

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. 9381

Прибылов Н.А.

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 их загрузки в основную память.

Ход работы.

1. Был написан текст .COM модуля, который определяет тип РС и версию системы. За основу был взят шаблон со вспомогательными процедурами преобразований символов в 10/16-ю системы счисления. Далее была произведена компиляция и линковка в «плохой» .EXE модуль. По «плохому» .EXE модулю был создан «хороший» .COM модуль.

2. Был написан текст модуля .EXE для последующей компиляции и линковки в «хороший» .EXE файл.

3. Было произведено сравнение текстов исходного кода файлов .COM и .EXE

4. Файлы «хорошего» .COM, «плохого» .EXE и «хорошего» .EXE модулей были просмотрены в шестнадцатеричном виде.

5. Модуль .COM был исследован с помощью отладчика.

6. Модуль .EXE был исследован с помощью отладчика.

Функции.

Названия	Описание
TETR_TO_HEX	Перевод тетрады (4 младших бита AL) в 16-ю систему. Результат в AL.
BYTE_TO_HEX	Перевод байта AL в 16-ю систему. Результат: старшая цифра в AL, младшая в AH.
WRD_TO_HEX	Перевод слова AX в 16-ю систему. Адрес последнего символа результата в DI.

BYTE_TO_DEC

Перевод байта AL в 10-ю систему.

Адрес младшей цифры результата в SI.

PRINT

Вывод строки из DX на экран.

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

Единственный сегмент, в котором размещаются программный код, данные и стек.

2) EXE-программа?

Любое количество сегментов. Для кода, данных и стека предусматривают отдельные сегменты.

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

ORG 100h — этой директивой программный счётчик устанавливается на значение 100h=256, так как первые 256 байт занимает структура PSP (Program Segment Prefix) и располагать код следует после этого блока.

ASSUME — этой директивой устанавливается соответствие сегментного регистра определённому сегменту. Сама по себе никак не изменяет сегментные регистры, но без неё процессор не сможет вычислять реальные адреса меток.

END — этой директивой завершается любая программа на ассемблере.

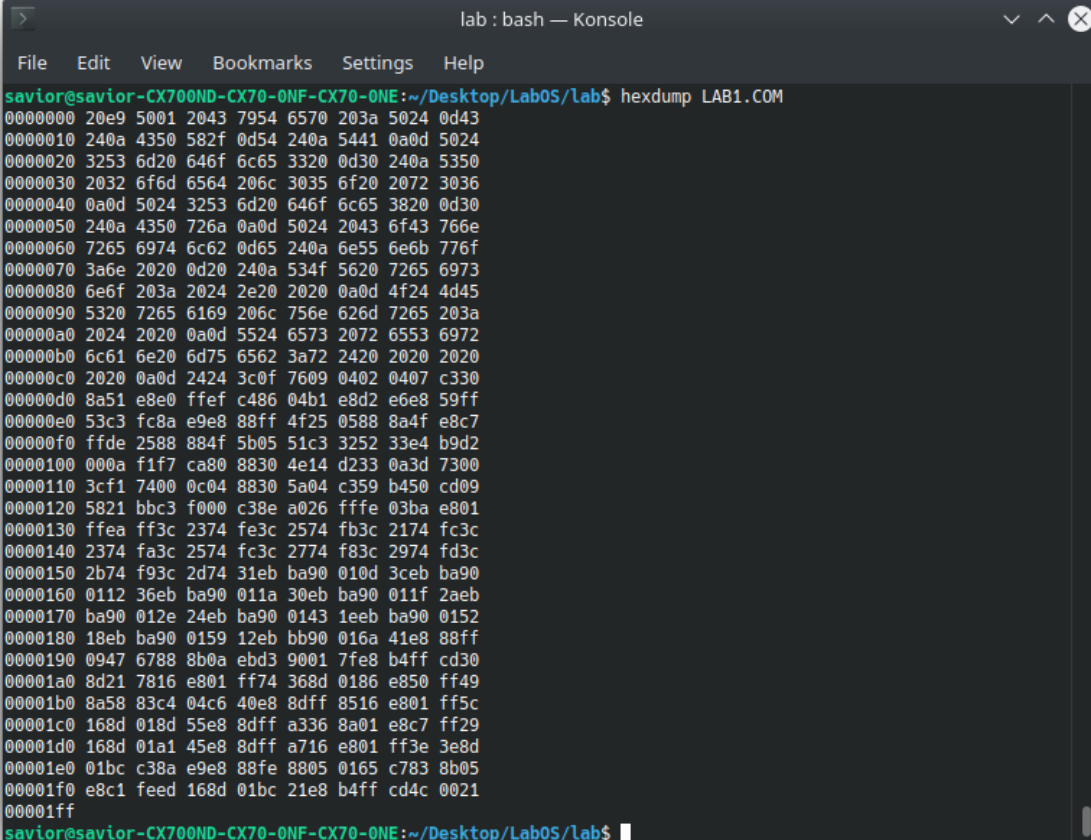
4) Все ли форматы команд можно использовать в COM-программе?

Нельзя использовать команды вида seg <name>, так как в COM-программах отсутствует таблица настроек.

Отличия форматов COM и EXE модулей

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

Код, данные и стек располагаются в одном сегменте. Код начинается с адреса 0.



```
lab : bash — Konsole
File Edit View Bookmarks Settings Help
savior@savior-CX700ND-CX70-0NF-CX70-0NE:~/Desktop/Lab05/Lab$ hexdump LAB1.COM
00000000 20e9 5001 2043 7954 6570 203a 5024 0d43
00000010 240a 4350 582f 0d54 240a 5441 0a0d 5024
00000020 3253 6d20 646f 6c65 3320 0d30 240a 5350
00000030 2032 6f6d 6564 206c 3035 6f20 2072 3036
00000040 0a0d 5024 3253 6d20 646f 6c65 3820 0d30
00000050 240a 4350 726a 0a0d 5024 2043 6f43 766e
00000060 7265 6974 6c62 0d65 240a 6e55 6e6b 776f
00000070 3a6e 2020 0d20 240a 534f 5620 7265 6973
00000080 6e6f 203a 2024 2e20 2020 0a0d 4f24 4d45
00000090 5320 7265 6169 206c 756e 626d 7265 203a
000000a0 2024 2020 0a0d 5524 6573 2072 6553 6972
000000b0 6c61 6e20 6d75 6562 3a72 2420 2020 2020
000000c0 2020 0a0d 2424 3c0f 7609 0402 0407 c330
000000d0 8a51 e8e0 ffe0 c486 04b1 e8d2 e6e8 59ff
000000e0 53c3 fc8a e9e8 88ff 4f25 0588 8a4f e8c7
000000f0 ffde 2588 884f 5b05 51c3 3252 33e4 b9d2
00000100 000a f1f7 ca80 8830 4e14 d233 0a3d 7300
00000110 3cf1 7400 0c04 8830 5a04 c359 b450 cd09
00000120 5821 bbc3 f000 c38e a026 ffe0 03ba e801
00000130 ffea ff3c 2374 fe3c 2574 fb3c 2174 fc3c
00000140 2374 fa3c 2574 fc3c 2774 f83c 2974 fd3c
00000150 2b74 f93c 2d74 31eb ba90 010d 3ceb ba90
00000160 0112 36eb ba90 011a 30eb ba90 011f 2aeb
00000170 ba90 012e 24eb ba90 0143 1eeb ba90 0152
00000180 18eb ba90 0159 12eb bb90 016a 41e8 88ff
00000190 0947 6788 8b0a ebd3 9001 7fe8 b4ff cd30
000001a0 8d21 7816 e801 ff74 368d 0186 e850 ff49
000001b0 8a58 83c4 04c6 40e8 8dff 8516 e801 ff5c
000001c0 168d 018d 55e8 8dff a336 8a01 e8c7 ff29
000001d0 168d 01a1 45e8 8dff a716 e801 ff3e 3e8d
000001e0 01bc c38a e9e8 88fe 8805 0165 c783 b005
000001f0 e8c1 feed 168d 01bc 21e8 b4ff cd4c 0021
00001fff
savior@savior-CX700ND-CX70-0NF-CX70-0NE:~/Desktop/Lab05/Lab$
```

Рис. 1 — 16-й вид модуля .COM

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

Код, данные и стек располагаются в одном сегменте. Код начинается с адреса 300h. С адреса 0 располагается заголовок и таблица настройки адресов.

```
lab : bash — Konsole
File Edit View Bookmarks Settings Help
savior@savior-CX700ND-CX70-0NF-CX70-0NE:~/Desktop/Lab05/Lab$ hexdump LAB1badEXE.EXE
00000000 5a4d 00ff 0003 0000 0020 0000 ffff 0000
00000010 0000 7560 0100 0000 001e 0000 0001 0000
00000020 0000 0000 0000 0000 0000 0000 0000 0000
*
0000300 20e9 5001 2043 7954 6570 203a 5024 0d43
0000310 240a 4350 582f 0d54 240a 5441 0a0d 5024
0000320 3253 6d20 646f 6c65 3320 0d30 240a 5350
0000330 2032 6f6d 6564 206c 3035 6f20 2072 3036
0000340 0a0d 5024 3253 6d20 646f 6c65 3820 0d30
0000350 240a 4350 726a 0a0d 5024 2043 6f43 766e
0000360 7265 6974 6c62 0d65 240a 6e55 6e6b 776f
0000370 3a6e 2020 0d20 240a 534f 5620 7265 6973
0000380 6e6f 203a 2024 2e20 2020 0a0d 4f24 4d45
0000390 5320 7265 6169 206c 756e 626d 7265 203a
00003a0 2024 2020 0a0d 5524 6573 2072 6553 6972
00003b0 6c61 6e20 6d75 6562 3a72 2420 2020 2020
00003c0 2020 0a0d 2424 3c0f 7609 0402 0407 c330
00003d0 8a51 e8e0 ffef c486 04b1 e8d2 e6e8 59ff
00003e0 53c3 fc8a e9e8 88ff 4f25 0588 8a4f e8c7
00003f0 ffde 2588 884f 5b05 51c3 3252 33e4 b9d2
0000400 000a f1f7 ca80 8830 4e14 d233 0a3d 7300
0000410 3cf1 7400 0c04 8830 5a04 c359 b450 cd09
0000420 5821 bbc3 f000 c38e a026 fffe 03ba e801
0000430 ffea ff3c 2374 fe3c 2574 fb3c 2174 fc3c
0000440 2374 fa3c 2574 fc3c 2774 f83c 2974 fd3c
0000450 2b74 f93c 2d74 31eb ba90 010d 3ceb ba90
0000460 0112 36eb ba90 011a 30eb ba90 011f 2aeb
0000470 ba90 012e 24eb ba90 0143 1eeb ba90 0152
0000480 18eb ba90 0159 12eb bb90 016a 41e8 88ff
0000490 0947 6788 8b0a ebd3 9001 7fe8 b4ff cd30
00004a0 8d21 7816 e801 ff74 368d 0186 e850 ff49
00004b0 8a58 83c4 04c6 40e8 8dff 8516 e801 ff5c
00004c0 168d 018d 55e8 8dff a336 8a01 e8c7 ff29
00004d0 168d 01a1 45e8 8dff a716 e801 ff3e 3e8d
00004e0 01bc c38a e9e8 88fe 8805 0165 c783 8b05
00004f0 e8c1 feed 168d 01bc 21e8 b4ff cd4c 0021
00004ff
savior@savior-CX700ND-CX70-0NF-CX70-0NE:~/Desktop/Lab05/Lab$
```

Рис. 2 — 16-й вид «плохого» модуля .EXE

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

Код, данные и стек располагаются в различных сегментах, в «плохом» же — в одном. В «хорошем» EXE код начинается с адреса 200h, т.к. отсутствует директива ORG 100h, которая выделяла 100h байт под PSP в «плохом» EXE.

```
lab : bash — Konsole
File Edit View Bookmarks Settings Help
savior@savior-CX700ND-CX70-0NF-CX70-0NE:~/Desktop/Lab05/Lab$ hexdump LAB1goodEXE.EXE
00000000 5a4d 0004 0003 0001 0020 0011 ffff 0021
00000010 0100 168e 0000 0000 001e 0000 0001 0062
00000020 0000 0000 0000 0000 0000 0000 0000 0000
*
00002000 5feb 2490 3c0f 7609 0402 0407 c330 8a51
00002010 e8e0 ffef c486 04b1 e8d2 e6e8 59ff 53c3
00002020 fc8a e9e8 88ff 4f25 0588 8a4f e8c7 ffde
00002030 2588 884f 5b05 51c3 3252 33e4 b9d2 000a
00002040 f1f7 ca80 8830 4e14 d233 0a3d 7300 3cf1
00002050 7400 0c04 8830 5a04 c359 b450 cd09 5821
00002060 b8c3 0014 d88e 00bb 8ef0 26c3 fea0 baff
00002070 0002 e5e8 3cff 74ff 3c23 74fe 3c25 74fb
00002080 3c21 74fc 3c23 74fa 3c25 74fc 3c27 74f8
00002090 3c29 74fd 3c2b 74f9 eb2d 9031 0cba eb00
000020a0 903c 11ba eb00 9036 19ba eb00 9030 1eba
000020b0 eb00 902a 2dba eb00 9024 42ba eb00 901e
000020c0 51ba eb00 9018 58ba eb00 9012 69bb e800
000020d0 ff3c 4788 8809 0a67 d38b 01eb e890 ff7a
000020e0 30b4 21cd 168d 0077 6fe8 8dff 8536 5000
000020f0 44e8 58ff c48a c683 e804 ff3b 168d 0084
00003000 57e8 8dff 8c16 e800 ff50 368d 00a2 c78a
00003010 24e8 8dff a016 e800 ff40 168d 00a6 39e8
00003020 8dff bb3e 8a00 e8c3 fee4 0588 6588 8301
00003030 05c7 c18b e8e8 8dfe bb16 e800 ff1c 4cb4
00003040 21cd 4350 5420 7079 3a65 2420 4350 0a0d
00003050 5024 2f43 5458 0a0d 4124 0d54 240a 5350
00003060 2032 6f6d 6564 206c 3033 0a0d 5024 3253
00003070 6d20 646f 6c65 3520 2030 726f 3620 0d30
00003080 240a 5350 2032 6f6d 6564 206c 3038 0a0d
00003090 5024 6a43 0d72 240a 4350 4320 6e6f 6576
000030a0 7472 6269 656c 0a0d 5524 6b6e 6f6e 6e77
000030b0 203a 2020 0a0d 4f24 2053 6556 7372 6f69
000030c0 3a6e 2420 2020 202e 0d20 240a 454f 204d
000030d0 6553 6972 6c61 6e20 6d75 6562 3a72 2420
000030e0 2020 0d20 240a 7355 7265 5320 7265 6169
000030f0 206c 756e 626d 7265 203a 2024 2020 2020
00004000 0d20 240a
0000404
savior@savior-CX700ND-CX70-0NF-CX70-0NE:~/Desktop/Lab05/Lab$
```

Рис. 3 — 16-й вид «хорошего» модуля EXE

Загрузка COM модуля в основную память

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

После загрузки программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h, поскольку первые 100h=256 байт отведены для структуры PSP.

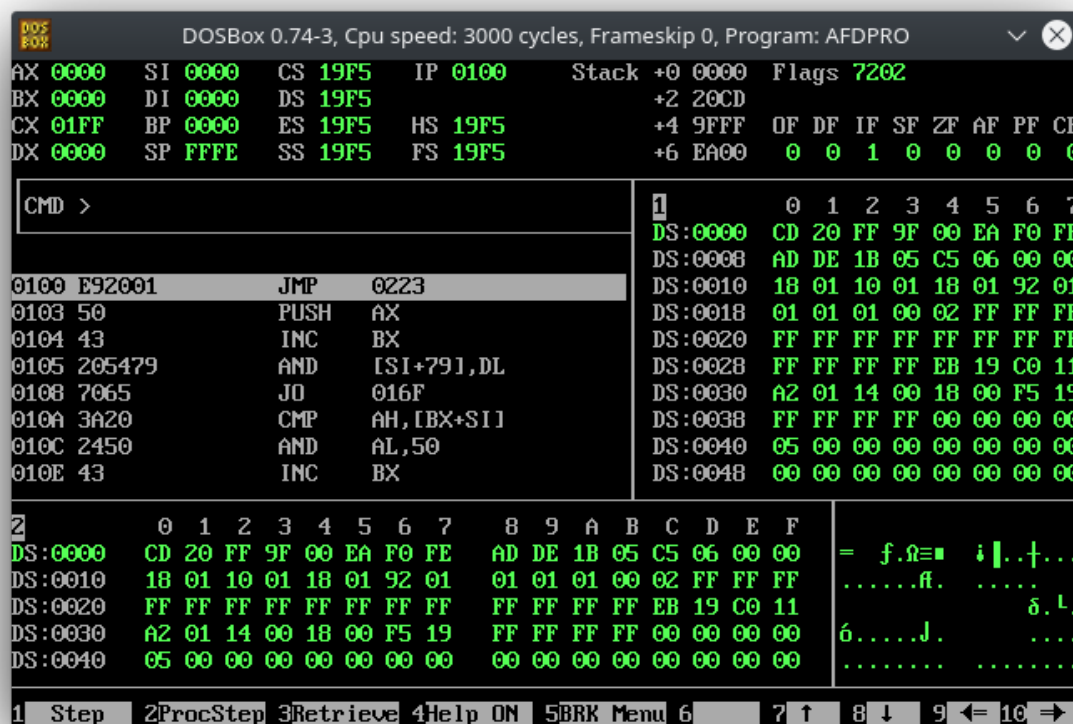


Рис. 4 — .COM модуль в отладчике

2) Что располагается с адреса 0?

Структура PSP — префикс программного сегмента.

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

CS, DS, ES, SS указывают на начало единственного сегмента, т.е. фактически на начало структуры PSP.

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

Стек создаётся автоматически, регистр SS (stack segment) указывает на начало — 0h, SP (stack pointer) — на FFFEh. Заполнение стека происходит в сторону уменьшения адресов.

Загрузка «хорошего» EXE модуля в основную память

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

Система пристраивает к началу программы префикс программы PSP размером 256 байт. В процессе загрузки считывается информация заголовка модуля. Загрузив программу в память, система инициализирует сегментные регистры, так что регистры DS (data segment) и ES (extra segment) указывают на начало PSP, CS (code segment) — на начало сегмента кода, SS — на начало сегмента стека. В указатель команд IP загружается смещение точки входа в программу, в указатель стека SP — смещение конца сегмента стека.

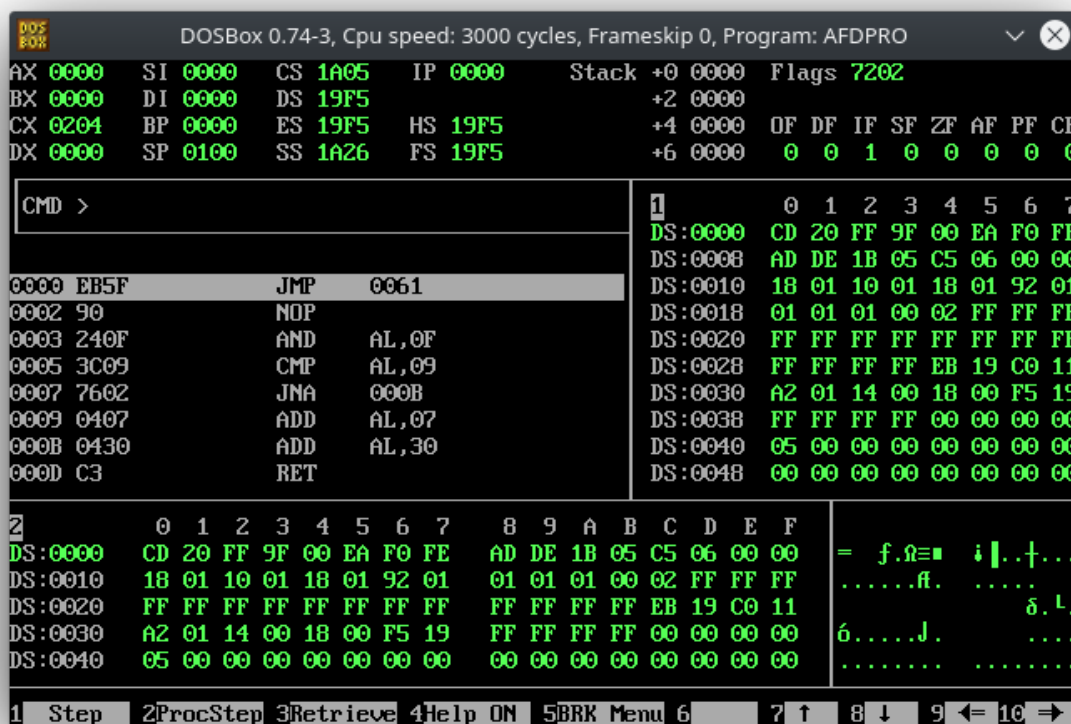


Рис. 5 — EXE модуль в отладчике

2) На что указывают регистры DS и ES?

После загрузки программы в память DS (data segment) и ES (extra segment) указывают на начало префикса программы PSP.

3) Как определяется стек?

Регистр SS указывает на начало сегмента стека, а в SP находится смещение конца сегмента стека.

4) Как определяется точка входа?

При помощи директивы END, на которой заканчивается выполнение программы. Точка входа обычно указывается в качестве операнда этой директивы.

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 ПРОГРАММЫ

Название файла: lab1com.asm

```

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

; Данные
PC_type_string      db  'PC Type: $'
PC_type_PC          db  'PC',0dh,0ah,'$'
PC_type_PCXT        db  'PC/XT',0dh,0ah,'$'
PC_type_AT          db  'AT',0dh,0ah,'$'
PC_type_PS230       db  'PS2 model 30',0dh,0ah,'$'
PC_type_PS25060     db  'PS2 model 50 or 60',0dh,0ah,'$'
PC_type_PS280       db  'PS2 model 80',0dh,0ah,'$'
PC_type_PCjr        db  'PCjr',0dh,0ah,'$'
PC_type_PCconv       db  'PC Convertible',0dh,0ah,'$'
Unknown_string      db  'Unknown: ',0dh,0ah,'$'
OS_version_string   db  'OS Version: $'
OS_version          db  ' . ',0dh,0ah,'$'
OEM_serial_number_string db  'OEM Serial number: $'
OEM_serial_number   db  ' ',0dh,0ah,'$'
User_serial_number_string db  'User Serial number: $'
User_serial_number  db  ' ',0dh,0ah,'$'

; 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 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 цифра
        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 AL 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 PROC near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT ENDP
;-----

; КОД
BEGIN:
process_type:
    mov bx, 0f000h
    mov es, bx
    mov al, es:[0ffffh]

```

```

        mov     dx, offset PC_type_string
        call    PRINT

        cmp     al, 0ffh
        je      print_type_PC
        cmp     al, 0feh
        je      print_type_PCXT
        cmp     al, 0fbh
        je      print_type_PCXT
        cmp     al, 0fch
        je      print_type_AT
        cmp     al, 0fah
        je      print_type_PS230
        cmp     al, 0fch
        je      print_type_PS25060
        cmp     al, 0f8h
        je      print_type_PS280
        cmp     al, 0fdh
        je      print_type_PCjr
        cmp     al, 0f9h
        je      print_type_PCconv
        jmp     print_type_unknown

print_type_PC:
        mov     dx, offset PC_type_PC
        jmp     print_type
print_type_PCXT:
        mov     dx, offset PC_type_PCXT
        jmp     print_type
print_type_AT:
        mov     dx, offset PC_type_AT
        jmp     print_type
print_type_PS230:
        mov     dx, offset PC_type_PS230
        jmp     print_type
print_type_PS25060:
        mov     dx, offset PC_type_PS25060
        jmp     print_type
print_type_PS280:
        mov     dx, offset PC_type_PS280
        jmp     print_type
print_type_PCjr:
        mov     dx, offset PC_type_PCjr
        jmp     print_type
print_type_PCconv:
        mov     dx, offset PC_type_PCconv
        jmp     print_type
print_type_unknown:
        mov     bx, offset Unknown_string
        call    BYTE_TO_HEX
        mov     [bx+9], al
        mov     [bx+10], ah
        mov     dx, bx
        jmp     print_type
print_type:
        call    PRINT

process_version:

```

```

        mov ah, 30h
        int 21h
print_os_version:
        lea dx, OS_version_string
        call PRINT

        lea si, OS_version + 1
        push ax
        call BYTE_TO_DEC
        pop ax
        mov al, ah
        add si, 4
        call BYTE_TO_DEC
        lea dx, OS_version
        call PRINT
print_oem_number:
        lea dx, OEM_serial_number_string
        call PRINT

        lea si, OEM_serial_number + 2
        mov al, bh
        call BYTE_TO_DEC
        lea dx, OEM_serial_number
        call PRINT
print_serial_number:
        lea dx, User_serial_number_string
        call PRINT

        lea di, User_serial_number
        mov al, bl
        call BYTE_TO_HEX
        mov [di], al
        mov [di+1], ah
        add di, 5
        mov ax, cx
        call WRD_TO_HEX
        lea dx, User_serial_number
        call PRINT
exit:
        mov ah, 4ch
        int 21h
TESTPC ENDS
        END START ; Конец модуля, START - точка входа

```

Название файла: lab1exe.asm

```

.model small
.stack 100h

; Данные
.data
PC_type_string      db 'PC Type: $'
PC_type_PC          db 'PC', 0dh, 0ah, '$'
PC_type_PCXT        db 'PC/XT', 0dh, 0ah, '$'
PC_type_AT           db 'AT', 0dh, 0ah, '$'
PC_type_PS230       db 'PS2 model 30', 0dh, 0ah, '$'

```

PC_type_PS25060	db	'PS2 model 50 or 60',0dh,0ah,'\$'
PC_type_PS280	db	'PS2 model 80',0dh,0ah,'\$'
PC_type_PCjr	db	'PCjr',0dh,0ah,'\$'
PC_type_PCconv	db	'PC Convertible',0dh,0ah,'\$'
Unknown_string	db	'Unknown: ',0dh,0ah,'\$'
OS_version_string	db	'OS Version: \$'
OS_version	db	' . ',0dh,0ah,'\$'
OEM_serial_number_string	db	'OEM Serial number: \$'
OEM_serial_number	db	' ',0dh,0ah,'\$'
User_serial_number_string	db	'User Serial number: \$'
User_serial_number	db	' ',0dh,0ah,'\$'

.code

START: JMP BEGIN

; 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 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 цифра

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 AL 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      PROC      near
    push     ax
    mov      ah, 09h
    int      21h
    pop      ax
    ret
PRINT      ENDP
;-----

; КОД
BEGIN:
    mov      ax, @data
    mov      ds, ax
process_type:
    mov      bx, 0f000h
    mov      es, bx
    mov      al, es:[0ffffh]

    mov      dx, offset PC_type_string
    call     PRINT

    cmp      al, 0ffh
    je       print_type_PC
    cmp      al, 0feh
    je       print_type_PCXT
    cmp      al, 0fbh
    je       print_type_PCXT
    cmp      al, 0fch
    je       print_type_AT
    cmp      al, 0fah
    je       print_type_PS230

```

```

        cmp     al, 0fch
        je      print_type_PS25060
        cmp     al, 0f8h
        je      print_type_PS280
        cmp     al, 0fdh
        je      print_type_PCjr
        cmp     al, 0f9h
        je      print_type_PCconv
        jmp     print_type_unknown

print_type_PC:
        mov     dx, offset PC_type_PC
        jmp     print_type
print_type_PCXT:
        mov     dx, offset PC_type_PCXT
        jmp     print_type
print_type_AT:
        mov     dx, offset PC_type_AT
        jmp     print_type
print_type_PS230:
        mov     dx, offset PC_type_PS230
        jmp     print_type
print_type_PS25060:
        mov     dx, offset PC_type_PS25060
        jmp     print_type
print_type_PS280:
        mov     dx, offset PC_type_PS280
        jmp     print_type
print_type_PCjr:
        mov     dx, offset PC_type_PCjr
        jmp     print_type
print_type_PCconv:
        mov     dx, offset PC_type_PCconv
        jmp     print_type
print_type_unknown:
        mov     bx, offset Unknown_string
        call    BYTE_TO_HEX
        mov     [bx+9], al
        mov     [bx+10], ah
        mov     dx, bx
        jmp     print_type
print_type:
        call    PRINT

process_version:
        mov     ah, 30h
        int     21h
print_os_version:
        lea     dx, OS_version_string
        call    PRINT

        lea     si, OS_version + 1
        push    ax
        call    BYTE_TO_DEC
        pop     ax
        mov     al, ah
        add     si, 4
        call    BYTE_TO_DEC

```



```

        lea     dx, OS_version
        call    PRINT
print_oem_number:
        lea     dx, OEM_serial_number_string
        call    PRINT

        lea     si, OEM_serial_number + 2
        mov     al, bh
        call    BYTE_TO_DEC
        lea     dx, OEM_serial_number
        call    PRINT
print_serial_number:
        lea     dx, User_serial_number_string
        call    PRINT

        lea     di, User_serial_number
        mov     al, bl
        call    BYTE_TO_HEX
        mov     [di], al
        mov     [di+1], ah
        add     di, 5
        mov     ax, cx
        call    WRD_TO_HEX
        lea     dx, User_serial_number
        call    PRINT
exit:
        mov     ah, 4ch
        int     21H

        END START ; Конец модуля, START - точка входа

```