```

AStack SEGMENT STACK
    DW 12 DUP(?)
AStack ENDS

DATA SEGMENT
PC db 'PC type: PC ', 0DH, 0AH, '$'
PC_XT db 'PC type: PC/XT ', 0DH, 0AH, '$'
AT db 'PC type: AT ', 0DH, 0AH, '$'
PS230 db 'PC type: PS2 model 30 ', 0DH, 0AH, '$'
PS25060 db 'PC type: PS2 model 50 or 60 ', 0DH, 0AH, '$'
PS280 db 'PC type: PS2 model 80 ', 0DH, 0AH, '$'
PCjr db 'PC type: PSCjr ', 0DH, 0AH, '$'
PC_convert db 'PC type: PC Convertible ', 0DH, 0AH, '$'
DOS_vs db 'MS DOS version:  . ', 0DH, 0AH, '$'
OEM_num db 'OEM number:      ', 0DH, 0AH, '$'
USER_num db 'User number:      ', 0DH, 0AH, '$'

```

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, 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

;-----

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 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 10с/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
;-----

PRINT_STRING PROC near
    mov AH, 09h
    int 21h
    ret
PRINT_STRING ENDP

PCTYPE_CHECKING PROC near
    mov AX, 0F000h

```



```
mov ES, AX
mov AL, ES:[0FFFFh] ;получаем байт
```

```
cmp AL, 0FFh
je pc_type
```

```
cmp AL, 0FEh
je pc_xt_type
```

```
cmp AL, 0FDh
je pcjr_type
```

```
cmp AL, 0FCh
je at_type
```

```
cmp AL, 0FCh
je ps2_5060
```

```
cmp AL, 0FBh
je pc_xt_type
```

```
cmp AL, 0FAh
je ps2_30
```

```
cmp AL, 0F8h
je ps2_80
```

```
cmp AL, 0F9h
je pc_conv
```

```
pc_type:
mov DX, offset PC
jmp end_proc
```

```
pc_xt_type:
mov DX, offset PC_XT
jmp end_proc
```

```
pcjr_type:
```

```

mov DX, offset PCjr
jmp end_proc

at_type:
mov DX, offset AT
jmp end_proc

ps2_5060:
mov DX, offset PS25060
jmp end_proc

ps2_80:
mov DX, offset PS280
jmp end_proc

ps2_30:
mov DX, offset PS230
jmp end_proc

pc_conv:
mov DX, offset PC_convert
jmp end_proc

end_proc:
call PRINT_STRING
ret

PCTYPE_CHECKING ENDP

OS_CHECKING PROC near
mov AH, 30h
int 21h
push AX

mov SI, offset DOS_vs
add si, 16
call BYTE_TO_DEC
pop AX
add SI, 3

```

```

mov AL, AH
call BYTE_TO_DEC
mov DX, offset DOS_vs
call PRINT_STRING

mov SI, offset OEM_num
add SI, 13
mov AL, BH
call BYTE_TO_DEC
mov DX, offset OEM_num
call PRINT_STRING

mov DI, offset USER_num
add DI, 18
mov AX, CX
call WRD_TO_HEX
mov AL, BL
call BYTE_TO_HEX
mov DI, offset USER_num
add DI, 13
mov [DI], AX
mov DX, offset USER_num
call PRINT_STRING
ret

end_proc2:
ret

OS_CHECKING ENDP

Main PROC FAR
    push DS
    sub AX,AX
    push AX
    mov AX,DATA
    mov DS,AX

    call PCTYPE_CHECKING
    call OS_CHECKING

```

```
    xor AL,AL
    mov AH,4Ch
    int 21H

Main ENDP
CODE ENDS
    END Main
```