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

Студентка гр. 8382 _____ Рочева А.К.

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.

1. Для определения типа PC и версии системы были написаны тексты .COM и .EXE модулей (см. Приложение A и B). Тип PC определяется предпоследним байтом ROM BIOS (табл. 1).

Таблица 1 – Соответствие кода и типа

PC	FF
PC/XT	FE, FB
AT	FC
PS2 модель 30	FA
PS2 модель 50 или 60	FC
PS2 модель 80	F8
PCjr	FD
PC Convertible	F9

Версия системы определяется значением регистров AL, AH, BH, BL:CX, полученных после выполнения функции 30H прерывания 21H. (в AL – номер основной версии, в AH – номер модификации, в BH – серийный номер OEM, в BL:CX – 24-битовый серийный номер пользователя).

Результат выполнения .COM модуля представлен на рис. 1. Результат выполнения «плохого» .EXE модуля, полученного из исходного текста для .COM модуля, представлен на рис. 2. Результат выполнения «хорошего» .EXE модуля представлен на рис. 3.

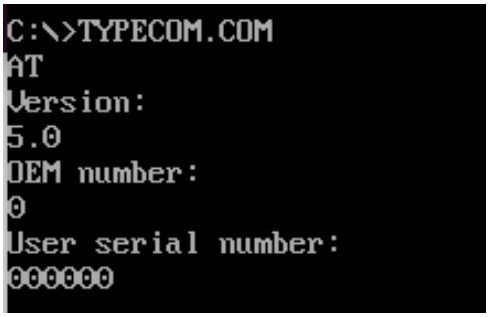

рис. 1 – результат выполнения .COM модуля

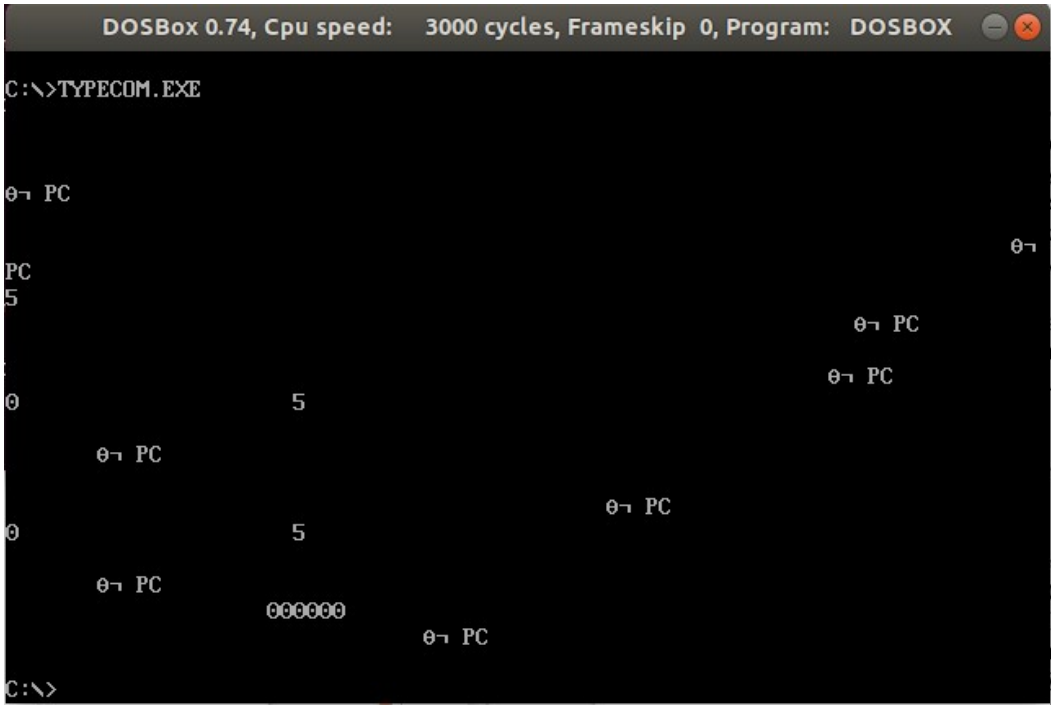

рис. 2 – результат выполнения «плохого» .EXE модуля

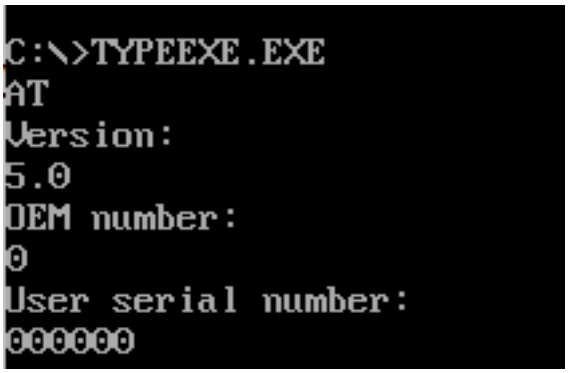

рис. 3 -результат выполнения «хорошего» .EXE модуля

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

- COM-программа должна содержать один сегмент.

2. EXE-программа?

- EXE-программа может содержать разное кол-во сегментов в зависимости от используемой модели памяти (пр: для small может быть только один сегмент кода и один сегмент данных, для large — несколько сегментов кода и несколько сегментов данных).

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

- ORG 100h — устанавливает относительный адрес для начала выполнения программы (т.е резервирует 100h байт от начального адреса для PSP).
- ASSUME — устанавливает соответствие сегментных регистров сегментам (все они будут указывать на один сегмент, потому что он единственный).

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

- в COM-программе нельзя использовать команды, использующие адрес сегмента, т.к этот адрес становится известным только после загрузки сегмента в память.

2. Файл загрузочного модуля .COM, «плохого» .EXE и «хорошего» в шестнадцатеричном виде представлены на рис. 4, 5, 6, 7.

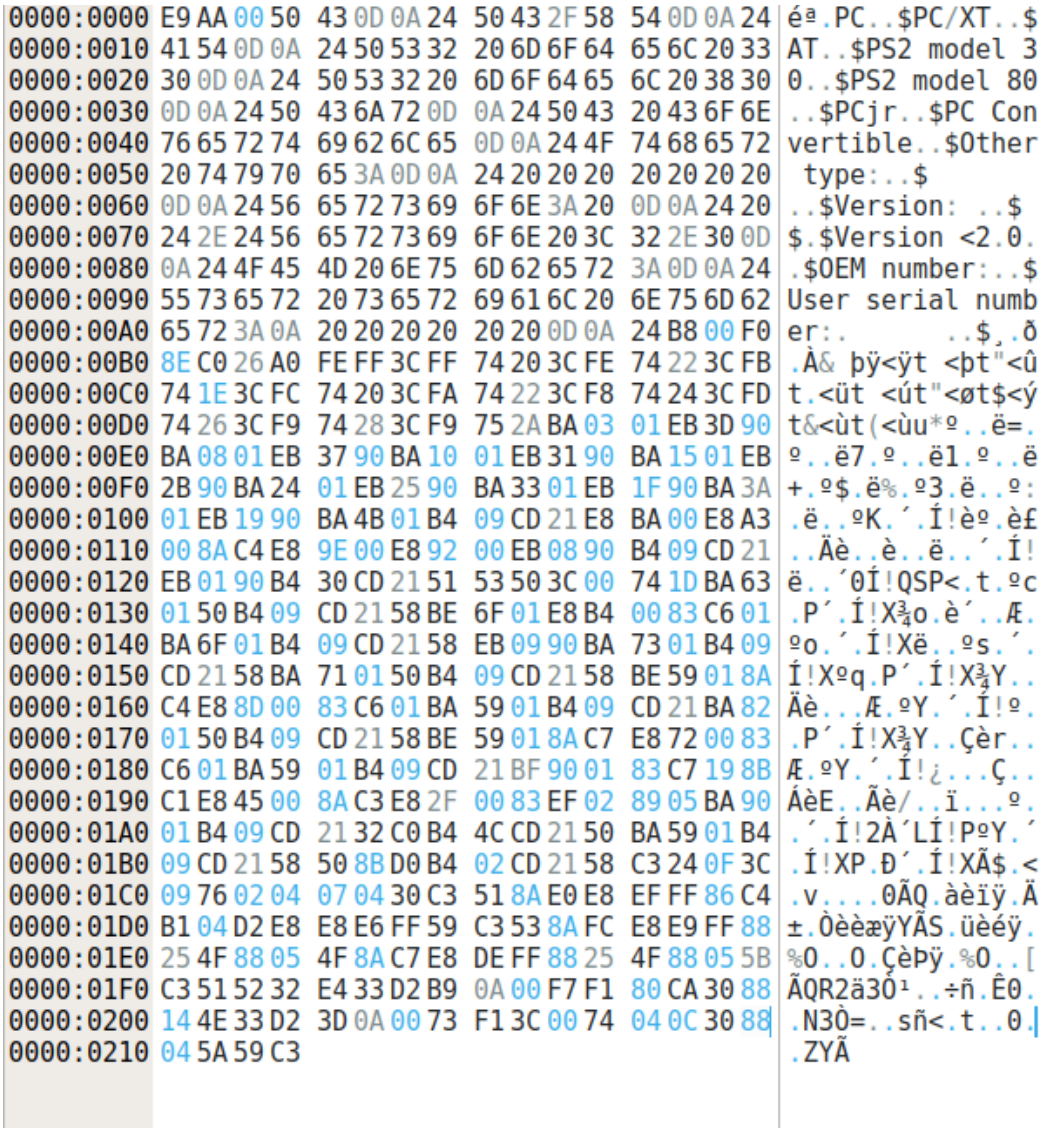

рис. 4 - Файл загрузочного модуля .COM в 16-м виде

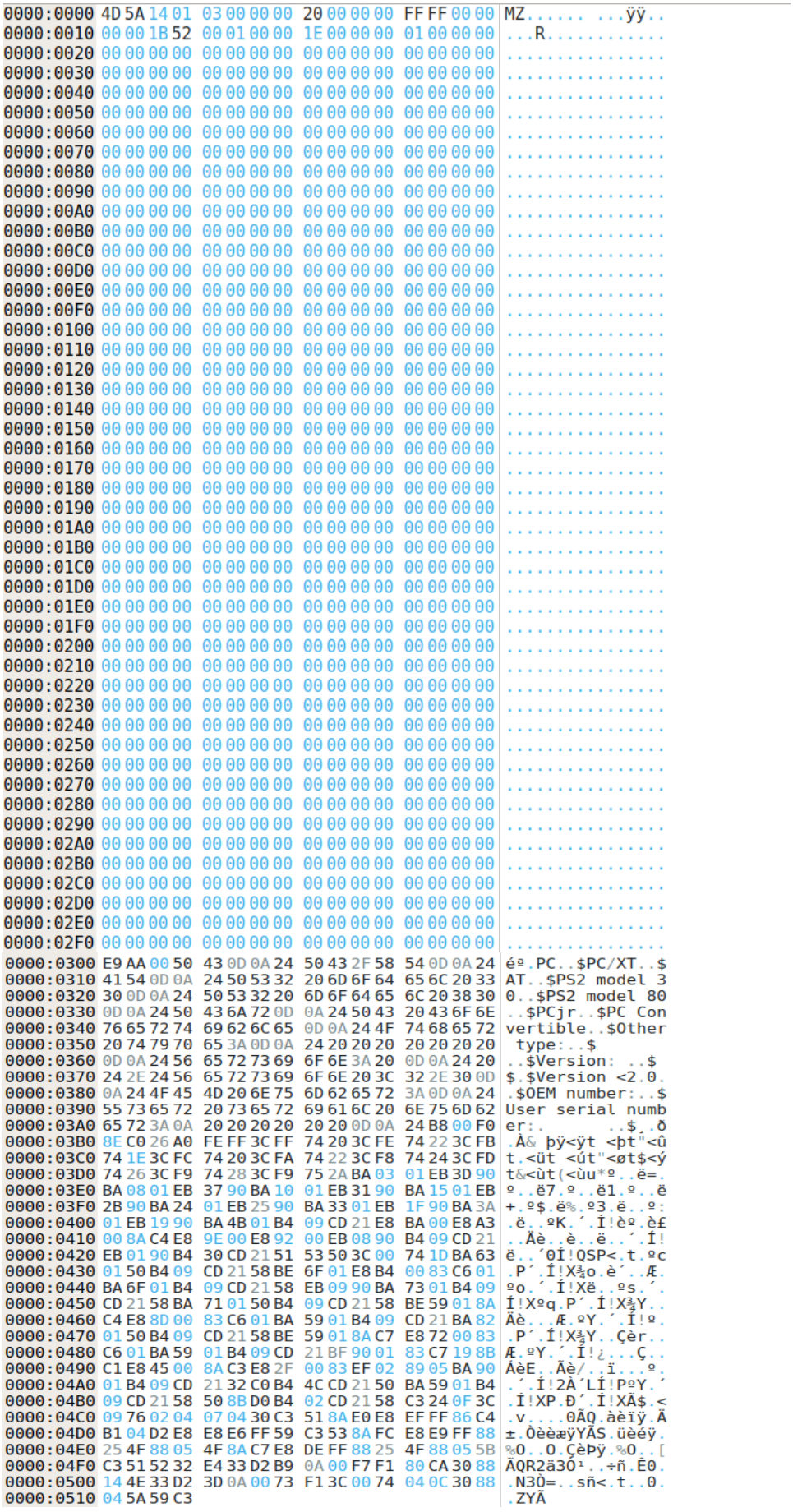

рис. 5 - Файл «плохого» .EXE модуля в 16-м виде

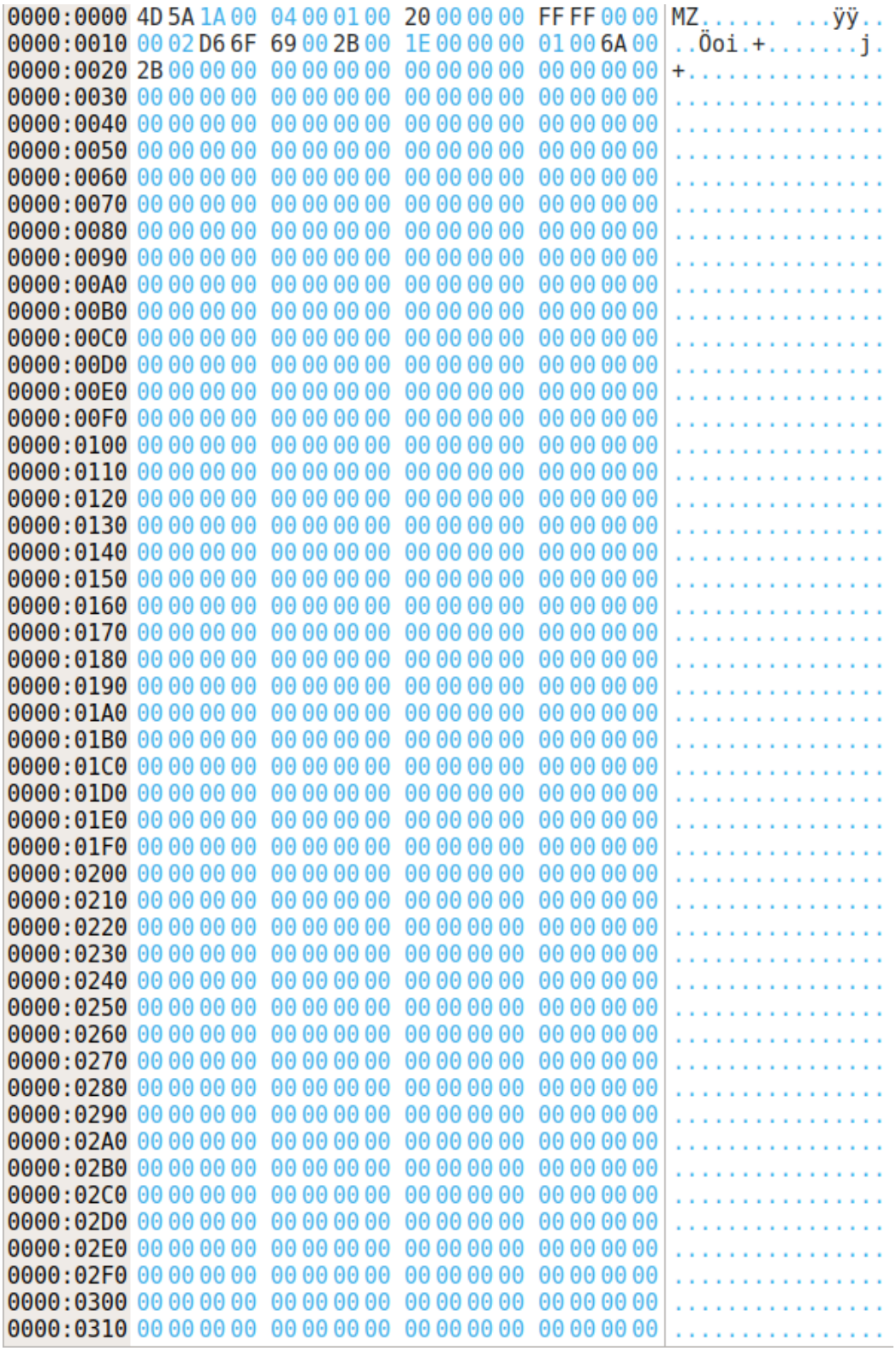

рис. 6 - Файл «хорошего» .EXE модуля в 16-м виде (начало)

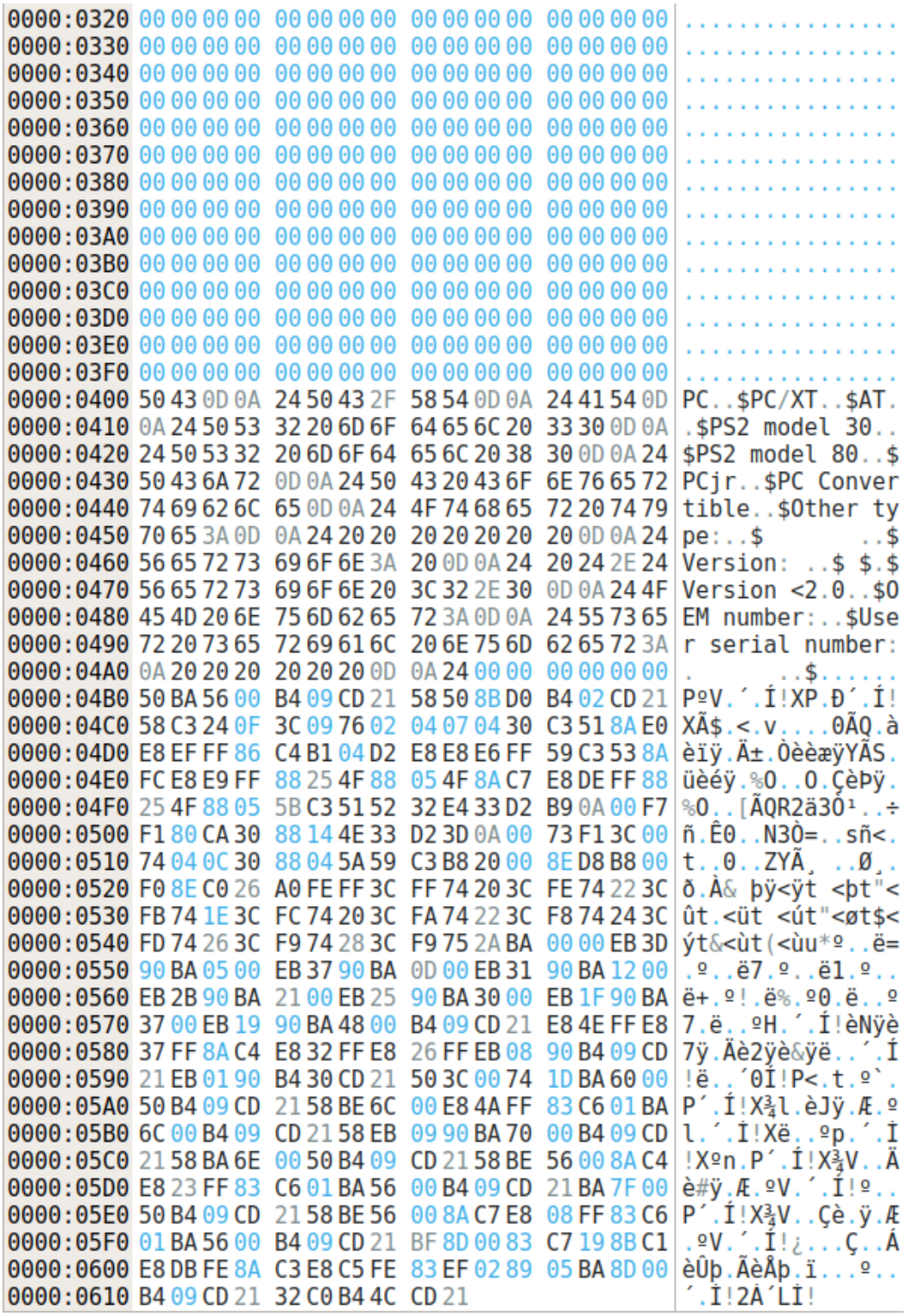

рис. 7 - Файл «хорошего» .EXE модуля в 16-м виде (конец)

Отличия форматов файлов COM и EXE модулей

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

- COM-файл не превышает 64Кб и содержит один сегмент. Код располагается с адреса 0h, при этом в первых 100h байт размещается PSP.

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

- «Плохой» EXE файл, так же, как и COM файла, имеет один сегмент. Код располагается с адреса 300h. С адреса 0 располагается информация для загрузчика, которая образует заголовок.

MZ показывает, что это EXE файл и нужно генерировать 16-битный код (а не 32-битный).

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

- В «хорошем» EXE файле есть сегменты данных, кода и стека, в отличии от «плохого», в котором только один сегмент. С нулевого адреса, так же, как и в “плохом” EXE, располагается заголовок, затем таблица настроек.

В «плохом» файле адресация всегда начинается с адреса 300h.

3. Запуск файла .COM в отладчике TD.EXE показан на рис. 8, запуск «хорошего» .EXE на рис. 9.

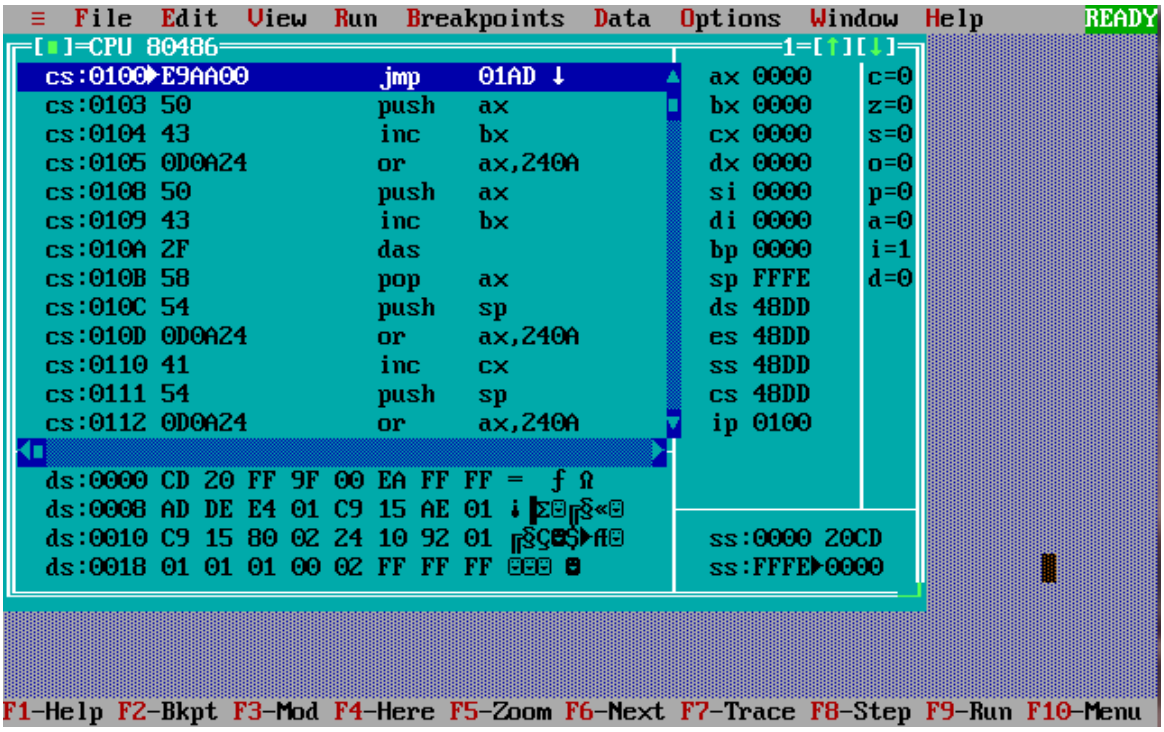

рис. 8 - файл .COM в отладчике

Загрузка COM модуля в основную память

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

- Смещение в сегменте команд равно 100h, сегментные регистры указывают на PSP.

2. Что располагается с адреса 0?

- С адреса 0 до адреса 100h располагается PSP.

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

- Сегментные регистры имеют значения 48DD и указывают на PSP.

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

- Значение регистра SP устанавливается (автоматически) так, чтобы он указывал на последнюю доступную в сегменте ячейку памяти (SP указывает на FFFE). Таким образом программа занимает начало, а стек - конец сегмента.

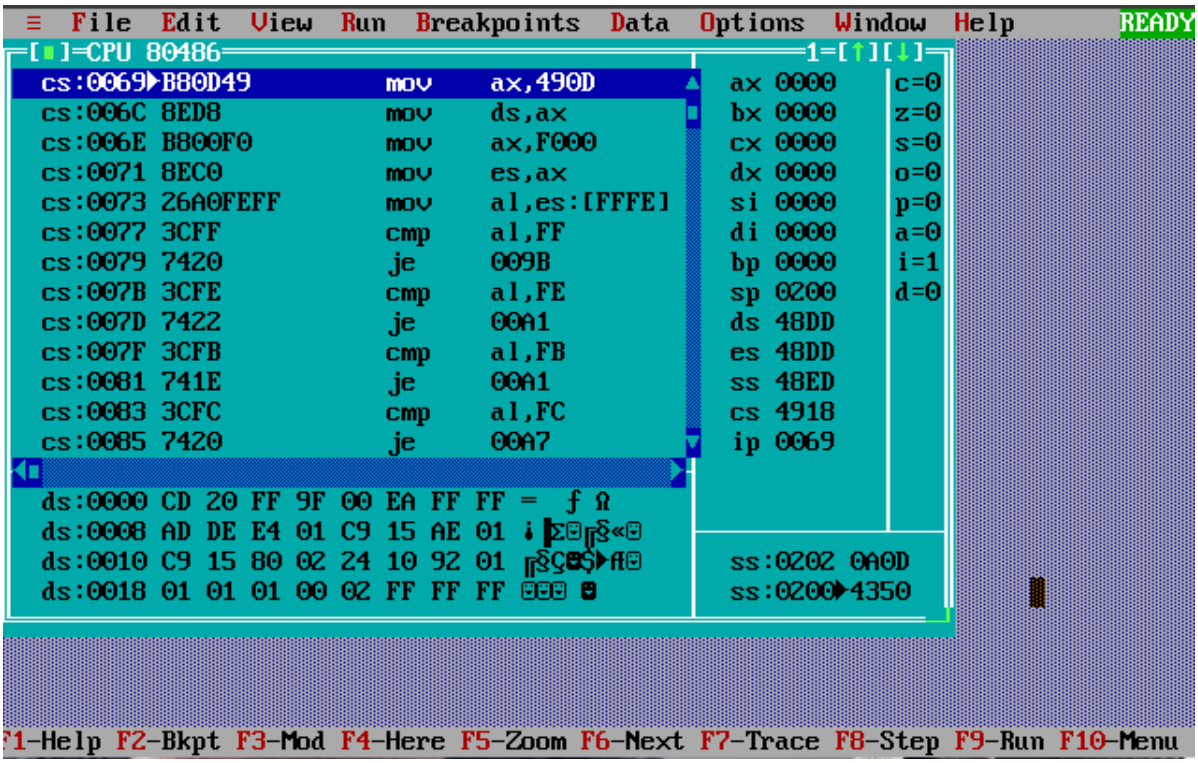

рис 9 – файл «хорошего» .EXE в отладчике

Загрузка «хорошего» EXE модуля в основную память

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

- Значения DS, ES (48DD) устанавливаются на начало PSP, SS (48ED) – на начало сегмента стека, CS (4918) – на начало сегмента команд.

2. На что указывают регистры DS и ES?

- Регистры DS и ES указывают на начало PSP.

3. Как определяется стек?

- Стек определяется при помощи директивы .STACK или при помощи диррективы ASSUME, которая установит сегментый регистр SS на начало сегмента стека.

4. Как определяется точка входа?

- Точка входа определяется при помощи директивы END (за ней идет название функции или метки, с которой нужно начать выполнение программы).

Выводы

В ходе выполенения работы были изучены COM и EXE файлы и их различия. Так же были получены две программы typecom.com и typeexe.exe.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД TYPECOM.ASM

```
TESTPC    SEGMENT
            ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
            ORG 100H ; резервирование места для PSP
START:    JMP BEGIN

PC db  'PC',0DH,0AH,'$'
XT db  'PC/XT',0DH,0AH,'$'
tAT db  'AT',0DH,0AH,'$'
PS2_30 db  'PS2 model 30',0DH,0AH,'$'
PS2_80 db  'PS2 model 80',0DH,0AH,'$'
PCJR db  'PCjr',0DH,0AH,'$'
PC_CONVERTIBLE db  'PC Convertible',0DH,0AH,'$'
OTHER_TYPE db  'Other type:',0DH,0AH,'$'
END_LINE db '       ', 0DH,0AH,'$'

VERSION db  'Version: ',0DH,0AH,'$'
VERS db ' $'
MODIFICATION db  '.$'
VERSION2 db  'Version <2.0',0DH,0AH,'$'
OEM db  'OEM number:',0DH,0AH,'$'
SERIAL_NUMBER db 'User serial number:', 0AH, '      ', 0DH,0AH,'$'


BEGIN:
            mov ax,0F000H
            mov es,ax
            mov al,es:[0FFFEH]

            cmp al, 0FFH
            je itIsPC
            cmp al, 0FEH
            je itIsPC_XT
            cmp al, 0FBH
            je itIsPC_XT
            cmp al, 0FCH
            je itIsAT
            cmp al, 0FAH
            je itIsPS2_30
            cmp al, 0F8H
            je itIsPS2_80
            cmp al, 0FDH
            je itIsPCjr
            cmp al, 0F9H
```

```
            je itIsPCconvertible

            cmp al, 0F9H
            jne itIsOther

;-------------------------------------
; Для вывода типа
itIsPC:
            mov dx, offset PC
            jmp writeType

itIsPC_XT:
            mov dx, offset XT
            jmp writeType

itIsAT:
            mov dx, offset tAT
            jmp writeType

itIsPS2_30:
            mov dx, offset PS2_30
            jmp writeType

itIsPS2_80:
            mov dx, offset PS2_80
            jmp writeType

itIsPCjr:
            mov dx, offset PCJR
            jmp writeType

itIsPCconvertible:
            mov dx, offset PC_CONVERTIBLE
            jmp writeType

itIsOther:
            mov dx, offset OTHER_TYPE
            mov ah, 09h
            int 21h
            call BYTE_TO_HEX
            call PRINT_NUM
            mov al, ah
            call PRINT_NUM
            call PRINT_END_LINE
            jmp OS_VERSION

writeType:
            mov ah, 09h
```

```
            int 21h
            jmp OS_VERSION
;-----------------------------------------


;-----------------------------------------
; Для вывода версии системы

OS_VERSION:
            mov ah, 30h
            int 21h

printVer:
            push ax
            cmp al, 0
            je ver2

            mov dx, offset VERSION
            push ax
            mov ah, 09h
            int 21h
            pop ax

            mov si, offset VERS
            call BYTE_TO_DEC
            add si, 1
            mov dx, offset VERS
            mov ah, 09h
            int 21h
            pop ax
            jmp numMod

ver2:
            mov dx, offset VERSION2
            mov ah, 09h
            int 21h
            pop ax

numMod:
            mov dx, offset MODIFICATION
            push ax
            mov ah, 09h
            int 21h
            pop ax
            mov si, offset END_LINE
            mov al, ah
            call BYTE_TO_DEC
            add si, 1
```

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

numOEM:
            mov dx, offset OEM
            push ax
            mov ah, 09h
            int 21h
            pop ax
            mov si, offset END_LINE
            mov al, bh
            call BYTE_TO_DEC
            add si, 1
            mov dx, offset END_LINE
            mov ah, 09h
            int 21h

serialNumb:

            mov di, offset SERIAL_NUMBER
            add di, 25
            mov ax, cx
            call WRD_TO_HEX
            mov al, bl
            call BYTE_TO_HEX
            sub di, 2
            mov [di], ax
            mov dx, offset SERIAL_NUMBER
            mov ah, 09h
            int 21h

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



;-----------------------------------------

PRINT_END_LINE PROC near
            push ax
            mov dx, offset END_LINE
            mov ah, 09h
            int 21h
            pop ax
PRINT_END_LINE ENDP
```

```
PRINT_NUM PROC near
            ; вывод  al
            push ax
            mov dx, ax
            mov ah, 02h
            int 21h
            pop ax
            ret
PRINT_NUM 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
;байт в AL переводится в два символа шест. 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, 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 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



TESTPC    ENDS
            END START
```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД TYPEEXE.ASM

```
AStack    SEGMENT  STACK
                  DW 100h DUP(?)
AStack    ENDS

DATA  SEGMENT
          PC db  'PC',0DH,0AH,'$'
          XT db  'PC/XT',0DH,0AH,'$'
          tAT db  'AT',0DH,0AH,'$'
          PS2_30 db  'PS2 model 30',0DH,0AH,'$'
          PS2_80 db  'PS2 model 80',0DH,0AH,'$'
          PCJR db  'PCjr',0DH,0AH,'$'
          PC_CONVERTIBLE db  'PC Convertible',0DH,0AH,'$'
          OTHER_TYPE db  'Other type:',0DH,0AH,'$'
          END_LINE db '       ', 0DH,0AH,'$'

          VERSION db  'Version: ',0DH,0AH,'$'
          VERS db ' $'
          MODIFICATION db  '.$'
          VERSION2 db  'Version <2.0',0DH,0AH,'$'
          OEM db  'OEM number:',0DH,0AH,'$'
          SERIAL_NUMBER db 'User serial number:', 0AH, '      ',
0DH,0AH,'$'
DATA ENDS

CODE SEGMENT
   ASSUME CS:CODE,DS:DATA,SS:AStack


;------------------------------
PRINT_END_LINE PROC near
           push ax
           mov dx, offset END_LINE
           mov ah, 09h
           int 21h
           pop ax
PRINT_END_LINE ENDP

PRINT_NUM PROC near
           ; вывод  al
           push ax
           mov dx, ax
           mov ah, 02h
           int 21h
```

```
            pop ax
            ret
PRINT_NUM 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
;байт в AL переводится в два символа шест. 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, 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 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
;--------------------------------


Main PROC FAR
            mov ax, DATA
            mov ds, ax

            mov ax,0F000H
            mov es,ax
            mov al,es:[0FFFEH]

            cmp al, 0FFH
            je itIsPC
            cmp al, 0FEH
            je itIsPC_XT
            cmp al, 0FBH
            je itIsPC_XT
            cmp al, 0FCH
            je itIsAT
            cmp al, 0FAH
            je itIsPS2_30
            cmp al, 0F8H
            je itIsPS2_80
            cmp al, 0FDH
            je itIsPCjr
            cmp al, 0F9H
```

```
            je itIsPCconvertible

            cmp al, 0F9H
            jne itIsOther

;-------------------------------------
; Для вывода типа
itIsPC:
            mov dx, offset PC
            jmp writeType

itIsPC_XT:
            mov dx, offset XT
            jmp writeType

itIsAT:
            mov dx, offset tAT
            jmp writeType

itIsPS2_30:
            mov dx, offset PS2_30
            jmp writeType

itIsPS2_80:
            mov dx, offset PS2_80
            jmp writeType

itIsPCjr:
            mov dx, offset PCJR
            jmp writeType

itIsPCconvertible:
            mov dx, offset PC_CONVERTIBLE
            jmp writeType

itIsOther:
            mov dx, offset OTHER_TYPE
            mov ah, 09h
            int 21h
            call BYTE_TO_HEX
            call PRINT_NUM
            mov al, ah
            call PRINT_NUM
            call PRINT_END_LINE
            jmp OS_VERSION

writeType:
            mov ah, 09h
```

```
            int 21h
            jmp OS_VERSION
;-----------------------------------------


;-----------------------------------------
; Для вывода версии системы

OS_VERSION:
            mov ah, 30h
            int 21h

printVer:
            push ax
            cmp al, 0
            je ver2

            mov dx, offset VERSION
            push ax
            mov ah, 09h
            int 21h
            pop ax

            mov si, offset VERS
            call BYTE_TO_DEC
            add si, 1
            mov dx, offset VERS
            mov ah, 09h
            int 21h
            pop ax
            jmp numMod

ver2:
            mov dx, offset VERSION2
            mov ah, 09h
            int 21h
            pop ax

numMod:
            mov dx, offset MODIFICATION
            push ax
            mov ah, 09h
            int 21h
            pop ax
            mov si, offset END_LINE
            mov al, ah
            call BYTE_TO_DEC
            add si, 1
```

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

numOEM:
            mov dx, offset OEM
            push ax
            mov ah, 09h
            int 21h
            pop ax
            mov si, offset END_LINE
            mov al, bh
            call BYTE_TO_DEC
            add si, 1
            mov dx, offset END_LINE
            mov ah, 09h
            int 21h

serialNumb:

            mov di, offset SERIAL_NUMBER
            add di, 25
            mov ax, cx
            call WRD_TO_HEX
            mov al, bl
            call BYTE_TO_HEX
            sub di, 2
            mov [di], ax
            mov dx, offset SERIAL_NUMBER
            mov ah, 09h
            int 21h

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


Main ENDP
CODE ENDS
      END Main
```