

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. 0382

Афанасьев Н. С.

Преподаватели

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследование различий в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

Задание.

Написать, отладить и сравнить .COM и .EXE модули, которые определяют тип PC и версию системы.

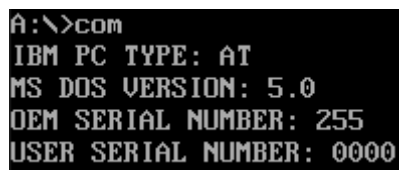
Выполнение работы.

При работе были использованы/созданы следующие процедуры (одинаковые для обоих .COM и .EXE модуля):

- TETR_TO_HEX, BYTE_TO_HEX, WRD_TO_HEX, BYTE_TO_DEC – процедуры, описанные в шаблоне, для перевода двоичных кодов в символы шестнадцатеричных чисел и десятичное число.
- PRINT – процедура для вывода строки, отступ на которую содержится в DX, на экран, используя функцию 09h прерывания 21h.
- PC_TYPE – процедура для вывода типа ПК, получаемого считыванием предпоследнего байта ROM BIOS (по адресу 0F000:0FFFEh). Далее процедура соотносит численное значение типа известным названиям, а если такого нету, то конвертирует число в строку. Наконец, строковое значение типа выводится с помощью процедуры PRINT.
- SYSTEM_VER – процедура для вывода версии MS DOS, а также серийных номеров OEM и пользователя. Вся необходимая информация получается при вызове функции 30h прерывания 21h. После вызова функции в AL содержится номер основной версии (если версия раньше 2.0 то AL=0), а в AH – номер модификации. В BH хранится серийный номер OEM, однако по неизвестным причинам это значение напрямую зависит от значения в AL до вызова функции (если AL = 0, то значение серийного номера будет 255 (FF), если 1 – то 16, иначе 0). В BL и CX хранится 3-байтовый серийный номер

пользователя. Все полученные значения конвертируются в строки и выводятся на экран.

В модуле .COM объявляется один сегмент (так как вся информация в этом типе содержится в одном сегменте). Директивой ORG 100h устанавливаем CS:IP на конец PSP. Далее объявляем необходимые нам данные – строковые значения типов ПК и связанные сообщения. После описания вышеперечисленных процедур запускаем процедуры PC_TYPE и SYSTEM_VER, после чего завершаем работу через функцию 4Ch прерывания 21h. Ниже приведён результат работы:



```
A:\>com
IBM PC TYPE: AT
MS DOS VERSION: 5.0
OEM SERIAL NUMBER: 255
USER SERIAL NUMBER: 0000
```

Рис.1 – выполнение .COM модуля

Если из исходного текста .COM модуля сделать .EXE модуль, то из-за разности в строении модулей программа будет работать некорректно:

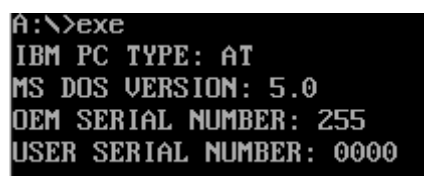


```
A:\>com.exe

5 0
щ(IBM PC TYPE:
щ(IBM PC TYPE:
щ(IBM PC TYPE:
TYPE:
```

Рис.2 – выполнение «плохого» .EXE модуля

При написании «хорошего» .EXE модуля производим следующие изменения: делим данные, стек и код на сегменты, вводим главную процедуру – начало выполнения программы, устанавливаем DS на сегмент данных и убираем директиву ORG, так как CS:IP изначально указывает на конец PSP. Ниже приведён результат работы:



```
A:\>exe
IBM PC TYPE: AT
MS DOS VERSION: 5.0
OEM SERIAL NUMBER: 255
USER SERIAL NUMBER: 0000
```

Рис.3 – выполнение «хорошего» .EXE модуля

Вопросы.

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

– 1 сегмент, не больше

2) EXE-программа?

– 1 обязательный CS + DS, SS, ES

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

– ORG 100h, чтобы установить CS:IP на конец PSP, и ASSUME для указания сегмента кода и данных на один сегмент.

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

– Нет, нельзя использовать команды с указанием сегментов, так как в .COM модулях отсутствует таблица настройки, и адреса сегментных регистров определяются при запуске программы, а не при линковке.

Отличия форматов файлов COM и EXE модулей

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

– PSP, код, данные и стек в одном сегменте. Код располагается с нулевого адреса (но устанавливается смещение 100h на конец PSP)

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	e9	28	02	49	42	4d	20	50	43	20	54	59	50	46	3a	20	8(.IBM PC TYPE:
00000010	20	20	0d	0a	24	49	42	4d	20	50	43	20	54	59	50	45	..\$IBM PC TYPE
00000020	3a	20	50	43	0d	0a	24	49	42	4d	20	50	43	20	54	59	: PC..\$IBM PC TY
00000030	50	45	3a	20	50	43	2f	58	54	0d	0a	24	49	42	4d	20	PE: PC/XT..\$IBM
00000040	50	43	20	54	59	50	45	3a	20	41	54	0d	0a	24	49	42	PC TYPE: AT..\$IB
00000050	4d	20	50	43	20	54	59	50	45	3a	20	50	53	32	20	4d	M PC TYPE: PS2 M
00000060	ef	64	65	6c	20	33	30	0d	0a	24	49	42	4d	20	50	43	odel 30..\$IBM PC
00000070	20	54	59	50	45	3a	20	50	53	32	20	4d	ef	64	65	6c	TYPE: PS2 Model
00000080	20	35	30	20	ef	72	20	36	30	0d	0a	24	49	42	4d	20	50 or 60..\$IBM
00000090	50	43	20	54	59	50	45	3a	20	50	53	32	20	4d	ef	64	PC TYPE: PS2 Mod
000000a0	65	6c	20	39	30	0d	0a	24	49	42	4d	20	50	43	20	54	e1 80..\$IBM PC T
000000b0	59	50	45	3a	20	50	43	6a	72	0d	0a	24	49	42	4d	20	YPE: PCjr..\$IBM
000000c0	50	43	20	54	59	50	45	3a	20	50	43	20	43	6f	6e	76	PC TYPE: PC Conv
000000d0	65	72	74	69	62	6c	65	0d	0a	24	4d	53	20	44	4f	53	ertible: SMS DOS
000000e0	20	56	45	52	53	49	4f	4e	3a	20	20	2e	20	20	0d	0a	VERSION: ..
000000f0	24	4f	45	4d	20	53	45	52	49	41	4c	20	4e	55	4d	42	\$ORM SERIAL NUMB
00000100	45	52	3a	20	20	20	20	0d	0a	24	55	53	45	52	20	53	ER: ..\$USER S
00000110	45	52	49	41	4c	20	4e	55	4d	42	45	52	3a	20	20	20	ERIAL NUMBER:
00000120	20	0d	0a	24	24	0f	3c	09	76	02	04	07	04	30	c3	51	...\$.<.v.....0TQ
00000130	8a	e0	e8	ef	ff	86	c4	b1	04	d2	e8	e8	e6	ff	59	c3	8asm*7дз.ТюсюИТ
00000140	53	8a	fc	e9	ef	88	25	4f	88	05	4f	8a	c7	e8	de	58	S8888888888888888
00000150	ff	88	25	4f	88	05	5b	c3	51	52	32	e4	33	d2	b9	0a	8888888888888888
00000160	00	f7	f1	80	ca	30	88	14	4e	33	d2	3d	0a	00	73	f1	8888888888888888
00000170	3c	00	74	04	0c	30	88	04	5a	59	c3	50	b4	09	cd	21	8888888888888888
00000180	58	c3	50	52	06	b8	00	f0	8a	c0	26	a0	fe	ff	3c	ff	8888888888888888
00000190	ba	15	01	74	42	3c	fe	ba	27	01	74	3b	3c	fb	74	37	8888888888888888
000001a0	3c	fc	ba	3c	01	74	30	3c	fa	ba	4e	01	74	29	3c	fc	8888888888888888
000001b0	ba	6a	01	74	22	3c	f8	ba	8c	01	74	1b	3c	fd	ba	a8	8888888888888888
000001c0	01	74	14	3c	f9	ba	bc	01	74	0d	e8	62	ff	a2	10	01	8888888888888888
000001d0	88	26	11	01	ba	03	01	e8	al	ff	07	5a	58	c3	50	53	8888888888888888
000001e0	51	57	56	2a	c0	b4	30	cd	21	be	da	01	83	c6	10	e8	8888888888888888
000001f0	66	ff	8a	c4	83	c6	03	e8	5e	ff	ba	da	01	e8	7b	ff	8888888888888888
00000200	8a	c7	ba	f1	01	8b	f2	83	c6	15	e8	4b	ff	e8	6b	ff	8888888888888888
00000210	8b	c1	ba	0a	02	8b	fa	83	c7	17	e8	23	ff	8a	c3	e8	8888888888888888
00000220	0d	ff	e8	56	ff	5e	5f	59	5b	58	c3	e8	54	ff	e8	ad	8888888888888888
00000230	ff	32	c0	b4	4c	cd	21										8888888888888888

Рис.4 – .COM модуль в 16-тиричном виде

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

– С нулевого адреса идёт таблица настроек, с адреса 300h (таблица настроек + смещение) идут код, данные и стек в одном сегменте.

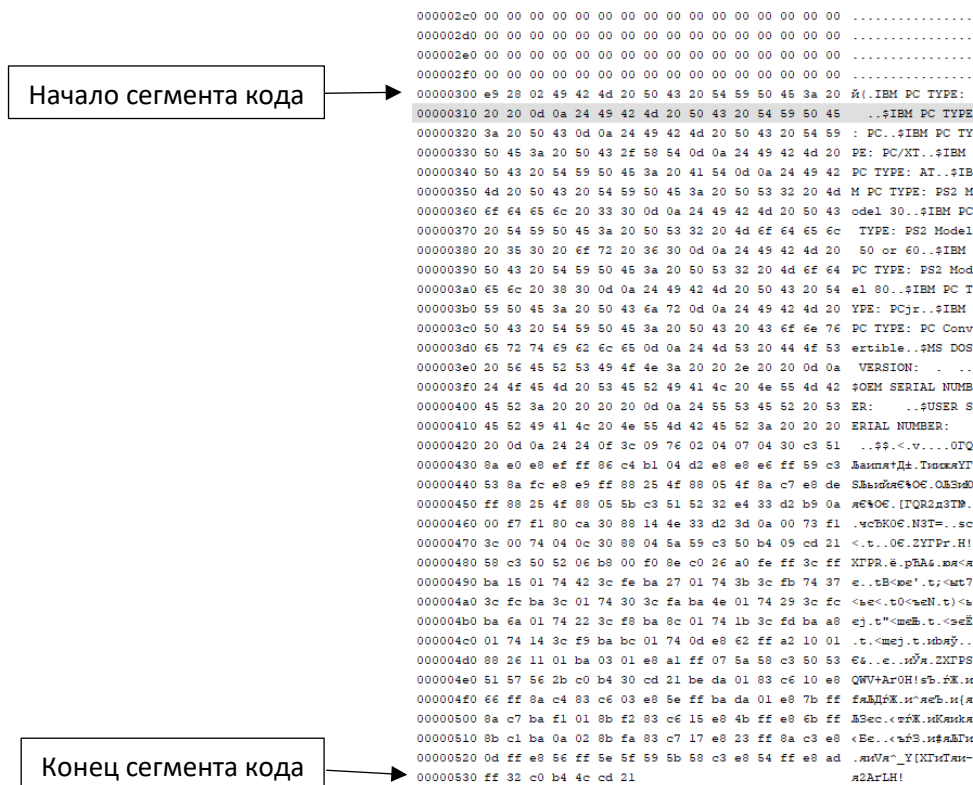


Рис.5 – «плохой» .EXE модуль в 16-тиричном виде

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

– Данные, стек и код расположены в разных сегментах, нет ограничения в 64Кб, код начинается без отступа, после таблицы настроек и выделенного места под данные (см. рис. 6).

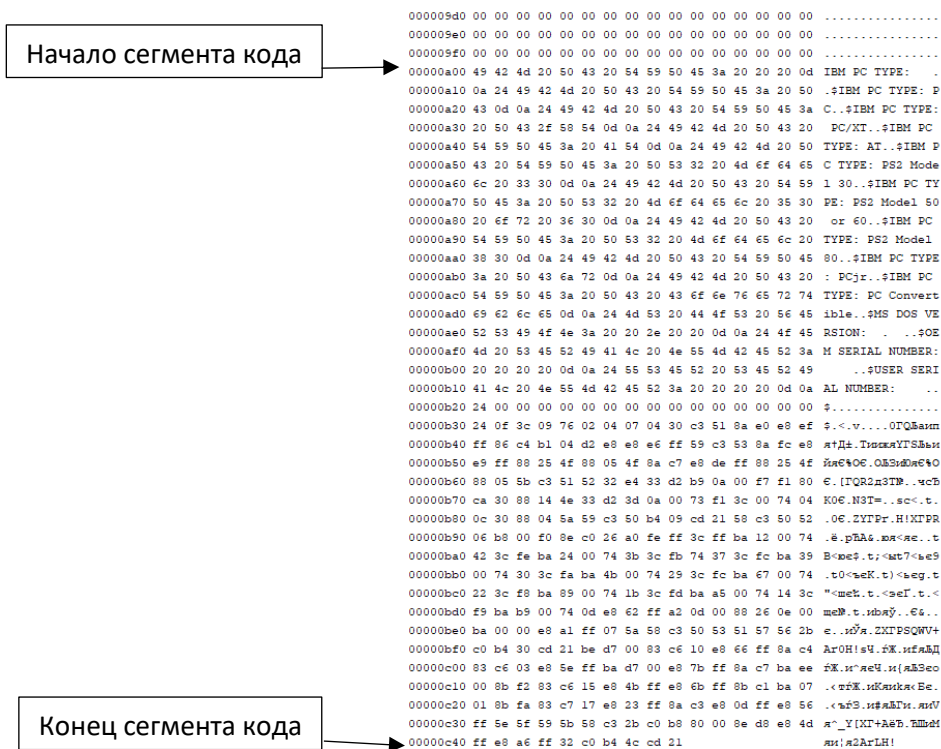


Рис.6 – «хороший» .EXE модуль в 16-тиричном виде

Загрузка COM модуля в основную память

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

– Копия загрузочного модуля переносится загрузчиком из файла на диске в раздел ОП, в который помещается модуль. Сегментные регистры устанавливаются на начало PSP (в данном случае 48DD), а SP – на конец модуля (FFFE, так как ограничение в 64Кб и на чётной позиции), в IP записывается адрес начала кода – 100h.

2) Что располагается с адреса 0?

– PSP

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

– Сегментные регистры указывают на начало PSP.

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

– Стек находится в одном сегменте со всем остальным, и поэтому область памяти под него составляет 64Kб. Адреса от 0h до FFFh. SP (FFFFh) указывает на конец стека, SS (0h) – на начало.

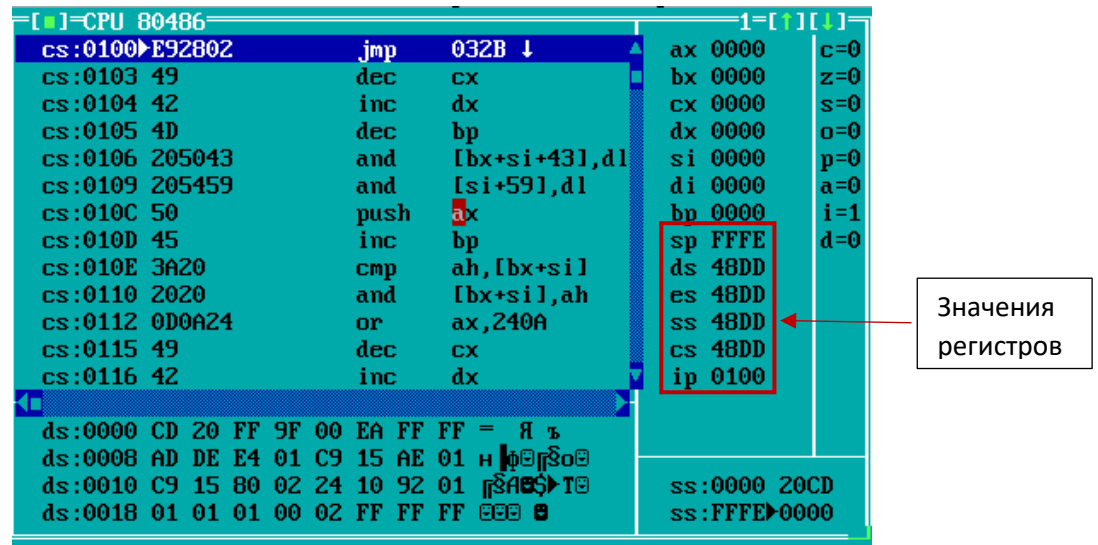


Рис.7 – Загрузка .COM модуля

Загрузка «хорошего» EXE модуля в основную память

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

– Так же подгружается в ОП. Преобразуются адреса согласно таблице настройки. DS и ES устанавливаются на начало PSP (в данном случае 48DD), SS и CS на начала своих собственных сегментов (в данном случае 48ED и 4980 соответственно). SP указывает на начало стека (в данном случае 0800). IP устанавливается согласно метке после директивы END в конце программы (в данном случае 0107), либо на начало CS.

2) *На что указывают регистры DS и ES?*

– На PSP

3) *Как определяется стек?*

– Стек в отдельном сегменте, ограничен этим сегментом. SS и SP указывают на границы этого сегмента.

4) *Как определяется точка входа?*

– IP устанавливается согласно метке после директивы END в конце программы, либо на начало CS.

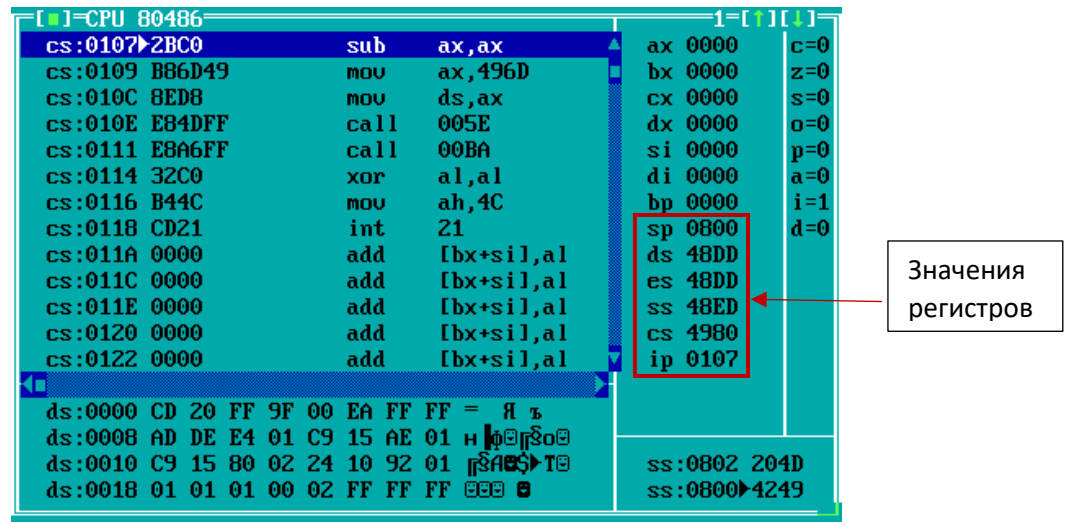


Рис.8 – Загрузка .EXE модуля

Выводы.

Были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: com.asm

```
CODE SEGMENT
    ASSUME CS:CODE, DS:CODE, ES:NOTHING, SS:NOTHING
    ORG 100H
    START: jmp BEGIN

    TYPE0 DB 'IBM PC TYPE:      ',0DH,0AH,'$'
    TYPE1 DB 'IBM PC TYPE: PC',0DH,0AH,'$'
    TYPE2 DB 'IBM PC TYPE: PC/XT',0DH,0AH,'$'
    TYPE3 DB 'IBM PC TYPE: AT',0DH,0AH,'$'
    TYPE4 DB 'IBM PC TYPE: PS2 Model 30',0DH,0AH,'$'
    TYPE5 DB 'IBM PC TYPE: PS2 Model 50 or 60',0DH,0AH,'$'
    TYPE6 DB 'IBM PC TYPE: PS2 Model 80',0DH,0AH,'$'
    TYPE7 DB 'IBM PC TYPE: PCjr',0DH,0AH,'$'
    TYPE8 DB 'IBM PC TYPE: PC Convertible',0DH,0AH,'$'
    VER    DB 'MS DOS VERSION:  . ',0DH,0AH,'$'
    OEM    DB 'OEM SERIAL NUMBER:      ',0DH,0AH,'$'
    USER   DB 'USER SERIAL NUMBER:      ',0DH,0AH,'$'

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
    ; input: AL, output: AX
    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
    ; input: AX, output: DI
    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
; input: AL, output: SI
        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

PC_TYPE PROC NEAR
        push AX
        push DX
        push ES
        mov AX, 0F000h
        mov ES, AX
        mov AL, ES:[0FFFEh]

        cmp AL, 0FFh
        mov DX, offset TYPE1
        je result
        cmp AL, 0FEh
        mov DX, offset TYPE2
        je result
        cmp AL, 0FBh
        je result
        cmp AL, 0FCh
        mov DX, offset TYPE3

```

```

        je result
        cmp AL, 0FAh
        mov DX, offset TYPE4
        je result
        cmp AL, 0FCh
        mov DX, offset TYPE5
        je result
        cmp AL, 0F8h
        mov DX, offset TYPE6
        je result
        cmp AL, 0FDh
        mov DX, offset TYPE7
        je result
        cmp AL, 0F9h
        mov DX, offset TYPE8
        je result

        call BYTE_TO_HEX
        mov TYPE0[13], AL
        mov TYPE0[14], AH
        mov DX, offset TYPE0

result:
        call PRINT
        pop ES
        pop DX
        pop AX
        ret
PC_TYPE ENDP

SYSTEM_VER PROC NEAR
        push AX
        push BX
        push CX
        push DI
        push SI

        sub AX, AX
        mov AH, 30h
        int 21h

        ; Version
        mov SI, offset VER
        add SI, 16
        call BYTE_TO_DEC
        mov AL, AH
        add SI, 3
        call BYTE_TO_DEC
        mov DX, offset VER
        call PRINT

        ; OEM Serial Number
        mov AL, BH
        mov DX, offset OEM
        mov SI, DX
        add SI, 21
        call BYTE_TO_DEC
        call PRINT

```

```

        ; User Serial Number
        mov AX, CX
        mov DX, offset USER
        mov DI, DX
        add DI, 23
        call WRD_TO_HEX
        mov AL, BL
        call BYTE_TO_HEX
        call PRINT

        pop SI
        pop DI
        pop CX
        pop BX
        pop AX
        ret
SYSTEM_VER ENDP

BEGIN:
        call PC_TYPE
        call SYSTEM_VER
        xor AL, AL
        mov AH, 4Ch
        int 21H
CODE ENDS
END START

```

Название файла: exe.asm

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

```

STACK SEGMENT STACK
    DW 1024 DUP(?)
STACK ENDS

```

```

DATA SEGMENT
    TYPE0 DB 'IBM PC TYPE:    ', 0DH, 0AH, '$'
    TYPE1 DB 'IBM PC TYPE: PC', 0DH, 0AH, '$'
    TYPE2 DB 'IBM PC TYPE: PC/XT', 0DH, 0AH, '$'
    TYPE3 DB 'IBM PC TYPE: AT', 0DH, 0AH, '$'
    TYPE4 DB 'IBM PC TYPE: PS2 Model 30', 0DH, 0AH, '$'
    TYPE5 DB 'IBM PC TYPE: PS2 Model 50 or 60', 0DH, 0AH, '$'
    TYPE6 DB 'IBM PC TYPE: PS2 Model 80', 0DH, 0AH, '$'
    TYPE7 DB 'IBM PC TYPE: PCjr', 0DH, 0AH, '$'
    TYPE8 DB 'IBM PC TYPE: PC Convertible', 0DH, 0AH, '$'
    VER    DB 'MS DOS VERSION:  . ', 0DH, 0AH, '$'
    OEM     DB 'OEM SERIAL NUMBER:    ', 0DH, 0AH, '$'
    USER    DB 'USER SERIAL NUMBER:    ', 0DH, 0AH, '$'
DATA ENDS

```

```

CODE SEGMENT
    TETR_TO_HEX PROC NEAR
        and AL, 0Fh
        cmp AL, 09
        jbe next
        add AL, 07
    next: add AL, 30h
    TETR_TO_HEX ENDP

```

```

        ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC NEAR
; input: AL, output: AX
    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
; input: AX, output: DI
    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
; input: AL, output: SI
    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

PRINT PROC NEAR

```

```

        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
PRINT ENDP

PC_TYPE PROC NEAR
    push AX
    push DX
    push ES
    mov AX, 0F000h
    mov ES, AX
    mov AL, ES:[0FFFEh]

    cmp AL, 0FFh
    mov DX, offset TYPE1
    je result
    cmp AL, 0FEh
    mov DX, offset TYPE2
    je result
    cmp AL, 0FBh
    je result
    cmp AL, 0FCh
    mov DX, offset TYPE3
    je result
    cmp AL, 0FAh
    mov DX, offset TYPE4
    je result
    cmp AL, 0FCh
    mov DX, offset TYPE5
    je result
    cmp AL, 0F8h
    mov DX, offset TYPE6
    je result
    cmp AL, 0FDh
    mov DX, offset TYPE7
    je result
    cmp AL, 0F9h
    mov DX, offset TYPE8
    je result

    call BYTE_TO_HEX
    mov TYPE0[13], AL
    mov TYPE0[14], AH
    mov DX, offset TYPE0

    result:
    call PRINT
    pop ES
    pop DX
    pop AX
    ret
PC_TYPE ENDP

SYSTEM_VER PROC NEAR
    push AX
    push BX

```

```

    push CX
    push DI
    push SI

    sub AX,AX
    mov AH, 30h
    int 21h

    ; Version
    mov SI, offset VER
    add SI, 16
    call BYTE_TO_DEC
    mov AL, AH
    add SI, 3
    call BYTE_TO_DEC
    mov DX, offset VER
    call PRINT

    ; OEM Serial Number
    mov AL, BH
    mov DX, offset OEM
    mov SI, DX
    add SI, 21
    call BYTE_TO_DEC
    call PRINT

    ; User Serial Number
    mov AX, CX
    mov DX, offset USER
    mov DI, DX
    add DI, 23
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    call PRINT

    pop SI
    pop DI
    pop CX
    pop BX
    pop AX
    final:
    ret
SYSTEM_VER ENDP

main PROC FAR
    sub AX, AX
    mov AX, DATA
    mov DS, AX

    call PC_TYPE
    call SYSTEM_VER
    xor AL,AL
    mov AH,4Ch
    int 21H
main ENDP
CODE ENDS
END main

```

