

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. 8382

Щемель Д.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

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 выполнения работы

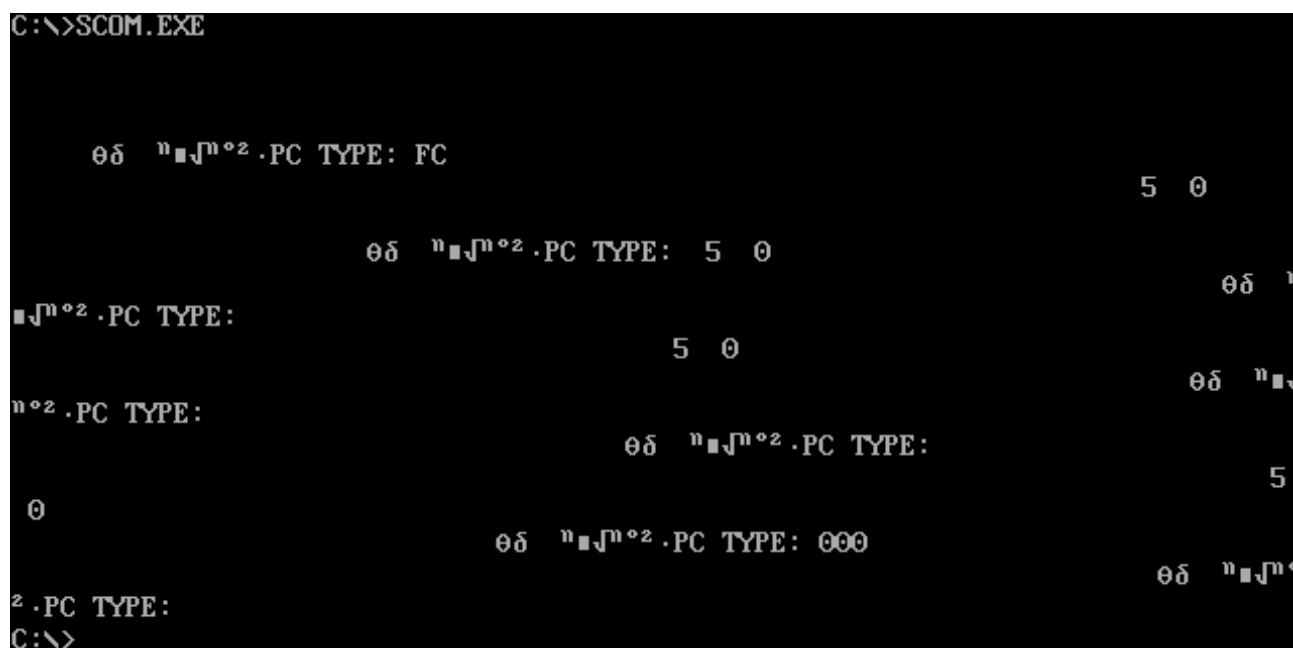
Был написан исходный код для **.COM**-модуля. Результат работы которого приведён на скриншоте ниже.



```
C:\>SCOM.COM
PC TYPE: AT
SYSTEM VERSION: 05.00
OEM: 0
NUMBER: 000
C:\>_
```

Рис. 1: Результат работы **.COM**

Из исходного кода **.COM**-модуля был скомпилирован “плохой” **.EXE**. Результат работы которого приведён на скриншоте ниже.



```
C:\>SCOM.EXE

0δ  "■√"°z .PC TYPE: FC
5 0

0δ  "■√"°z .PC TYPE: 5 0
0δ  "

"■√"°z .PC TYPE:
5 0

0δ  "

"°z .PC TYPE:
0δ  "■√"°z .PC TYPE:
5

0
0δ  "■√"°z .PC TYPE: 000
0δ  "■√"°

z .PC TYPE:
C:\>_
```

Рис. 2: Результат работы “плохого” **.EXE**

Для “хорошего” **.EXE** был написан код, выполняющий те же самые действия, но с учётом особенностей для **.EXE**-модуля. Результат его работы представлен на скриншоте ниже.

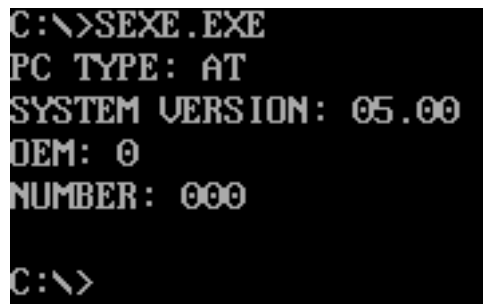


Рис. 3: Результат работы хорошего .EXE

Контрольные вопросы

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

Только один.

2. EXE-программа?

Более одного. Обязательный сегмент - сегмент кода. Опциональными являются сегменты: данных, стэк и др.

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

ORG - задаёт смещение внутри сегмента. (Для того, чтобы в начале в последствии разместился *PSP* без затирания кода программы)

ASSUME - устанавливает соответствие между сегментами и сегментными регистрами.

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

Нет. Нельзя создавать более одного сегмента и использовать адресацию сегментов. т.к. отсутствует таблица настройки адресов (*relocation information*)

Отличия форматов файлов COM и EXE модулей

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

Один сегмент. Адрес начала - 0. Ниже приведён скриншот.

2. Какова структура “плохого” EXE? С какого адреса располагается код? Что располагается с адреса 0?

С адреса 0 начинается заголовок .EXE-программы. Код начинается с 300h(header + offset

```

dmitry@localhost~/D/O/lab1> hexyl SCOM.COM
00000000 e9 c1 00 ff fc fe fb fc f8 fd f9 50 43 0d 0a 24 xx0xxxxx xxxPC__$
00000010 50 43 2f 58 54 0d 0a 24 41 54 0d 0a 24 50 53 32 PC/XT__$AT__$PS2
00000020 20 6d 6f 64 65 6c 20 33 30 0d 0a 24 50 53 32 20 model 3:0__$PS2
00000030 6d 6f 64 65 6c 20 35 30 2f 36 30 0d 0a 24 50 53 model 50:/60__$PS
00000040 32 20 6d 6f 64 65 6c 20 38 30 0d 0a 24 50 43 6a 2 model :80__$PCj
00000050 72 0d 0a 24 50 43 20 43 6f 6e 76 65 72 74 69 62 r__$PC C onvertib
00000060 6c 65 0d 0a 24 30 30 2e 30 30 0d 0a 24 30 0d 0a le__$00.00__$0__
00000070 24 30 30 30 0d 0a 24 32 c0 b4 4c cd 21 c3 50 b4 $000__$2:xxLx!*Px
00000080 09 cd 21 58 c3 24 0f 3c 09 76 02 04 07 04 30 c3 _x!Xx$<_v...0x
00000090 51 8a e0 e8 ef ff 86 c4 b1 04 d2 e8 e8 e6 ff 59 QxxxxxxxxxxxxxY
000000a0 c3 51 52 32 e4 33 d2 b9 0a 00 f7 f1 80 ca 30 88 xQR2x3xx_0xxxx0x
000000b0 14 4e 33 d2 3d 0a 00 73 f1 3c 00 74 04 0c 30 88 .N3x=_0s x<0t_0x
000000c0 04 5a 59 c3 b8 00 f0 8e c0 26 a0 fe ff 3a 06 03 .ZYxx0xx x&xxx:..
000000d0 01 74 33 3a 06 05 01 74 33 3a 06 06 01 74 2d 3a .t3:...t 3:...t-:
000000e0 06 04 01 74 2d 3a 06 07 01 74 2d 3a 06 08 01 74 ...t-:...t-:...t
000000f0 2d 3a 06 3e 01 74 2d 3a 06 09 01 74 2d 3a 06 0a -:>.t-:_.t-:._
00000100 01 74 2d eb 31 90 ba 0b 01 eb 3b 90 ba 10 01 eb .t-x1xx. .x;xxx.x
00000110 35 90 ba 18 01 eb 2f 90 ba 1d 01 eb 29 90 ba 2c 5xxx.x/x .xxx)xx,
00000120 01 eb 23 90 ba 3e 01 eb 1d 90 ba 4d 01 eb 17 90 .x#xx>.x .xxM.x.x
00000130 ba 54 01 eb 11 90 e8 57 ff 8b d0 b4 02 cd 21 86 xT.x.x.xW xxxxx.x!x
00000140 d6 cd 21 eb 04 90 e8 35 ff b4 30 cd 21 8b d0 be xx!x.x.x5 xx0x!xxx
00000150 65 01 46 e8 4b ff be 65 01 83 c6 04 8a c6 e8 40 e.FxKxXe .xx.xxxx@
00000160 ff ba 65 01 e8 17 ff be 6d 01 8a c7 ba 6d 01 e8 xxe.x.x.x m.xxxxm.x
00000170 0c ff be 71 01 8a c3 e8 27 ff be 71 01 46 8a c1 _xxq.xxxx 'xxq.Fxx
00000180 e8 1e ff be 71 01 83 c6 02 8a c5 e8 13 ff ba 71 x.x.xq.xx .xxx.x.xq
00000190 01 e8 ea fe e8 e0 fe .xxxxxx

```

Рис. 4: Структура .COM

+ offset (повторно, со стороны OS)). См скриншот ниже.

```
dmitry@localhost~/.D/O/lab1> hexyl SCOM.EXE
00000000 4d 5a 97 00 03 00 00 00 20 00 00 00 ff ff 00 00 MZ×0·000 000××00
00000010 00 00 26 28 00 01 00 00 1e 00 00 00 01 00 00 00 00&(0·00 ·000·000
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00000000 00000000
*
00000300 e9 c1 00 ff fc fe fb fc f8 fd f9 50 43 0d 0a 24 ××0××××× ×××PC__$
00000310 50 43 2f 58 54 0d 0a 24 41 54 0d 0a 24 50 53 32 PC/XT__$ AT__$PS2
00000320 20 6d 6f 64 65 6c 20 33 30 0d 0a 24 50 53 32 20 model 3:0__$PS2
00000330 6d 6f 64 65 6c 20 35 30 2f 36 30 0d 0a 24 50 53 model 50:/60__$PS
00000340 32 20 6d 6f 64 65 6c 20 38 30 0d 0a 24 50 43 6a 2 model 80__$PCj
00000350 72 0d 0a 24 50 43 20 43 6f 6e 76 65 72 74 69 62 r__$PC C onvertib
00000360 6c 65 0d 0a 24 30 30 2e 30 30 0d 0a 24 30 0d 0a le__$00. 00__$0__
00000370 24 30 30 30 0d 0a 24 32 c0 b4 4c cd 21 c3 50 b4 $000__$2 ××L×!×P×
00000380 09 cd 21 58 c3 24 0f 3c 09 76 02 04 07 04 30 c3 _×!X×$< _v····0×
00000390 51 8a e0 e8 ef ff 86 c4 b1 04 d2 e8 e8 e6 ff 59 Q××××××× x·×××××Y
000003a0 c3 51 52 32 e4 33 d2 b9 0a 00 f7 f1 80 ca 30 88 ×QR2×3×× _0××××0×
000003b0 14 4e 33 d2 3d 0a 00 73 f1 3c 00 74 04 0c 30 88 ·N3×=_0s ×<0t·_0×
000003c0 04 5a 59 c3 b8 00 f0 8e c0 26 a0 fe ff 3a 06 03 ·ZY××0×× ×&×××:··
000003d0 01 74 33 3a 06 05 01 74 33 3a 06 06 01 74 2d 3a ·t3:···t 3:···t-:
000003e0 06 04 01 74 2d 3a 06 07 01 74 2d 3a 06 08 01 74 ···t-:··· ·t-:···t
000003f0 2d 3a 06 3e 01 74 2d 3a 06 09 01 74 2d 3a 06 0a -:·>·t-: ·_·t-:·_
00000400 01 74 2d eb 31 90 ba 0b 01 eb 3b 90 ba 10 01 eb ·t-×1××· ·x;××··×
00000410 35 90 ba 18 01 eb 2f 90 ba 1d 01 eb 29 90 ba 2c 5××··×/× ×··×)××,
00000420 01 eb 23 90 ba 3e 01 eb 1d 90 ba 4d 01 eb 17 90 ·×#×>·× ·××M·×·×
00000430 ba 54 01 eb 11 90 e8 57 ff 8b d0 b4 02 cd 21 86 ×T·×·××W ××××·×!×
00000440 d6 cd 21 eb 04 90 e8 35 ff b4 30 cd 21 8b d0 be ××!×·××5 ××0×!×××
00000450 65 01 46 e8 4b ff be 65 01 83 c6 04 8a c6 e8 40 e·F×K××e ·××·×××@
00000460 ff ba 65 01 e8 17 ff be 6d 01 8a c7 ba 6d 01 e8 ××e·×·×× m·×××m·×
00000470 0c ff be 71 01 8a c3 e8 27 ff be 71 01 46 8a c1 _××q·××× '××q·F××
00000480 e8 1e ff be 71 01 83 c6 02 8a c5 e8 13 ff ba 71 x·××q·×× ·×××·××q
00000490 01 e8 ea fe e8 e0 fe ·××××××
dmitry@localhost~/.D/O/lab1> 
```

Рис. 5: Структура плохого .EXE-файла

3. Какова структура “хорошего” EXE? Чем он отличается от файла “плохого EXE”?

С 200h(header + offset (1 раз со стороны OS)) начинается стэк. С 220h начинается сегмент данных. С 2a0h начинается сегмент кода. Оличается от “плохого” EXE наличием более одного сегмента и стэка. Порядок сегментов может меняться. Так же, в “плохом” EXE устанавливается смещение на 100h, чего нет в “хорошем” EXE, а потому и разница между началом стэка и кода в “хорошем” и “плохом” EXE соотетственно составляет 100h.

Загрузка COM модуля в основную память

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

```

dmitry@localhost~/D/O/lab1> hexyl SEXE.EXE
00000000 4d 5a c5 01 02 00 01 00 20 00 00 00 ff ff 00 00 MZ×××0×0 000××00
00000010 20 00 3b 3b 4d 00 0a 00 1e 00 00 00 01 00 4e 00 0;M0_0×000×0N0
00000020 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 _00000000 00000000
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00000000 00000000
*
00000220 ff fc fe fb fc f8 fd f9 50 43 0d 0a 24 50 43 2f ×××××××× PC__$PC/
00000230 58 54 0d 0a 24 41 54 0d 0a 24 50 53 32 20 6d 6f XT__$AT__$SPS2 mo
00000240 64 65 6c 20 33 30 0d 0a 24 50 53 32 20 6d 6f 64 del 30__$SPS2 mod
00000250 65 6c 20 35 30 2f 36 30 0d 0a 24 50 53 32 20 6d el 50/60__$SPS2 m
00000260 6f 64 65 6c 20 38 30 0d 0a 24 50 43 6a 72 0d 0a odel 80__$PCjr__
00000270 24 50 43 20 43 6f 6e 76 65 72 74 69 62 6c 65 0d $PC Convertible_
00000280 0a 24 30 30 2e 30 30 0d 0a 24 30 0d 0a 24 30 30 _$00.00__$0__$00
00000290 30 0d 0a 24 00 00 00 00 00 00 00 00 00 00 00 00 0__$0000 00000000
000002a0 32 c0 b4 4c cd 21 c3 50 b4 09 cd 21 58 c3 24 0f 2××L×!×P×××××××
000002b0 3c 09 76 02 04 07 04 30 c3 51 8a e0 e8 ef ff 86 <_v×××××××××××××
000002c0 c4 b1 04 d2 e8 e8 e6 ff 59 c3 51 52 32 e4 33 d2 ××××××××××Y×QR2×3×
000002d0 b9 0a 00 f7 f1 80 ca 30 88 14 4e 33 d2 3d 0a 00 ×_0××××××××××N3×=_0
000002e0 73 f1 3c 00 74 04 0c 30 88 04 5a 59 c3 b8 02 00 s×<0t×_0××ZY×××0
000002f0 8e d8 b8 00 f0 8e c0 26 a0 fe ff 3a 06 00 00 74 ×××0×××8×××××;00t
00000300 33 3a 06 02 00 74 33 3a 06 03 00 74 2d 3a 06 01 3:××0t3:××0t-:××
00000310 00 74 2d 3a 06 04 00 74 2d 3a 06 05 00 74 2d 3a 0t-:××0t-:××0t-:××0t-:
00000320 06 3b 00 74 2d 3a 06 06 00 74 2d 3a 06 07 00 74 ;0t-:××0t-:××0t-:××0t
00000330 2d eb 31 90 ba 08 00 eb 3b 90 ba 0d 00 eb 35 90 -×1×××0××;××_0×5×
00000340 ba 15 00 eb 2f 90 ba 1a 00 eb 29 90 ba 29 00 eb ××0×/××××0×)××)0×
00000350 23 90 ba 3b 00 eb 1d 90 ba 4a 00 eb 17 90 ba 51 #××;0××××J0××××Q
00000360 00 eb 11 90 e8 52 ff 8b d0 b4 02 cd 21 86 d6 cd 0××××R××××××!×××
00000370 21 eb 04 90 e8 30 ff b4 30 cd 21 8b d0 be 62 00 !×××0×××0×!×××b0
00000380 46 e8 46 ff be 62 00 83 c6 04 8a c6 e8 3b ff ba F×F××b0×××××;××
00000390 62 00 e8 12 ff be 6a 00 8a c7 ba 6a 00 e8 07 ff b0××××j0××××j0×××
000003a0 be 6e 00 8a c3 e8 22 ff be 6e 00 46 8a c1 e8 19 ×n0×××"××n0F××××
000003b0 ff be 6e 00 83 c6 02 8a c5 e8 0e ff ba 6e 00 e8 ××n0××××××××n0×
000003c0 e5 fe e8 db fe ×××××
dmitry@localhost~/D/O/lab1>

```

Рис. 6: Структура хорошего .EXE-файла

PSP - размером 100h байт. После чего с 100h начинается код.

2. Что располагается с адреса 0?

PSP - системный сегмент с адресом выхода, информацией о работе предыдущей программы, параметры, с которыми была запущена текущая программа (переменные среды, аргументы командой строки).

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

Сегментные значения указывают на адрес 19F5 - начало *PSP*. См скриншот ниже.

CS	19F5	IP	0100
DS	19F5		
ES	19F5	HS	19F5
SS	19F5	FS	19F5

Рис. 7: Системные регистры

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

Начальное значение *SP* - FFFEH. Это самое начало сегмента памяти. Расширение стэка происходит вниз без ограничений.

Загрузка “хорошего” EXE модуля в основную память

1. Как загружается “хороший” EXE? Какие значения имеют сегментные регистры?

В начало добавляется *PSP*. *CS* указывает на 1A0F - адрес сегмента кода. *SS* на 1A05 - адрес начала сегмента стэка. Остальные сегменты указывают на 19F5 - начало *PSP*. См скриншот ниже.

CS	1A0F	IP	004D
DS	19F5		
ES	19F5	HS	19F5
SS	1A05	FS	19F5

Рис. 8: Сегментные регистры

2. На что указывают регистры DS и ES?

PSP

3. Как определяется стек?

Размер стека задаётся программно. При запуске программы SP указывает на *размер стека*.

4. Как определяется точка входа?

END , либо с начала сегмента кода, при отсутствии первого выражения.

Вывод

В ходе выполнения лабораторной работы были получены практические навыки написания кода для .COM-модулей и .EXE на примере программы для получения сведений о машине, на которой они будут запущены. Были выявлены и изучены различия между .COM и .EXE модулями.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ SCOM

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

PC db 0ffh

AT db 0fch

PCXT_1 db 0feh

PCXT_2 db 0fbh

PS230 db 0fch

PS25060 db 0f8h

PCjr db 0fdh

PCConv db 0f9h

PC_TYPE_STRING db 'PC TYPE: \$'

SYSTEM_VERSION_STRING db 'SYSTEM VERSION: \$'

OEM_STRING_LABEL db 'OEM: \$'

NUMBER_STRING_LABEL db 'NUMBER: \$'

PC_STRING db 'PC', 13, 10, '\$'

PC_XT_STRING db 'PC/XT', 13, 10, '\$'

AT_STRING db 'AT', 13, 10, '\$'

PS230_STRING db 'PS2 model 30', 13, 10, '\$'

PS25060_STRING db 'PS2 model 50/60', 13, 10, '\$'

PS280_STRING db 'PS2 model 80', 13, 10, '\$'

PCjr_STRING db 'PCjr', 13, 10, '\$'

PCConv_STRING db 'PC Convertible', 13, 10, '\$'

VERSION_STRING db '00.00', 13, 10, '\$'

OEM_STRING db '0', 13, 10, '\$'

NUMBER_STRING db '000', 13, 10, '\$'

EXIT PROC near

xor AL, AL

```

        mov AH, 4ch
        int 21h
        ret
EXIT ENDP

PRINT PROC near
        push ax
        mov ah, 09h
        int 21h
        pop ax
        ret
PRINT 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 near
        push cx
        mov ah, al
        call TETR_TO_HEX
        xchg al, ah
        mov cl, 4
        shr al, cl
        call TETR_TO_HEX
        pop cx
        ret
BYTE_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

BEGIN:

```
    mov ax, 0f000h
    mov es, ax
    mov al, es:[0fffh]

    mov dx, offset PC_TYPE_STRING
    call PRINT

    cmp al, PC
```

```

je print_PC
cmp al, PCXT_1
je print_PC_XT
cmp al, PCXT_2
je print_PC_XT
cmp al, AT
je print_AT
cmp al, PS230
je print_PS2_30
cmp al, PS25060
je print_PS2_50_or_60
cmp al, PS280_STRING
je print_PS2_80
cmp al, PCjr
je print_PCjr
cmp al, PCConv
je print_PCConv

jmp print_unknown

print_PC:
    mov dx, offset PC_STRING
    jmp finish
print_PC_XT:
    mov dx, offset PC_XT_STRING
    jmp finish
print_AT:
    mov dx, offset AT_STRING
    jmp finish
print_PS2_30:
    mov dx, offset PS230_STRING
    jmp finish
print_PS2_50_or_60:
    mov dx, offset PS25060_STRING

```

```

        jmp finish
print_PS2_80:
        mov dx, offset PS280_STRING
        jmp finish
print_PCjr:
        mov dx, offset PCjr_STRING
        jmp finish
print_PCConv:
        mov dx, offset PCConv_STRING
        jmp finish
print_unknown:
        call BYTE_TO_HEX
        mov dx, ax
        mov ah, 02h
        int 21h
        xchg dl, dh
        int 21h
        jmp skip
finish:
        call PRINT
skip:

mov ah, 30h
int 21h
mov dx, ax
mov si, offset VERSION_STRING
inc si
call BYTE_TO_DEC
mov si, offset VERSION_STRING
add si, 4
mov al, dh
call BYTE_TO_DEC
mov dx, offset SYSTEM_VERSION_STRING
call PRINT

```

```

    mov dx, offset VERSION_STRING
    call PRINT
    mov dx, offset OEM_STRING_LABEL
    call PRINT
    mov si, offset OEM_STRING
    mov al, bh
    mov dx, offset OEM_STRING
    call PRINT
    mov dx, offset NUMBER_STRING_LABEL
    call PRINT
    mov si, offset NUMBER_STRING
    mov al, bl
    call BYTE_TO_DEC
    mov si, offset NUMBER_STRING
    inc si
    mov al, cl
    call BYTE_TO_DEC
    mov si, offset NUMBER_STRING
    add si, 2
    mov al, ch
    call BYTE_TO_DEC
    mov dx, offset NUMBER_STRING
    call PRINT

    call EXIT
TESTPC ENDS
END START

```

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ *SEXE*

```
_STACK SEGMENT STACK
```

```
    DW 10h DUP(0)
```

```
_STACK ENDS
```

```
DATA SEGMENT
```

```
PC db 0ffh
```

```
AT db 0fch
```

```
PCXT_1 db 0feh
```

```
PCXT_2 db 0fbh
```

```
PS230 db 0fch
```

```
PS25060 db 0f8h
```

```
PCjr db 0fdh
```

```
PCConv db 0f9h
```

```
PC_TYPE_STRING db 'PC TYPE: $'
```

```
SYSTEM_VERSION_STRING db 'SYSTEM VERSION: $'
```

```
OEM_STRING_LABEL db 'OEM: $'
```

```
NUMBER_STRING_LABEL db 'NUMBER: $'
```

```
PC_STRING db 'PC', 13, 10, '$'
```

```
PC_XT_STRING db 'PC/XT', 13, 10, '$'
```

```
AT_STRING db 'AT', 13, 10, '$'
```

```
PS230_STRING db 'PS2 model 30', 13, 10, '$'
```

```
PS25060_STRING db 'PS2 model 50/60', 13, 10, '$'
```

```
PS280_STRING db 'PS2 model 80', 13, 10, '$'
```

```
PCjr_STRING db 'PCjr', 13, 10, '$'
```

```
PCConv_STRING db 'PC Convertible', 13, 10, '$'
```

```
VERSION_STRING db '00.00', 13, 10, '$'
```

```
OEM_STRING db '0', 13, 10, '$'
```

```
NUMBER_STRING db '000', 13, 10, '$'
```

```
DATA ENDS
```

```
CODE SEGMENT
```

```
ASSUME CS:CODE, DS:DATA, SS:_STACK
```

```
EXIT PROC near
```

```
    xor AL, AL
```

```
    mov AH, 4ch
```

```
    int 21h
```

```
    ret
```

```
EXIT ENDP
```

```
PRINT PROC near
```

```
    push ax
```

```
    mov ah, 09h
```

```
    int 21h
```

```
    pop ax
```

```
    ret
```

```
PRINT 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 near
```

```
    push cx
```

```
    mov ah, al
```

```
    call TETR_TO_HEX
```

```
    xchg al, ah
```

```
    mov cl, 4
```

```
    shr al, cl
```



```

        call TETR_TO_HEX
        pop cx
        ret
BYTE_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

MAIN:
        mov ax, DATA
        mov ds, ax

        mov ax, 0f000h

```

```

mov es, ax
mov al, es:[0fffeh]

mov dx, offset PC_TYPE_STRING
call PRINT

cmp al, PC
je print_PC
cmp al, PCXT_1
je print_PC_XT
cmp al, PCXT_2
je print_PC_XT
cmp al, AT
je print_AT
cmp al, PS230
je print_PS2_30
cmp al, PS25060
je print_PS2_50_or_60
cmp al, PS280_STRING
je print_PS2_80
cmp al, PCjr
je print_PCjr
cmp al, PCConv
je print_PCConv

jmp print_unknown

print_PC:
    mov dx, offset PC_STRING
    jmp finish
print_PC_XT:
    mov dx, offset PC_XT_STRING
    jmp finish
print_AT:

```

```

        mov dx, offset AT_STRING
        jmp finish
print_PS2_30:
        mov dx, offset PS230_STRING
        jmp finish
print_PS2_50_or_60:
        mov dx, offset PS25060_STRING
        jmp finish
print_PS2_80:
        mov dx, offset PS280_STRING
        jmp finish
print_PCjr:
        mov dx, offset PCjr_STRING
        jmp finish
print_PCConv:
        mov dx, offset PCConv_STRING
        jmp finish
print_unknown:
        call BYTE_TO_HEX
        mov dx, ax
        mov ah, 02h
        int 21h
        xchg dl, dh
        int 21h
        jmp skip
finish:
        call PRINT
skip:

mov ah, 30h
int 21h
mov dx, ax
mov si, offset VERSION_STRING
inc si

```

```

call BYTE_TO_DEC
mov si, offset VERSION_STRING
add si, 4
mov al, dh
call BYTE_TO_DEC
mov dx, offset SYSTEM_VERSION_STRING
call PRINT
mov dx, offset VERSION_STRING
call PRINT
mov dx, offset OEM_STRING_LABEL
call PRINT
mov si, offset OEM_STRING
mov al, bh
mov dx, offset OEM_STRING
call PRINT
mov dx, offset NUMBER_STRING_LABEL
call PRINT
mov si, offset NUMBER_STRING
mov al, bl
call BYTE_TO_DEC
mov si, offset NUMBER_STRING
inc si
mov al, cl
call BYTE_TO_DEC
mov si, offset NUMBER_STRING
add si, 2
mov al, ch
call BYTE_TO_DEC
mov dx, offset NUMBER_STRING
call PRINT

call EXIT
CODE ENDS
END MAIN

```