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 роботоспособна (запускалась) достаточно 1 сегмента (кода).

3) Какие директивы должны обязатель 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 │$_0×××××│
│00000070│ fc f8 fd f9 03 01 08 01 │ 08 01 10 01 15 01 24 01 │××××••••│••••••$•│
│00000080│ 39 01 48 01 4f 01 56 65 │ 72 73 69 6f 6e 3a 20 24 │9•H•O•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 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 │••0×Q×××│××××•××│
│000000d0│ e8 e6 ff 59 c3 53 8a fc │ e8 e9 ff 88 25 4f 88 05 │×××Y×S××│××××%O×•│
│000000e0│ 4f 8a c7 e8 de ff 88 25 │ 4f 88 05 5b c3 51 52 32 │O××××××%│O×•[×QR2│
│000000f0│ e4 33 d2 b9 0a 00 f7 f1 │ 80 ca 30 88 14 4e 33 d2 │×3××_0××│××0×•N3×│
│00000100│ 3d 0a 00 73 f1 3c 00 74 │ 04 0c 30 88 04 5a 59 c3 │=_0s×<0t│•_0×•ZY×│
│00000110│ b8 00 f0 8e c0 26 a0 fe │ ff 8b 0e 69 01 49 be 6b │×0×××&××│××•i•I×k│
│00000120│ 01 8b d9 8a 18 3a d8 74 │ 17 e2 f6 ba 60 01 b4 09 │•×××•:××t│•×××`•×_│
│00000130│ cd 21 8a c1 e8 8d ff 8b │ d0 b4 02 cd 21 eb 10 90 │×!××××××│××•×!×•×│
│00000140│ 8b c1 be 74 01 8b d9 03 │ db 8b 10 b4 09 cd 21 ba │×××t•××•│××•×_×!×│
│00000150│ 86 01 b4 09 cd 21 b4 30 │ cd 21 be 90 01 46 50 e8 │×•×_×!×0│×!××•FP×│
│00000160│ 8b ff 58 86 e0 be 90 01 │ 83 c6 03 e8 7f ff ba 90 │××X××××•│××•×•×××│
│00000170│ 01 b4 09 cd 21 ba 98 01 │ b4 09 cd 21 8a c7 be 9e │•×_×!××•│×_×!××××│
│00000180│ 01 e8 69 ff ba 9e 01 b4 │ 09 cd 21 ba a3 01 cd 21 │•×i×××•×│_×!××•×!│
│00000190│ 8a c3 be b3 01 e8 55 ff │ 8a c5 be b3 01 46 e8 4c │××××•×U×│××××•F×L│
│000001a0│ ff 8a c1 be b3 01 83 c6 │ 02 e8 41 ff ba b3 01 b4 │××××•×××│•×A×××•×│
│000001b0│ 09 cd 21 32 c0 b4 4c cd │ 21                      │_×!2××L×│!       │
└────────┴─────────────────────────┴─────────────────────────┴────────┴────────┘
~/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 │MZ×0•000│ 000××00│
│00000010│ 00 00 36 b9 00 01 00 00 │ 1e 00 00 00 01 00 00 00 │006×0•00│•000•000│
│00000020│ 00 00 00 00 00 00 00 00 │ 00 00 00 00 00 00 00 00 │00000000│00000000│
│*       │                         │                         │        │        │
│00000300│ e9 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 │$_0×××××│
│00000370│ fc f8 fd f9 03 01 08 01 │ 08 01 10 01 15 01 24 01 │××××••••│••••••$•│
│00000380│ 39 01 48 01 4f 01 56 65 │ 72 73 69 6f 6e 3a 20 24 │9•H•O•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 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 │••0×Q×××│××××•××│
│000003d0│ e8 e6 ff 59 c3 53 8a fc │ e8 e9 ff 88 25 4f 88 05 │×××Y×S××│××××%O×•│
│000003e0│ 4f 8a c7 e8 de ff 88 25 │ 4f 88 05 5b c3 51 52 32 │O××××××%│O×•[×QR2│
│000003f0│ e4 33 d2 b9 0a 00 f7 f1 │ 80 ca 30 88 14 4e 33 d2 │×3××_0××│××0×•N3×│
│00000400│ 3d 0a 00 73 f1 3c 00 74 │ 04 0c 30 88 04 5a 59 c3 │=_0s×<0t│•_0×•ZY×│
│00000410│ b8 00 f0 8e c0 26 a0 fe │ ff 8b 0e 69 01 49 be 6b │×0×××&××│××•i•I×k│
│00000420│ 01 8b d9 8a 18 3a d8 74 │ 17 e2 f6 ba 60 01 b4 09 │•×××•:××t│•×××`•×_│
│00000430│ cd 21 8a c1 e8 8d ff 8b │ d0 b4 02 cd 21 eb 10 90 │×!××××××│××•×!×•×│
│00000440│ 8b c1 be 74 01 8b d9 03 │ db 8b 10 b4 09 cd 21 ba │×××t•××•│××•×_×!×│
│00000450│ 86 01 b4 09 cd 21 b4 30 │ cd 21 be 90 01 46 50 e8 │×•×_×!×0│×!××•FP×│
│00000460│ 8b ff 58 86 e0 be 90 01 │ 83 c6 03 e8 7f ff ba 90 │××X××××•│××•×•×××│
│00000470│ 01 b4 09 cd 21 ba 98 01 │ b4 09 cd 21 8a c7 be 9e │•×_×!××•│×_×!××××│
│00000480│ 01 e8 69 ff ba 9e 01 b4 │ 09 cd 21 ba a3 01 cd 21 │•×i×××•×│_×!××•×!│
│00000490│ 8a c3 be b3 01 e8 55 ff │ 8a c5 be b3 01 46 e8 4c │××××•×U×│××××•F×L│
│000004a0│ ff 8a c1 be b3 01 83 c6 │ 02 e8 41 ff ba b3 01 b4 │××××•×××│•×A×××•×│
│000004b0│ 09 cd 21 32 c0 b4 4c cd │ 21                      │_×!2××L×│!       │
└────────┴─────────────────────────┴─────────────────────────┴────────┴────────┘
~/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 подтвержает.

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.

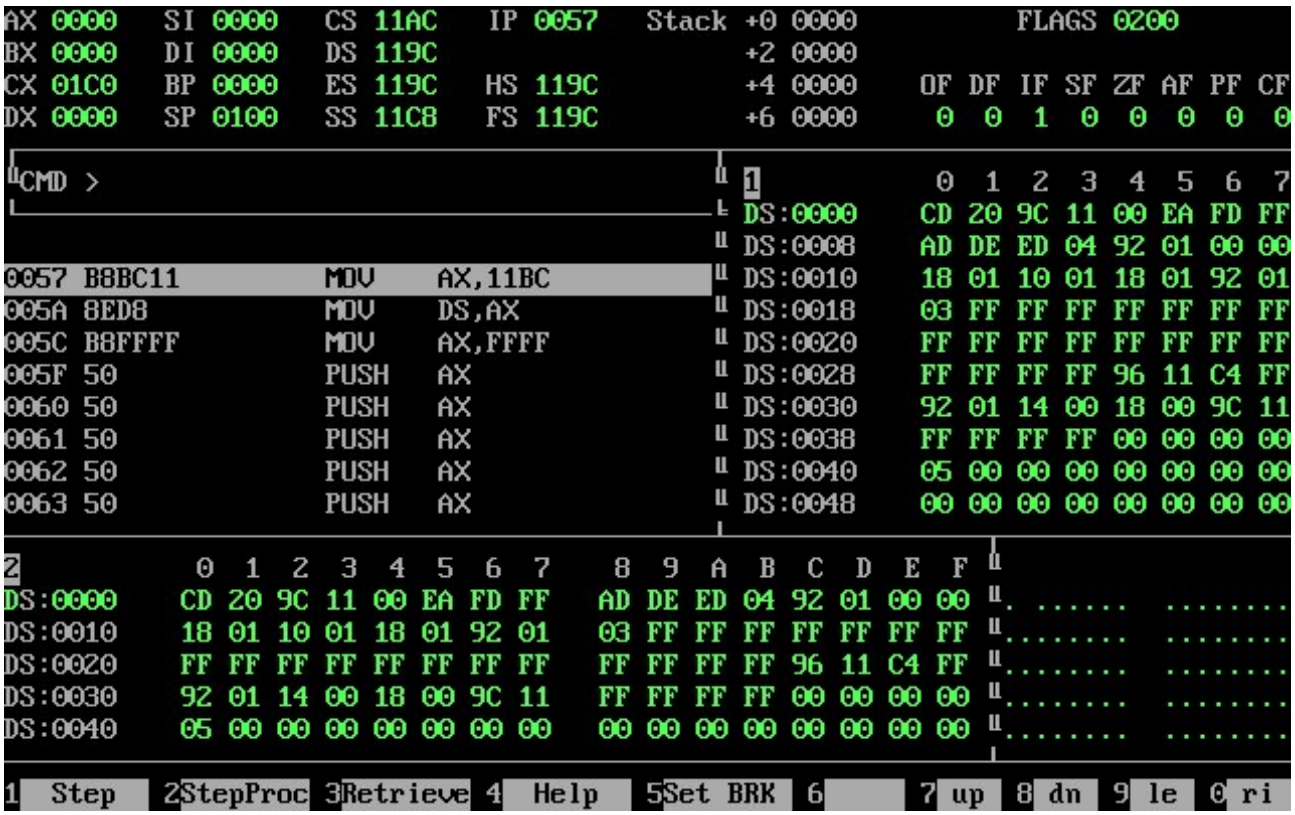

Рис. 4: Значения регистров в запущенном отладчике

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 функци 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:[0fffeh]

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:[0fffeh]

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
```