

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. 8383

Шишкин И.В.

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 работы.

Был написан исходный .COM модуль, который определяет тип РС и версию системы. В результате был получен "хороший" .COM модуль, результат выполнения которого представлен на рис. 1.

```
C:\>tlink os1_com.obj/t
Turbo Link Version 5.1 Copyright (c) 1992 Borland International

C:\>os1_com.com

System version: 5.0
OEM serial number: 255
User serial number: 000000
PC type: AT
```

Рисунок 1 – Результат выполнения "хорошего" .COM модуля

Был построен "плохой" .EXE файл из исходного текста для .COM модуля. Результат выполнения "плохого" .EXE файла представлен на рис. 2. Как видно на рисунке, после линковки, выдано предупреждение "No stack".

```
Warning: No stack
C:\>os1_com.exe

PC type: 5 0
PC type: 0x8
PC type: 5 0
PC type: 255
PC type: 5 0
PC type: 0x8
PC type: 000000 5 0
PC type: 0x8
PC type: 000000 5 0
PC type: 0x8
PC type: 255 000000 5 0
PC type: 0x8
```

Рисунок 2 – Результат выполнения "плохого" .EXE модуля

Был написан текст исходного .EXE модуля, который выполняет те же функции, что и .COM модуль. Таким образом был получен "хороший" .EXE модуль. Результат его выполнения представлен на рис. 3.

```
C:\>os1_exe.exe

System version: 5.0
OEM serial number: 255
User serial number: 000000
PC type: AT
```

Рисунок 3 – Результат выполнения "хорошего" .EXE модуля

Была запущена программа Far manager, и в ней открыты .COM файл (рис. 4), "плохой" .EXE файл (рис. 5) и "хороший" .EXE файл (рис.6) в шестнадцатеричном виде.



Рисунок 4 – .COM файл в FAR

0000000000:	4D 5A 15 01 03 00 00 00	20 00 00 00 FF FF 00 00	MZSE♥	яя
0000000010:	00 00 00 00 00 01 00 00	3E 00 00 00 01 00 FB 50	☺ > ☺ мР	
0000000020:	6A 72 00 00 00 00 00 00	00 00 00 00 00 00 00 00	jr	
0000000030:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000040:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000050:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000060:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000070:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000080:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000090:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
00000000A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
00000000B0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
00000000C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
00000000D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
00000000E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
00000000F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000100:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000110:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000120:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000130:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000140:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000150:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000160:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000170:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000180:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
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:	E9 06 02 0D 0A 50 43 20	74 79 70 65 3A 20 24 50	й♣JPC type: \$P	
0000000310:	43 24 50 43 2F 58 54 24	41 54 24 50 53 32 20 6D	C\$PC/XI\$AT\$PS2 m	
0000000320:	6F 64 65 6C 20 33 30 24	50 53 32 20 6D 6F 64 65	odel 30\$PS2 mode	
0000000330:	6C 20 38 30 24 50 43 6A	72 24 50 53 32 20 6D 6F	l 80\$PCjr\$PS2 mo	
0000000340:	64 65 6C 20 35 30 20 6F	72 20 36 30 24 50 43 20	del 50 or 60\$PC	
0000000350:	43 6F 6E 76 65 72 74 69	62 6C 65 24 55 6E 6B 6E	Convertible\$Unkn	
0000000360:	6F 77 6E 20 24 20 20 20	24 0D 0A 53 79 73 74 65	own \$ \$JCSyste	
0000000370:	6D 20 76 65 72 73 69 6F	6E 3A 20 24 3C 32 2E 30	m version: \$<2.0	
0000000380:	24 0D 0A 4F 45 4D 20 73	65 72 69 61 6C 20 6E 75	\$JQDEM serial nu	
0000000390:	6D 62 65 72 3A 20 24 20	20 20 20 20 20 24 20 20	mber: \$ \$	
00000003A0:	20 20 20 20 20 20 20 20	20 20 24 20 2E 20 24 0D	\$. \$P	
00000003B0:	0A 55 73 65 72 20 73 65	72 69 61 6C 20 6E 75 6D	QUser serial num	
00000003C0:	62 65 72 3A 20 24 24 0F	3C 09 76 02 04 07 04 30	ber: \$\$ж<Cu♣♦♦0	
00000003D0:	C3 51 8A E0 E8 EF FF 86	C4 B1 04 D2 E8 E8 E6 FF	ГQ?анпя+Д♦+Иижа	
00000003E0:	59 C3 53 8A FC E8 E9 FF	88 25 4F 88 05 4F 8A C7	YGS?ыйия?%0?%0?3	
00000003F0:	E8 DE FF 88 25 4F 88 05	5B C3 51 52 32 E4 33 D2	иЮя?%0?%IGQR2л3I	
0000000400:	B9 0A 00 F7 F1 80 CA 30	88 14 4E 33 D2 3D 0A 00	ИQ чс?K0?4N3I=Q	
0000000410:	73 F1 3C 00 74 04 0C 30	88 04 5A 59 C3 50 B4 09	sc< t♦%0?♦ZYGP?o	
0000000420:	CD 21 58 C3 50 52 B4 30	CD 21 50 BA 69 01 E8 EC	H!XГPR?OH!PeiOм	
0000000430:	FF 3C 00 74 50 BE AB 01	E8 BF FF 58 8A C4 83 C6	я< tP?<OмiаX?Д?Ж	
0000000440:	03 E8 B6 FF BA AB 01 E8	D3 FF BA 81 01 E8 CD FF	♥иЧяе<OмЧяе?OмНя	
0000000450:	BE 97 01 83 C6 02 8A C7	E8 9F FF BA 97 01 E8 BC	?-O?ЖO?3и?яе-Oм?	
0000000460:	FF BA AF 01 E8 B6 FF BF	9E 01 83 C7 06 8B C1 E8	яеI OмЧяI?O?3♦<Би	
0000000470:	70 FF 8A C3 E8 5A FF 83	EF 02 89 05 BA 9E 01 E8	ря?ГиЗя?пO?%е?Oм	
0000000480:	9B FF EB 06 90 BA 7C 01	EB BD 5A 58 C3 50 52 B8	>ял♦?е!Oл?ZXГPRё	
0000000490:	00 F0 8E C0 26 A0 FE FF	BA 03 01 E8 7F FF 3C FF	р?А& няе♥Oмдя<я	
00000004A0:	74 34 3C FE 74 36 3C FB	74 32 3C FC 74 34 3C FA	t4<wt6<wt2<ьt4<ь	
00000004B0:	74 36 3C FC 74 38 3C F8	74 3A 3C FD 74 3C 3C F9	t6<ьt8<wt:<эт<<щ	
00000004C0:	74 3E BF 65 01 E8 09 FF	89 05 BA 65 01 E8 4D FF	t>IeOмOя%2eOмMя	
00000004D0:	BA 5C 01 EB 2E 90 BA 0F	01 EB 28 90 BA 12 01 EB	е\Oл.?сжOл(?е!Oл	
00000004E0:	22 90 BA 18 01 EB 1C 90	BA 1B 01 EB 16 90 BA 3A	"?е!Oл-?е+Oл-?е:	
00000004F0:	01 EB 10 90 BA 28 01 EB	0A 90 BA 35 01 EB 04 90	еOм?е<OлO?е5Oл♦?	
0000000500:	BA 4D 01 E8 17 FF 5A 58	C3 E8 18 FF E8 7E FF 32	еM Oм!яZXИтияи~я2	
0000000510:	C0 B4 4C CD 21		A?LH!	

Рисунок 5 – "Плохой" .EXE файл в FAR



Рисунок 6 – "Хороший" .EXE файл в FAR

Был загружен .COM файл и "хороший" .EXE в отладчик TD.EXE. Соответствующие результаты представлены на рис. 7 и рис. 8.

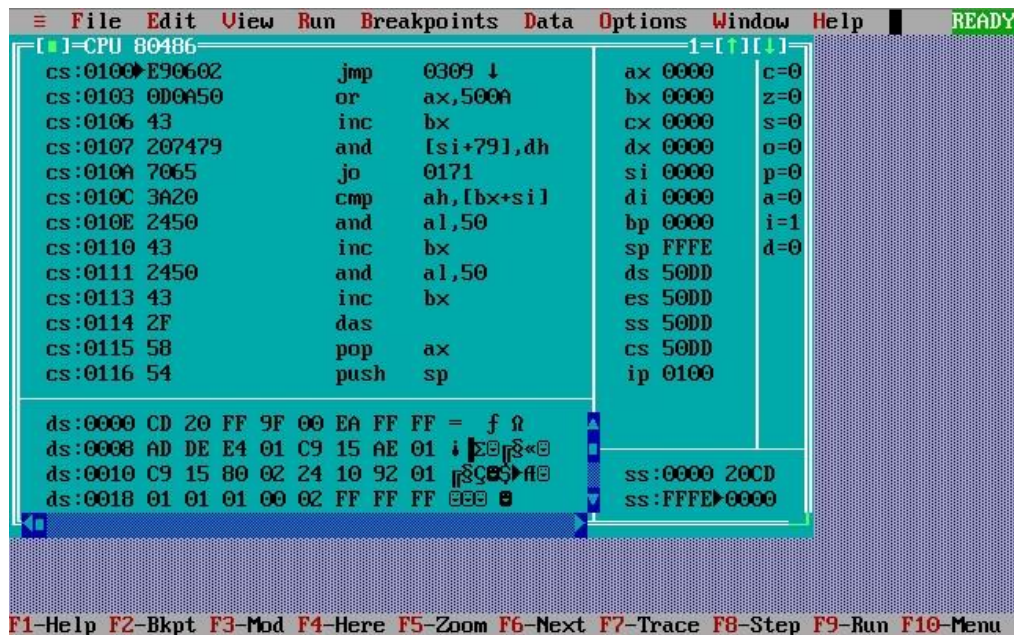


Рисунок 7 – .COM файл в TD.EXE

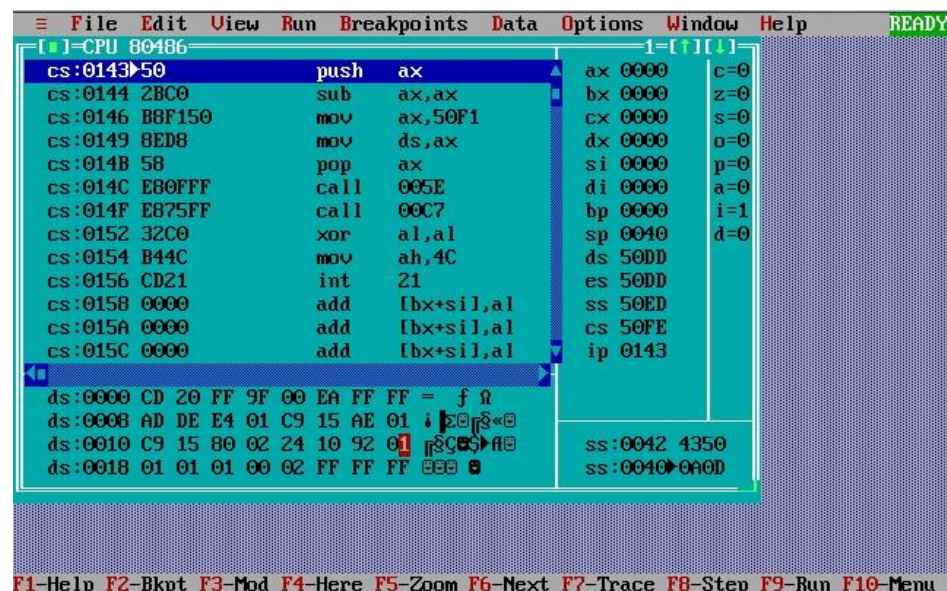


Рисунок 8 – "Хороший" .EXE файл в TD.EXE

Все нужные файлы представлены в приложении.

Контрольные вопросы.

Отличия исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM-программа?

Один. В COM-программе код и данные находятся в одном сегменте, задавать сегмент стека не нужно, так как для COM-программы стек выделяется операционной системой в конце сегмента программы.

2) Сколько сегментов должна содержать EXE-программа?

Количество допустимых сегментов определяется используемой моделью памяти. Сегментов должно быть не менее одного. Верхнего порога нет. Например, модель huge имеет много сегментов кода и сегментов данных.

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

Так как адресация начинается со смещения 100h от начала PSP, COM-программа должна содержать директиву `org 100h`. Должна быть в конце программы директива `END`. Так же должна быть директива `assume`, т.к. эта строка указывает Ассемблеру на привязку сегментного регистра CS к нашему сегменту (CSEG). Пример ошибки, если закомментировать `assume`, и залинковать программу с помощью TLINK, приведена на рис. 9.

```

**Error** os1_com.asm(171) Near jump or call to different CS
**Error** os1_com.asm(173) Near jump or call to different CS
**Error** os1_com.asm(175) Near jump or call to different CS
**Error** os1_com.asm(177) Near jump or call to different CS
**Error** os1_com.asm(179) Near jump or call to different CS
**Error** os1_com.asm(183) Near jump or call to different CS
**Error** os1_com.asm(186) Near jump or call to different CS
**Error** os1_com.asm(188) Near jump or call to different CS
**Error** os1_com.asm(192) Near jump or call to different CS
**Error** os1_com.asm(195) Near jump or call to different CS
**Error** os1_com.asm(198) Near jump or call to different CS
**Error** os1_com.asm(201) Near jump or call to different CS
**Error** os1_com.asm(204) Near jump or call to different CS
**Error** os1_com.asm(207) Near jump or call to different CS
**Error** os1_com.asm(210) Near jump or call to different CS
**Error** os1_com.asm(215) Near jump or call to different CS
**Error** os1_com.asm(225) Near jump or call to different CS
**Error** os1_com.asm(226) Near jump or call to different CS
Error messages: 45
Warning messages: None
Passes: 1
Remaining memory: 470k
```

Рисунок 9 – Ошибка при линковке с помощью TLINK.EXE

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

Нет. Нельзя использовать команды с оператором `seg`, т.к. в COM-программе нет заголовка.

Отличия форматов файлов COM и EXE модулей.

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

В файлах данного типа, обычно не имеющими даже заголовка файла, содержатся только машинный код и данные программы. Код располагается с адреса 100h, т.к. в начале программы происходит смещение.

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

Код и данные содержатся в одном сегменте. Исходя из рис. 5, "плохой" EXE файл начинается с адреса 300h. С адреса 0 располагается заголовок и таблица настроек.

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

Код, данные и стек содержатся в разных сегментах. Так как "плохой" EXE файл получен из исходного текста для COM модуля, то стек, в отличие от "хорошего" EXE, явно не определен. Еще одно отличие в том, что код и данные у "плохого" EXE содержатся в одном сегменте, как было сказано выше. Так же "хороший" EXE начинается с 240h, т.к. 200h для заголовка, а 40h для стека. В "плохом" EXE те же самые 200h для заголовка, а 100h из-за директивы org 100h.

Загрузка COM модуля в основную память.

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

Команды, данные, PSP и стек располагаются в одном сегменте. Код располагается с адреса 100h.

2) Что располагается с адреса 0?

С адреса 0 располагается PSP.

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

Во время загрузки COM-программы выделяется первый свободный сегмент памяти и в его начале размещается PSP. Все сегментные регистры устанавливаются на этот сегмент. Регистры CS и SS указывают на PSP, регистр SP - на конец сегмента PSP (т.е. SP = 0FFFEh).

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

Сегмент стека задавать не нужно, т.к. он выделяется ОС в конце сегмента программы. Все 64 Кбайт занимают данные, коды команд и область стека. В стек задается число 00h. Указатель стека SP устанавливается на конец сегмента программы.

Загрузка "хорошего" EXE модуля в основную память.

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

Во время загрузки "хорошей" EXE-программы выделяется сегмент для PSP. Затем следует сегмент программы. На основании информации в заголовке EXE-программы загрузчик пересчитывает дальние ссылки с учетом реального расположения сегментов. Таким образом, программа заранее не знает, в каких сегментах она будет выполняться. Поэтому для обращения к данным в начале EXE-программы необходимо загрузить в регистр DS значение этого указателя

2) На что указывают регистры DS и ES?

На PSP.

3) Как определяется стек?

EXE-программа по умолчанию записывает нулевое слово в стек. Его можно определить вручную директивой `stack`.

4) Как определяется точка входа?

По умолчанию точкой входа является начало сегмента кода. Задать точку входа можно, написав в конце программы `"end <точка входа>"`.

Выводы.

В ходе выполнения лабораторной работы были исследованы различия в исходных текстах модулей типов .COM и .EXE. Были разобраны структуры файлов загрузочных модулей и способы их загрузки в основную память.

ПРИЛОЖЕНИЕ А

КОД .COM МОДУЛЯ

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

PC_TYPE db 13, 10, "PC type: \$"

TYPE_PC db "PC\$"

TYPE_PCXT db "PC/XT\$"

TYPE_AT db "AT\$"

TYPE_30 db "PS2 model 30\$"

TYPE_80 db "PS2 model 80\$"

TYPE_PCJR db "PCjr\$"

TYPE_5060 db "PS2 model 50 or 60\$"

TYPE_CONVERTIBLE db "PC Convertible\$"

TYPE_UNKNOWN db "Unknown \$"

UNKNOWN_STR db " \$"

SYSTEM_VERSION db 13,10,"System version: \$"

IF_AL0 db "<2.0\$"

OEM_SN db 13, 10, "OEM serial number: \$"

STR_OEM db " \$"

STR_USER db " \$"

SYSVER_STR db " . \$"

USER_SN db 13, 10, "User serial number: \$"

;-----

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, в 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_1

or AL,30h

mov [SI],AL

end_1: pop DX

pop CX

ret

BYTE_TO_DEC ENDP

;-----

PRINT PROC near

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT ENDP

;-----

OS_VERSION_DETECTION PROC near

push ax

push dx

```

mov ah, 30h
int 21h
push ax
mov dx, offset SYSTEM_VERSION
call PRINT
cmp al, 0
je ifal0
mov si, offset SYSVER_STR
call BYTE_TO_DEC
pop ax
mov al, ah
add si, 3
call BYTE_TO_DEC
mov dx, offset SYSVER_STR
exit_from_al0:
    call PRINT

mov dx, offset OEM_SN
call PRINT
mov si, offset STR_OEM
add si, 2    ;т.к. OEM может состоять макс. из 2 цифр в 16

с.с.

mov al, bh
call BYTE_TO_DEC
mov dx, offset STR_OEM
call PRINT

mov dx, offset USER_SN
call PRINT
mov di, offset STR_USER ;т.к. WRD_TO_HEX принимает di
add di, 6
mov ax, cx
call WRD_TO_HEX
mov al, bl

```

```

    call BYTE_TO_HEX
    sub di, 2
    mov [di], ax
    mov dx, offset STR_USER
    call PRINT
    jmp exit_from_osdetection

ifal0:
    mov dx, offset IF_AL0
    jmp exit_from_al0

exit_from_osdetection:
    pop dx
    pop ax
    ret
OS_VERSION_DETECTION ENDP
;-----
PC_TYPE_DETECTION PROC near
    push ax
    push dx
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]
    mov dx, offset PC_TYPE
    call PRINT

    ;Определение типа
    cmp al, 0FFh ;PC
    je pc_model
    cmp al, 0FEh ;PC/XT
    je xt_model
    cmp al, 0FBh ;PC/XT
    je xt_model

```



```

cmp al, 0FCh ;AT
je at_model
cmp al, 0FAh ;PS2 30
je ps2_model_30
cmp al, 0FCh ;PS2 50 or 60
je ps2_model_5060
cmp al, 0F8h ;PS2 80
je ps2_model_80
cmp al, 0FDh ;PCjr
je pcjr_model
cmp al, 0F9h ;PC Convertible
je pc_convertible

```

```

;Если неизвестный тип
mov di, offset UNKNOWN_STR
call BYTE_TO_HEX
mov [di], ax
mov dx, offset UNKNOWN_STR
call PRINT
mov dx, offset TYPE_UNKNOWN
jmp end_of_detection

```

```
pc_model:
```

```

    mov dx, offset TYPE_PC
    jmp end_of_detection

```

```
xt_model:
```

```

    mov dx, offset TYPE_PCXT
    jmp end_of_detection

```

```
at_model:
```

```

    mov dx, offset TYPE_AT
    jmp end_of_detection

```

```
ps2_model_30:
```

```

    mov dx, offset TYPE_30
    jmp end_of_detection

```

```

ps2_model_5060:
    mov dx, offset TYPE_5060
    jmp end_of_detection
ps2_model_80:
    mov dx, offset TYPE_80
    jmp end_of_detection
pcjr_model:
    mov dx, offset TYPE_PCJR
    jmp end_of_detection
pc_convertible:
    mov dx, offset TYPE_CONVERTIBLE

end_of_detection:
    call PRINT
    pop dx
    pop ax
    ret

PC_TYPE_DETECTION ENDP
;-----

BEGIN:

    call OS_VERSION_DETECTION
    call PC_TYPE_DETECTION

    xor AL, AL
    mov AH, 4Ch
    int 21h
TESTPC ENDS
END START

```

ПРИЛОЖЕНИЕ Б

КОД .EXE МОДУЛЯ

```
AStack    SEGMENT    STACK
           DW 20h DUP(?)
AStack    ENDS
```

DATA SEGMENT

```
PC_TYPE db 13, 10, "PC type: $"
TYPE_PC db "PC$"
TYPE_PCXT db "PC/XT$"
TYPE_AT db "AT$"
TYPE_30 db "PS2 model 30$"
TYPE_80 db "PS2 model 80$"
TYPE_PCJR db "PCjr$"
TYPE_5060 db "PS2 model 50 or 60$"
TYPE_CONVERTIBLE db "PC Convertible$"
TYPE_UNKNOWN db "Unknown $"
UNKNOWN_STR db "    $"
SYSTEM_VERSION db 13,10,"System version: $"
IF_AL0 db "<2.0$"
OEM_SN db 13, 10, "OEM serial number: $"
STR_OEM db "        $"
STR_USER db "                $"
SYSVER_STR db " . $"
USER_SN db 13, 10, "User serial number: $"
```

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, в 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_1
        or AL,30h
        mov [SI],AL
end_1: pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

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

```

;-----

OS_VERSION_DETECTION PROC near

```
    push ax
    push dx
    mov ah, 30h
    int 21h
    push ax
    mov dx, offset SYSTEM_VERSION
    call PRINT
    cmp al, 0
    je ifal0
    mov si, offset SYSVER_STR
    call BYTE_TO_DEC
    pop ax
    mov al, ah
    add si, 3
    call BYTE_TO_DEC
    mov dx, offset SYSVER_STR
    exit_from_al0:
        call PRINT
```

```
    mov dx, offset OEM_SN
    call PRINT
    mov si, offset STR_OEM
    add si, 2    ;т.к. OEM может состоять макс. из 2 цифр в 16
```

c.c.

```
    mov al, bh
    call BYTE_TO_DEC
    mov dx, offset STR_OEM
    call PRINT
```

```
    mov dx, offset USER_SN
    call PRINT
    mov di, offset STR_USER ;т.к. WRD_TO_HEX принимает di
```

```

add di, 6
mov ax, cx
call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset STR_USER
call PRINT
jmp exit_from_osdetection

```

```

ifal0:
    mov dx, offset IF_AL0
    jmp exit_from_al0

```

```

exit_from_osdetection:
    pop dx
    pop ax
    ret

```

OS_VERSION_DETECTION ENDP

;-----

PC_TYPE_DETECTION PROC near

```

push ax
push dx
mov ax, 0F000h
mov es, ax
mov al, es:[0FFFEH]
mov dx, offset PC_TYPE
call PRINT

```

```

;Определение типа
cmp al, 0FFh ;PC
je pc_model

```

```

cmp al, 0FEh ;PC/XT
je xt_model
cmp al, 0FBh ;PC/XT
je xt_model
cmp al, 0FCh ;AT
je at_model
cmp al, 0FAh ;PS2 30
je ps2_model_30
cmp al, 0FCh ;PS2 50 or 60
je ps2_model_5060
cmp al, 0F8h ;PS2 80
je ps2_model_80
cmp al, 0FDh ;PCjr
je pcjr_model
cmp al, 0F9h ;PC Convertible
je pc_convertible

```

```

;Если неизвестный тип
mov di, offset UNKNOWN_STR
call BYTE_TO_HEX
mov [di], ax
mov dx, offset UNKNOWN_STR
call PRINT
mov dx, offset TYPE_UNKNOWN
jmp end_of_detection

```

```
pc_model:
```

```

    mov dx, offset TYPE_PC
    jmp end_of_detection

```

```
xt_model:
```

```

    mov dx, offset TYPE_PCXT
    jmp end_of_detection

```

```
at_model:
```

```

    mov dx, offset TYPE_AT

```



```

        jmp end_of_detection
ps2_model_30:
        mov dx, offset TYPE_30
        jmp end_of_detection
ps2_model_5060:
        mov dx, offset TYPE_5060
        jmp end_of_detection
ps2_model_80:
        mov dx, offset TYPE_80
        jmp end_of_detection
pcjr_model:
        mov dx, offset TYPE_PCJR
        jmp end_of_detection
pc_convertible:
        mov dx, offset TYPE_CONVERTIBLE

end_of_detection:
        call PRINT
pop dx
pop ax
ret

```

PC_TYPE_DETECTION ENDP

;-----

BEGIN PROC FAR

```

push ax
sub ax, ax
mov ax, data
mov ds, ax
pop ax
call OS_VERSION_DETECTION
call PC_TYPE_DETECTION

```

```
        xor AL, AL
        mov AH, 4Ch
        int 21h
BEGIN ENDP
CODE ENDS
END BEGIN
```