

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ
по лабораторной работе №1
по дисциплине «Операционные системы»
Тема: Исследование структур загрузочных модулей

Студент гр.8382

_____ Фильцин 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

В ходе лабораторной работы был написан код программы (см. исх. код в приложении А) и получен “хороший” .COM модуль.

При запуске программы происходит печать следующего сообщения:

PS2 model 50 or 60

Version: 05.00

OEM: 0

Serial number: 000

Из того же самого исходного кода был получен “плохой” .EXE модуль.

При запуске программы происходит печать несвязного мусора.

После этого исходный код был модифицирован (см. исх. код в приложении Б) и получен “хороший” .EXE модуль, который вывел ту же самую информацию, что .COM модуль.

Контрольные вопросы

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

1 сегмент.

2) EXE-программа?

Может содержать несколько сегментов (например, сегмент кода, стека,

данных и т.д.) . Для того, чтобы программа была работоспособна (запускалась) достаточно 1 сегмента (кода).

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

`assume` - “привязывает” сегмент к сегментному регистру. (Сохраняет адрес сегмента в сегментный регистр)

`org` - задаёт смещение внутри сегмента.

Т.к. в DOS первые 100h байт резервируются под PSP, необходимо писать `org 100h`.

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

Нельзя использовать команды, связанные с адресацией сегментов т.к. отсутствует таблица настройки адресов (которая состоит из значений сегмент-смещение).

Отличия форматов файлов COM и EXE модулей

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

Файл состоит из 1 сегмента и начинается с 0 адреса (См. рис. 1)

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

Структура файла “плохого” EXE показана на рис. 2.

В 0 адресе находится стандартный заголовок EXE программы. Код начинается с адреса 300h.

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

Структура файла “хорошего” EXE показана на рис. 3.

В хорошем “EXE” сегмент кода начинается с адреса 200h. При этом сегмент данных начинается с адреса 300h. Так же из листинга программы ясно, что сегмент данных и сегмент стека объединены в одну группу DGROUP.

```

~/o/c/л/MASM >>> hexyl 1COM.COM
00000000 e9 0d 01 50 43 0d 0a 24 50 43 2f 58 54 0d 0a 24 x •PC_$ PC/XT_$
00000010 41 54 0d 0a 24 50 53 32 20 6d 6f 64 65 6c 20 33 AT_$PS2 model 3
00000020 30 0d 0a 24 50 53 32 20 6d 6f 64 65 6c 20 35 30 0_$PS2 model 50
00000030 20 6f 72 20 36 30 0d 0a 24 50 53 32 20 6d 6f 64 or 60_$PS2 mod
00000040 65 6c 20 38 30 0d 0a 24 50 43 6a 72 0d 0a 24 50 el 80_$PCjr_$P
00000050 43 20 43 6f 6e 76 65 72 74 69 62 6c 65 0d 0a 24 C Conver tible_$
00000060 41 6e 6f 74 68 65 72 20 24 09 00 ff fe fb fc fa Another $_0xxxxx
00000070 fc f8 fd f9 03 01 08 01 08 01 10 01 15 01 24 01 xxxx•••• •••••$•
00000080 39 01 48 01 4f 01 56 65 72 73 69 6f 6e 3a 20 24 9•H•0•Ve rsion: $
00000090 30 30 2e 30 30 0d 0a 24 4f 45 4d 3a 20 24 20 20 00,00_$ OEM: $
000000a0 0d 0a 24 53 65 72 69 61 6c 20 6e 75 6d 62 65 72 _$Seria l number
000000b0 3a 20 24 30 30 30 0d 0a 24 24 24 0f 3c 09 76 02 04 : $000_$ $•<_v••
000000c0 07 04 30 c3 51 8a e0 e8 ef ff 86 c4 b1 04 d2 e8 ••0xQxxx xxxxx•xx
000000d0 e8 e6 ff 59 c3 53 8a fc e8 e9 ff 88 25 4f 88 05 xxxYxSxx xxxx%0x•
000000e0 4f 8a c7 e8 de ff 88 25 4f 88 05 5b c3 51 52 32 0xxxxxx% 0x•[xQR2
000000f0 e4 33 d2 b9 0a 00 f7 f1 80 ca 30 88 14 4e 33 d2 x3xx_0xx xx0x•N3x
00000100 3d 0a 00 73 f1 3c 00 74 04 0c 30 88 04 5a 59 c3 =_0sx<0t •_0x•ZYx
00000110 b8 00 f0 8e c0 26 a0 fe ff 8b 0e 69 01 49 be 6b x0xxx&xx xx•i•I•k
00000120 01 8b d9 8a 18 3a d8 74 17 e2 f6 ba 60 01 b4 09 •xxx•:xt •xxx••x_
00000130 cd 21 8a c1 e8 8d ff 8b d0 b4 02 cd 21 eb 10 90 x!xxxxxx xx•x!x•x
00000140 8b c1 be 74 01 8b d9 03 db 8b 10 b4 09 cd 21 ba xxxt•xx• xx•x_x!x
00000150 86 01 b4 09 cd 21 b4 30 cd 21 be 90 01 46 50 e8 xx•x_x!x0 x!xx•FPx
00000160 8b ff 58 86 e0 be 90 01 83 c6 03 e8 7f ff ba 90 xxXxxxx• xx•x•xxx
00000170 01 b4 09 cd 21 ba 98 01 b4 09 cd 21 8a c7 be 9e •x_x!xx• x_x!xxxx
00000180 01 e8 69 ff ba 9e 01 b4 09 cd 21 ba a3 01 cd 21 •x!xxxx•x _x!xx•x!
00000190 8a c3 be b3 01 e8 55 ff 8a c5 be b3 01 46 e8 4c xxxx•xUx xxxx•FxL
000001a0 ff 8a c1 be b3 01 83 c6 02 e8 41 ff ba b3 01 b4 xxxxx•xx •xAxxx•x
000001b0 09 cd 21 32 c0 b4 4c cd 21 _x!2xxLx !
~/o/c/л/MASM >>> 

```

Рис. 1: Структура файла COM

```

~/o/c/л/MASM >>> hexyl 1COM.EXE
00000000 4d 5a b9 00 03 00 00 00 20 00 00 00 ff ff 00 00 MZx0•000 000xx00
00000010 00 00 36 b9 00 01 00 00 1e 00 00 00 01 00 00 00 006x0•00 •000•000
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00000000 00000000
*
00000300 e9 0d 01 50 43 0d 0a 24 50 43 2f 58 54 0d 0a 24 x •PC_$ PC/XT_$
00000310 41 54 0d 0a 24 50 53 32 20 6d 6f 64 65 6c 20 33 AT_$PS2 model 3
00000320 30 0d 0a 24 50 53 32 20 6d 6f 64 65 6c 20 35 30 0_$PS2 model 50
00000330 20 6f 72 20 36 30 0d 0a 24 50 53 32 20 6d 6f 64 or 60_$PS2 mod
00000340 65 6c 20 38 30 0d 0a 24 50 43 6a 72 0d 0a 24 50 el 80_$PCjr_$P
00000350 43 20 43 6f 6e 76 65 72 74 69 62 6c 65 0d 0a 24 C Conver tible_$
00000360 41 6e 6f 74 68 65 72 20 24 09 00 ff fe fb fc fa Another $_0xxxxx
00000370 fc f8 fd f9 03 01 08 01 08 01 10 01 15 01 24 01 xxxx•••• •••••$•
00000380 39 01 48 01 4f 01 56 65 72 73 69 6f 6e 3a 20 24 9•H•0•Ve rsion: $
00000390 30 30 2e 30 30 0d 0a 24 4f 45 4d 3a 20 24 20 20 00,00_$ OEM: $
000003a0 0d 0a 24 53 65 72 69 61 6c 20 6e 75 6d 62 65 72 _$Seria l number
000003b0 3a 20 24 30 30 30 0d 0a 24 24 24 0f 3c 09 76 02 04 : $000_$ $•<_v••
000003c0 07 04 30 c3 51 8a e0 e8 ef ff 86 c4 b1 04 d2 e8 ••0xQxxx xxxxx•xx
000003d0 e8 e6 ff 59 c3 53 8a fc e8 e9 ff 88 25 4f 88 05 xxxYxSxx xxxx%0x•
000003e0 4f 8a c7 e8 de ff 88 25 4f 88 05 5b c3 51 52 32 0xxxxxx% 0x•[xQR2
000003f0 e4 33 d2 b9 0a 00 f7 f1 80 ca 30 88 14 4e 33 d2 x3xx_0xx xx0x•N3x
00000400 3d 0a 00 73 f1 3c 00 74 04 0c 30 88 04 5a 59 c3 =_0sx<0t •_0x•ZYx
00000410 b8 00 f0 8e c0 26 a0 fe ff 8b 0e 69 01 49 be 6b x0xxx&xx xx•i•I•k
00000420 01 8b d9 8a 18 3a d8 74 17 e2 f6 ba 60 01 b4 09 •xxx•:xt •xxx••x_
00000430 cd 21 8a c1 e8 8d ff 8b d0 b4 02 cd 21 eb 10 90 x!xxxxxx xx•x!x•x
00000440 8b c1 be 74 01 8b d9 03 db 8b 10 b4 09 cd 21 ba xxxt•xx• xx•x_x!x
00000450 86 01 b4 09 cd 21 b4 30 cd 21 be 90 01 46 50 e8 xx•x_x!x0 x!xx•FPx
00000460 8b ff 58 86 e0 be 90 01 83 c6 03 e8 7f ff ba 90 xxXxxxx• xx•x•xxx
00000470 01 b4 09 cd 21 ba 98 01 b4 09 cd 21 8a c7 be 9e •x_x!xx• x_x!xxxx
00000480 01 e8 69 ff ba 9e 01 b4 09 cd 21 ba a3 01 cd 21 •x!xxxx•x _x!xx•x!
00000490 8a c3 be b3 01 e8 55 ff 8a c5 be b3 01 46 e8 4c xxxx•xUx xxxx•FxL
000004a0 ff 8a c1 be b3 01 83 c6 02 e8 41 ff ba b3 01 b4 xxxxx•xx •xAxxx•x
000004b0 09 cd 21 32 c0 b4 4c cd 21 _x!2xxLx !
~/o/c/л/MASM >>> 

```

Рис. 2: Структура “плохого” EXE

Таким образом, сегмент стека будет расположен сразу за основной памятью программы (и выровнен по адресу кратному 16) Запуск программы в отладчике это подтверждает.

```

~/о/с/л/МASM >>> hexyl 1EXE.EXE
00000000 4d 5a be 01 02 00 01 00 20 00 11 00 ff ff 1c 00 MZx...0x0 0x0xx0
00000010 00 01 29 e5 00 00 00 00 1e 00 00 00 01 00 5b 00 0x0)x0000 00000[0
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00000000:00000000
*
00000200 eb 58 90 24 0f 3c 09 76 02 04 07 04 30 c3 51 8a xXx$*<_v .....0xQx
00000210 e0 e8 ef ff 86 c4 b1 04 d2 e8 e8 e6 ff 59 c3 53 xxxxxxxx xxxxxYxS
00000220 8a fc e8 e9 ff 88 25 4f 88 05 4f 8a c7 e8 de ff xxxxxxx0 x0xxxxx
00000230 88 25 4f 88 05 5b c3 51 52 32 e4 33 d2 b9 0a 00 x%0x*[xQ R2x3xx_0
00000240 f7 f1 80 ca 30 88 14 4e 33 d2 3d 0a 00 73 f1 3c xxxx0x*N 3x=_0$x<
00000250 00 74 04 0c 30 88 04 5a 59 c3 b8 10 00 8e d8 b8 0t*_0x*Z Yxx*0xxx
00000260 00 f0 8e c0 26 a0 fe ff 8b 0e 6e 00 49 be 70 00 0xxx0xxx x*n0Ixp0
00000270 8b d9 8a 18 3a d8 74 17 e2 f6 ba 65 00 b4 09 cd xxx*:xt* xxex0x_x
00000280 21 8a c1 e8 88 ff 8b d0 b4 02 cd 21 eb 10 90 8b !xxxxxx x*x!x*x
00000290 c1 be 79 00 8b d9 03 db 8b 10 b4 09 cd 21 ba 8b xxy0xx*x x*x!xx
000002a0 00 b4 09 cd 21 b4 30 cd 21 be 95 00 46 50 e8 86 0x_x!x0x !xx0FPxx
000002b0 ff 58 86 e0 be 95 00 83 c6 03 e8 7a ff ba 95 00 xXxxxx0x xzxxxx0
000002c0 b4 09 cd 21 ba 9d 00 b4 09 cd 21 8a c7 be a3 00 x_x!xx0x _x!xxxx0
000002d0 e8 64 ff ba a3 00 b4 09 cd 21 ba a8 00 cd 21 8a xdx0x_ x!xx0x!x
000002e0 c3 be b8 00 e8 50 ff 8a c5 be b8 00 46 e8 47 ff xxx0xPxx xxx0Fxx
000002f0 8a c1 be b8 00 83 c6 02 e8 3c ff ba b8 00 b4 09 xxx0xxx x<xxx0x_
00000300 cd 21 32 c0 b4 4c cd 21 50 43 0d 0a 24 50 43 2f x!2xxLx! PC__$PC/
00000310 58 54 0d 0a 24 41 54 0d 0a 24 50 53 32 20 6d 6f XT__$AT_ $PS2 mo
00000320 64 65 6c 20 33 30 0d 0a 24 50 53 32 20 6d 6f 64 del 30_ $PS2 mod
00000330 65 6c 20 35 30 20 6f 72 20 36 30 0d 0a 24 50 53 el 50 or 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 41 6e 6f 74 68 65 72 20 24 09 00 le_ $Ano ther $_0
00000370 ff fe fb fc fa fc f8 fd f9 08 00 0d 00 0d 00 15 xxxxxxxx x0_0_0_0_
00000380 00 1a 00 29 00 3e 00 4d 00 54 00 56 65 72 73 69 0x0)0>0M 0T0Versi
00000390 6f 6e 3a 20 24 30 30 2e 30 30 0d 0a 24 4f 45 4d on: $00_ 00_ $OEM
000003a0 3a 20 24 20 20 0d 0a 24 53 65 72 69 61 6c 20 6e : $ __$ Serial n
000003b0 75 6d 62 65 72 3a 20 24 30 30 30 0d 0a 24 umber: $ 000_ $
~/о/с/л/МASM >>> 

```

Рис. 3: Структура “хорошего” EXE

Загрузка COM модуля в основную память

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

Код располагается с адреса 100h.

2) Что располагается с адреса 0?

PSP.

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

Сегментные регистры имеют значение 119C. Они указывают на начало

PSP.

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

При запуске программы SP = FFFEH. Это значит, что стек расположен в самом верху сегмента и растёт вниз.

Загрузка “хорошего” EXE модуля в основную память

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

CS = 11AC

DS = 119C

ES = 119C

SS = 11C8

Значения регистров см. на рис. 4.

AX 0000	SI 0000	CS 11AC	IP 0057	Stack +0 0000	FLAGS 0200												
BX 0000	DI 0000	DS 119C		+2 0000													
CX 01C0	BP 0000	ES 119C	HS 119C	+4 0000	OF	DF	IF	SF	ZF	AF	PF	CF					
DX 0000	SP 0100	SS 11C8	FS 119C	+6 0000	0	0	1	0	0	0	0	0					
CMD >					1												
					DS:0000 CD 20 9C 11 00 EA FD FF												
					DS:0008 AD DE ED 04 92 01 00 00												
0057 B8BC11 MDV AX,11BC					DS:0010 18 01 10 01 18 01 92 01												
005A 8ED8 MDV DS,AX					DS:0018 03 FF FF FF FF FF FF FF												
005C B8FFFF MDV AX,FFFF					DS:0020 FF FF FF FF FF FF FF FF												
005F 50 PUSH AX					DS:0028 FF FF FF FF 96 11 C4 FF												
0060 50 PUSH AX					DS:0030 92 01 14 00 18 00 9C 11												
0061 50 PUSH AX					DS:0038 FF FF FF FF 00 00 00 00												
0062 50 PUSH AX					DS:0040 05 00 00 00 00 00 00 00												
0063 50 PUSH AX					DS:0048 00 00 00 00 00 00 00 00												
2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
DS:0000	CD	20	9C	11	00	EA	FD	FF	AD	DE	ED	04	92	01	00	00	
DS:0010	18	01	10	01	18	01	92	01	03	FF	FF	FF	FF	FF	FF	FF	
DS:0020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	96	11	C4	FF		
DS:0030	92	01	14	00	18	00	9C	11	FF	FF	FF	FF	00	00	00	00	
DS:0040	05	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
1	Step	2StepProc	3Retrieve	4	Help	5Set BRK	6		7	up	8	dn	9	le	0	ri	

Рис. 4: Значения регистров в запущенном отладчике

DOS определяет минимальный адрес, начиная с которого может быть загружен EXE (программный сегмент). По смещению 0h в программном сегменте находится PSP, сама программа находится по смещению 0100h.

2) На что указывают регистры DS и ES?

PSP

3) Как определяется стек?

Размер стека задается программно в момент компиляции (директива .stack или с помощью директивы assume).

Так, на момент запуска програмы $SP = 0100h$.

4) Как определяется точка входа?

С помощью директивы end и указания функции или метки. Если метка или функции не указана, точкой входа является первая инструкция в сегменте code.

Вывод

В ходе лабораторной работы были исследованы COM и EXE файлы с выявлением их различий.

Приложение А. Исходный код программы COM

```
testpc segment
    assume CS:testpc, ds:testpc, es:nothing, ss:
        nothing
    org 100h
start: jmp begin

pc_t db 'PC', 13, 10, '$'
pcxt_t db 'PC/XT', 13, 10, '$'
at_t db 'AT', 13, 10, '$'
ps2_30_t db 'PS2 model 30', 13, 10, '$'
ps2_50_t db 'PS2 model 50 or 60', 13, 10, '$'
ps2_80_t db 'PS2 model 80', 13, 10, '$'
pcjr_t db 'PCjr', 13, 10, '$'
pcconv_t db 'PC Convertible', 13, 10, '$'
another_t db 'Another ', '$'
count_types dw 9
type_array db 0ffh
            db 0feh
            db 0fbh
            db 0fch
            db 0fah
            db 0fch
            db 0f8h
            db 0fdh
            db 0f9h
offset_array dw offset pc_t
```

```
    dw offset pcxt_t
    dw offset pcxt_t
    dw offset at_t
    dw offset ps2_30_t
    dw offset ps2_50_t
    dw offset ps2_80_t
    dw offset pcjr_t
    dw offset pcconv_t
```

```
os_ver_title db 'Version: ', '$'
os_ver db '00.00', 13, 10, '$'
oem_title db 'OEM: ', '$'
oem_value db ' ', 13, 10, '$'
serial_title db 'Serial number: ', '$'
serial_value db '000', 13, 10, '$'
```

```
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

```

```

wrd_to_hex proc near
    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**

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:[0ffffh]

mov cx, count_types
dec cx
mov si, offset type_array
find_type:
    mov bx, cx
    mov bl, ds:[si + bx]
    cmp bl, al
    je result_type
    loop find_type
fail_type:
    mov dx, offset another_t
    mov ah, 09h
    int 21h
    mov al, cl
    call byte_to_hex
    mov dx, ax
    mov ah, 02h
    int 21h
    jmp finish_type
result_type:
    mov ax, cx
    mov si, offset offset_array
    mov bx, cx

```

```

    mov dx, ds:[si + bx]
    mov ah, 09h
    int 21h
finish_type:

    mov dx, offset os_ver_title
    mov ah, 09h
    int 21h

    mov ah, 30h
    int 21h

    mov si, offset os_ver
    inc si
    push ax
    call byte_to_dec
    pop ax

    xchg ah, al

    mov si, offset os_ver
    add si, 3
    call byte_to_dec

    mov dx, offset os_ver
    mov ah, 09h
    int 21h

```



```
mov dx, offset oem_title
mov ah, 09h
int 21h
```

```
mov al, bh
mov si, offset oem_value
call byte_to_dec
mov dx, offset oem_value
mov ah, 09h
int 21h
```

```
mov dx, offset serial_title
int 21h
```

```
mov al, bl
mov si, offset serial_value
call byte_to_dec
```

```
mov al, ch
mov si, offset serial_value
inc si
call byte_to_dec
```

```
mov al, cl
mov si, offset serial_value
add si, 2
```

```
call byte_to_dec

mov dx, offset serial_value
mov ah, 09h
int 21h

xor al, al
mov ah, 4ch
int 21h

testpc ends

end start
```

Приложение Б. Исходный код программы EXE

.model small

.stack 100h

.data

pc_t **db** 'PC', 13, 10, '\$'

pcxt_t **db** 'PC/XT', 13, 10, '\$'

at_t **db** 'AT', 13, 10, '\$'

ps2_30_t **db** 'PS2 model 30', 13, 10, '\$'

ps2_50_t **db** 'PS2 model 50 or 60', 13, 10, '\$'

ps2_80_t **db** 'PS2 model 80', 13, 10, '\$'

pcjr_t **db** 'PCjr', 13, 10, '\$'

pcconv_t **db** 'PC Convertible', 13, 10, '\$'

another_t **db** 'Another ', '\$'

count_types **dw** 9

type_array **db** 0ffh

db 0feh

db 0fbh

db 0fch

db 0fah

db 0fch

db 0f8h

db 0fdh

db 0f9h

offset_array **dw offset** pc_t

dw offset pcxt_t

dw offset pcxt_t

```

        dw offset at_t
        dw offset ps2_30_t
        dw offset ps2_50_t
        dw offset ps2_80_t
        dw offset pcjr_t
        dw offset pcconv_t

```

```

os_ver_title db 'Version: ', '$'
os_ver      db '00.00', 13, 10, '$'
oem_title   db 'OEM: ', '$'
oem_value   db ' ', 13, 10, '$'
serial_title db 'Serial number: ', '$'
serial_value db '000', 13, 10, '$'

```

```

.code

```

```

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

```

```

wrd_to_hex proc near
    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**

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, @data


```

mov ds, ax

mov ax, 0f000h
mov es, ax
mov al, es:[0ffffh]

mov cx, count_types
dec cx
mov si, offset type_array
find_type:
    mov bx, cx
    mov bl, ds:[si + bx]
    cmp bl, al
    je result_type
    loop find_type
fail_type:
    mov dx, offset another_t
    mov ah, 09h
    int 21h
    mov al, cl
    call byte_to_hex
    mov dx, ax
    mov ah, 02h
    int 21h
    jmp finish_type
result_type:
    mov ax, cx

```

```
    mov si, offset offset_array
    mov bx, cx
    add bx, bx
    mov dx, ds:[si + bx]
    mov ah, 09h
    int 21h
finish_type:
```

```
mov dx, offset os_ver_title
int 21h
```

```
mov si, offset os_ver
inc si
push ax
call byte_to_dec
pop ax
```

```
xchg ah, al
```

```
mov si, offset os_ver
add si, 3
call byte_to_dec
```

```
mov dx, offset os_ver
mov ah, 09h
int 21h
```

```
mov dx, offset oem_title
mov ah, 09h
int 21h
```

```
mov al, bh
mov si, offset oem_value
call byte_to_dec
mov dx, offset oem_value
mov ah, 09h
int 21h
```

```
mov dx, offset serial_title
int 21h
```

```
mov al, bl
mov si, offset serial_value
call byte_to_dec
```

```
mov al, ch
mov si, offset serial_value
inc si
call byte_to_dec
```

```
mov al, cl
mov si, offset serial_value
add si, 2
call byte_to_dec
```

```
mov dx, offset serial_value
mov ah, 09h
int 21h

xor al, al
mov ah, 4ch
int 21h
end begin
```