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. 9382 _____ Кузьмин Д. И.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург

2021

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.

Шаг 1. Напишите текст исходного .COM модуля, который определяет тип PC и версию системы. Это довольно простая задача и для тех, кто уже имеет опыт программирования на ассемблере, это будет небольшой разминкой. Для тех, кто раньше не сталкивался с программированием на ассемблере, это неплохая задача для первого опыта. За основу возьмите шаблон, приведенный в разделе «Основные сведения». Необходимые сведения о том, как извлечь требуемую информацию, представлены в следующем разделе. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип PC и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения. Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx - номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM и серийным номером пользователя. Полученные строки выводятся на экран. Отладьте полученный исходный модуль. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

Шаг 2. Напишите текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» .EXE.

Шаг 3. Сравните исходные тексты для .COM и .EXE модулей. Ответьте на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустите FAR и откройте (F3/F4) файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» .EXE и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Откройте отладчик TD.EXE и загрузите .COM. Ответьте на контрольные вопросы «Загрузка COM модуля в основную память». Представьте в отчете план загрузки модуля .COM в основную память.

Шаг 6. Откройте отладчик TD.EXE и загрузите «хороший» .EXE. Ответьте на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Шаг 7. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

Выполнение работы.

1. Первым шагом было создание исходного кода для com файла. В нем содержится один сегмент TESTPC. С помощью ASSUME он задается в соответствие с сегментными регистрами CS и DS.
2. Далее следует директива org 100h для отступа в 256 байт.
3. Далее были созданы участки памяти для строк, содержащих информацию о системе.
4. Затем созданы процедуры для загрузки требуемой информации и ее Вывода.
5. Далее был создан исходный код для EXE файла. В нем сегмент кода и данных разделены, а также инициализирован сегмент стека.

Демонстрацию работы программы см. в приложении Б

Исходный код см. в приложении А

Контрольные вопросы.

1.1. Сколько сегментов должна содержать COM-программа?

COM программа должна содержать единственный сегмент, в котором размещаются программный код и данные.

1.2 .EXE-программа?

EXE программа должна иметь минимум один сегмент.

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

ORG 100h для смещения на 256 байт. ASSUME для задания соответствия между единственным сегментом и сегментными регистрами.

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

Нельзя использовать команды, где используются перемещаемые сегменты (relocatable segment), например содержащие оператор seg, потому что нет таблицы настройки. Таблица настройки содержит смещения, которые применяются к адресам при загрузке программы в память.

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

Файл COM состоит из единственного сегмента, хранящего код и данные. Код располагается с адреса 0. Максимальный размер файла – 64 кб.

2.2. Какова структура файла плохого EXE? С какого адреса располагается код? Что располагается с адреса 0?

В плохом EXE код и данные хранятся в одном сегменте. С адреса 0 располагается заголовок, содержащий информацию о файле. Код располагается с адреса 300h.

2.3. Какова структура "хорошего" EXE? Чем он отличается от файла "плохого" EXE?

В "хорошем" EXE код и данные хранятся в разных сегментах. Есть заголовок и таблица настройки, содержащая информацию об адресах, которые изменятся при загрузке программы в память. В «плохом» EXE код и данные не разделяются на сегменты, также «плохой» EXE не учитывает стек и смещается

на 100h. Итоговое смещение будет 300h, в то время как в «хорошем» EXE стек располагается в отдельном сегменте и нет дополнительного смещения 100h. Итоговое смещение – 280h.

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

Система заполняет зарезервированные 256 байт для PSP. Затем инициализирует сегментные регистры, которые все указывают на начало PSP. Регистр IP инициализируется 100h, то есть первым байтом после PSP; с этого места располагается код. SP инициализируется FFFEh. Программе выделяется 64кб.

3.2. Что располагается с адреса 0?

Программный префикс PSP, который загружает система.

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

Сегментные регистры имеют одинаковые значения равные 48DD. Они указывают на начало PSP.

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

Стек определяется автоматически и располагается с адреса 0h до FFFEh.

4.1. Как загружается "хороший" EXE? Какие значения имеют сегментные регистры?

К началу программы пристраивается PSP размером 256 байт. Происходит настройка адресов сегментов в соответствии с таблицей настройки. Затем инициализируются сегментные регистры. DS и ES указывают на PSP – 48DD, SS - на начало сегмента стека 48ED, CS - на начало сегмента кода 4907. В указатель IP загружается точка входа в программу, в SP - конец сегмента стека.

4.2. На что указывают регистры DS и ES?

DS и ES указывают на начало PSP.

4.3. Как определяется стек?

Размер стека задается в программе. При загрузке в память В SS загружается начало сегмента. В SP - конец.

4.4. Как определяется точка входа?

Точка входа определяется операндом директивы END.

Выводы.

Были изучены различия в структурах исходных текстов модулей типов .COM и .EXE, структуры файлов загрузочных модулей и способы их загрузки в основную память.

ПРИЛОЖЕНИЕ А. ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

Рис 1. - Файл загрузочного модуля COM.

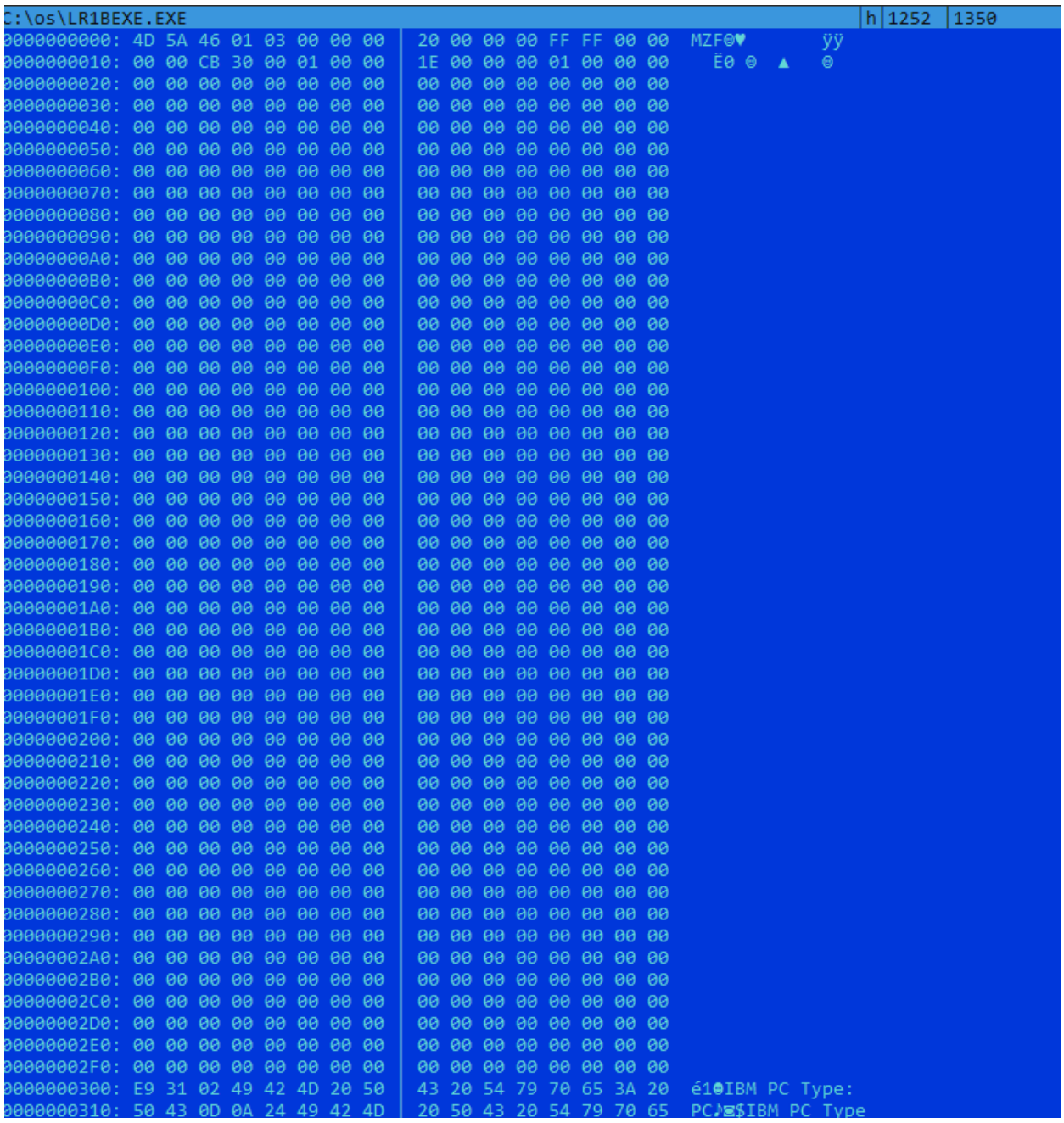

Рис. 2 - Файл загрузочного модуля «плохого» EXE

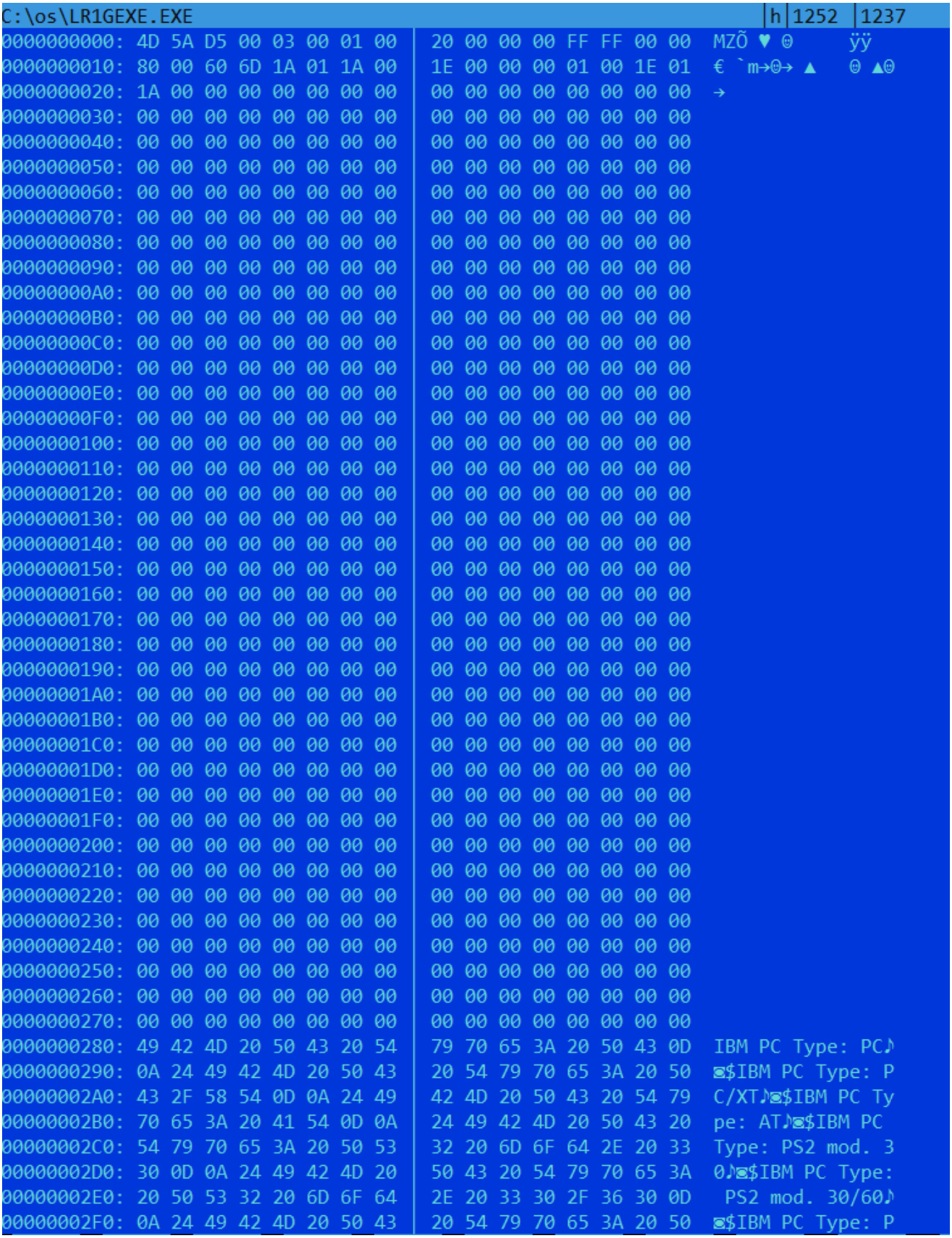

Рис. 3 - Файл загрузочного модуля «хорошего» EXE

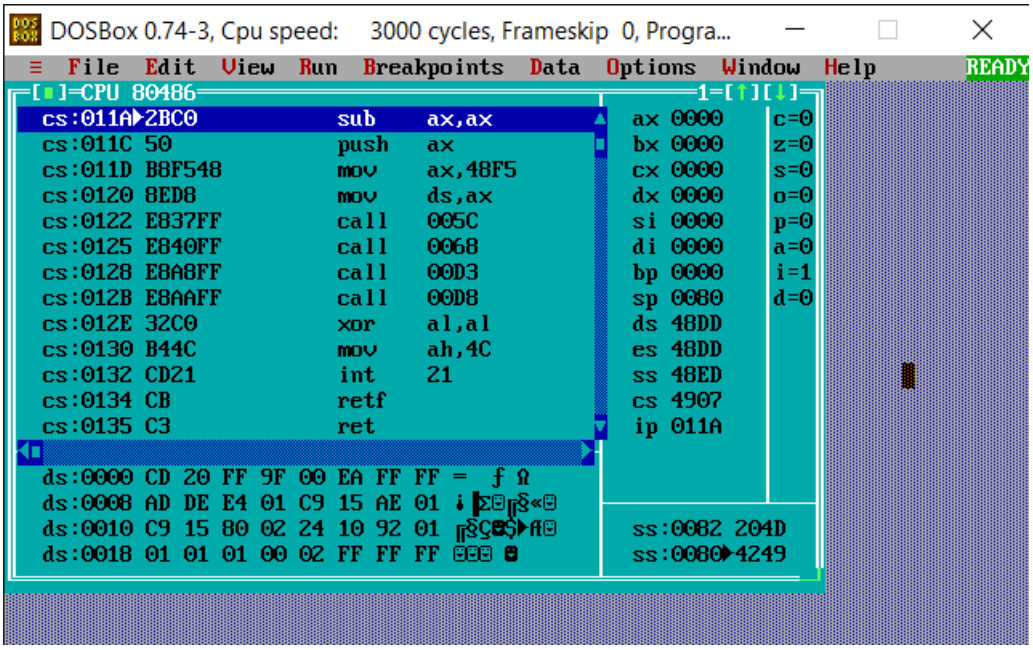

Рис. 4 - Загрузка «хорошего» EXE

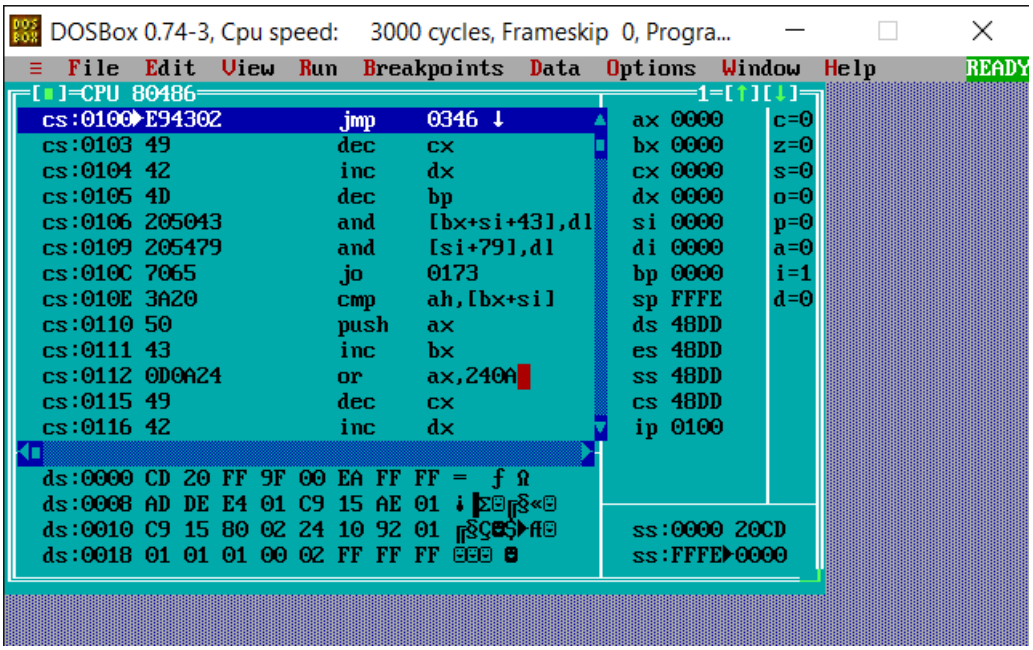

Рис. 5 - Загрузка COM

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

Файл lr1com.asm

```
TESTPC SEGMENT
 ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
 ORG 100H
START: JMP BEGIN
; Данные
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, '$'
type4 db 'IBM PC Type: PS2 mod. 30', 0DH, 0AH, '$'
type5 db 'IBM PC Type: PS2 mod. 30/60', 0DH, 0AH, '$'
type6 db 'IBM PC Type: PS2 mod. 60', 0DH, 0AH, '$'
type7 db 'IBM PC Type: PCjr', 0DH, 0AH, '$'
type8 db 'IBM PC Type: PC Convertible', 0DH, 0AH, '$'
notype db 'Unknown type:  ', 0DH, 0AH, '$'
dos_ver_string db 'MS-DOS Version:  .  ', 0DH, 0AH, '$'
oem_string db 'OEM Number:   ', 0DH, 0AH, '$'
user_number_string db 'User serial number:        ', 0DH, 0AH, '$'
; Процедуры
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
;--------------------------------
WRITE_MSG PROC near
                mov AH, 09h
                int 21h
                ret
WRITE_MSG 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, в AH - младшая
                pop CX
                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 10чную с/с, SI - адрес 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
;--------------------------------
GET_PC_TYPE PROC near
;Вывод типа IBM PC
                push BX
                mov BX,0F000h
                mov ES,BX
                mov AL,ES:[0FFFEh]
                pop BX
                ret
GET_PC_TYPE ENDP
;--------------------------------
PRINT_PC_TYPE PROC near
                cmp AL, 0FFh
                je type1_found
                cmp AL, 0FEh
                je type2_found
                cmp AL, 0FCh
                je type3_found
                cmp AL, 0FAh
                je type4_found
                cmp AL, 0FCh
                je type5_found
                cmp AL, 0F8h
                je type6_found
```

```
                cmp AL, 0FDh
                je type7_found
                cmp AL, 0F9h
                je type8_found
                jmp notype_found
type1_found:
                mov DX, OFFSET type1
                call WRITE_MSG
                ret
type2_found:
                mov DX, OFFSET type2
                call WRITE_MSG
                ret
type3_found:
                mov DX, OFFSET type3
                call WRITE_MSG
                ret
type4_found:
                mov DX, OFFSET type4
                call WRITE_MSG
                ret
type5_found:
                mov DX, OFFSET type5
                call WRITE_MSG
                ret
type6_found:
                mov DX, OFFSET type6
                call WRITE_MSG
                ret
type7_found:
                mov DX, OFFSET type7
                call WRITE_MSG
                ret
type8_found:
                mov DX, OFFSET type8
                call WRITE_MSG
                ret
notype_found:
                mov DI, offset notype
                call BYTE_TO_HEX
                mov [DI + 13], AX
                mov DX, offset notype
                call WRITE_MSG
                ret
PRINT_PC_TYPE ENDP
;---------------------------------------
LOAD_INFO PROC near
                mov AH,30h
                int 21h
                ret
LOAD_INFO ENDP
;---------------------------------------
PRINT_MS_DOS_VER PROC near
                ;Вывод версий DOS
                mov SI, OFFSET dos_ver_string
                add SI, 16
                call BYTE_TO_DEC
```

```
                add SI, 3
                mov AL, AH
                call BYTE_TO_DEC
                mov DX, OFFSET dos_ver_string
                call WRITE_MSG
                ;Вывод OEM
                mov SI, OFFSET oem_string
                add SI, 13
                mov AL, BH
                call BYTE_TO_DEC
                mov DX, OFFSET oem_string
                call WRITE_MSG
                ;Вывод серийного номера пользователя
                mov DI, OFFSET user_number_string
                add DI, 22
                mov AX, CX
                call WRD_TO_HEX
                mov AL, BL
                call BYTE_TO_HEX
                mov [DI + 4], AX
                mov DX, OFFSET user_number_string
                call WRITE_MSG
                ret
PRINT_MS_DOS_VER ENDP
;--------------------------------------
; КОД
BEGIN:
                call GET_PC_TYPE
                call PRINT_PC_TYPE
                call LOAD_INFO
                call PRINT_MS_DOS_VER
;Выход в DOS
                xor AL,AL
                mov AH,4Ch
                int 21H
TESTPC ENDS
 END START ; Конец модуля, start - точка входа
```

Файл lr1exe.asm

```
ASTACK     SEGMENT         STACK
          DW 64 DUP(?)
ASTACK     ENDS

DATA SEGMENT
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, '$'
type4 db 'IBM PC Type: PS2 mod. 30', 0DH, 0AH, '$'
type5 db 'IBM PC Type: PS2 mod. 30/60', 0DH, 0AH, '$'
type6 db 'IBM PC Type: PS2 mod. 60', 0DH, 0AH, '$'
type7 db 'IBM PC Type: PCjr', 0DH, 0AH, '$'
type8 db 'IBM PC Type: PC Convertible', 0DH, 0AH, '$'
notype db 'Unknown type:  ', 0DH, 0AH, '$'
dos_ver_string db 'MS-DOS Version:  .  ', 0DH, 0AH, '$'
oem_string db 'OEM Number:   ', 0DH, 0AH, '$'
user_number_string db 'User serial number:        ', 0DH, 0AH, '$'
```

```
DATA ENDS
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:ASTACK
; Процедуры
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
;--------------------------------
WRITE_MSG PROC near
                mov AH, 09h
                int 21h
                ret
WRITE_MSG 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, в AH - младшая
                pop CX
                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 10чную с/с, SI - адрес 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
;-------------------------------
GET_PC_TYPE PROC near
;Вывод типа IBM PC
                push BX
                mov BX,0F000h
                mov ES,BX
                mov AL,ES:[0FFFEh]
                pop BX
                ret
GET_PC_TYPE ENDP
;--------------------------------
PRINT_PC_TYPE PROC near
                cmp AL, 0FFh
                je type1_found
                cmp AL, 0FEh
                je type2_found
                cmp AL, 0FCh
                je type3_found
                cmp AL, 0FAh
                je type4_found
                cmp AL, 0FCh
                je type5_found
                cmp AL, 0F8h
                je type6_found
                cmp AL, 0FDh
                je type7_found
                cmp AL, 0F9h
                je type8_found
                jmp notype_found
type1_found:
                mov DX, OFFSET type1
                call WRITE_MSG
                ret
type2_found:
                mov DX, OFFSET type2
                call WRITE_MSG
                ret
type3_found:
                mov DX, OFFSET type3
                call WRITE_MSG
                ret
```

```
type4_found:
                mov DX, OFFSET type4
                call WRITE_MSG
                ret
type5_found:
                mov DX, OFFSET type5
                call WRITE_MSG
                ret
type6_found:
                mov DX, OFFSET type6
                call WRITE_MSG
                ret
type7_found:
                mov DX, OFFSET type7
                call WRITE_MSG
                ret
type8_found:
                mov DX, OFFSET type8
                call WRITE_MSG
                ret
notype_found:
                mov DI, offset notype
                call BYTE_TO_HEX
                mov [DI + 13], AX
                mov DX, offset notype
                call WRITE_MSG
                ret
PRINT_PC_TYPE ENDP
;---------------------------------------
LOAD_INFO PROC near
                mov AH,30h
                int 21h
                ret
LOAD_INFO ENDP
;---------------------------------------
PRINT_DOS_VER PROC near
                ;Вывод версий DOS
                mov SI, OFFSET dos_ver_string
                add SI, 16
                call BYTE_TO_DEC
                add SI, 3
                mov AL, AH
                call BYTE_TO_DEC
                mov DX, OFFSET dos_ver_string
                call WRITE_MSG
                ;Вывод OEM
                mov SI, OFFSET oem_string
                add SI, 13
                mov AL, BH
                call BYTE_TO_DEC
                mov DX, OFFSET oem_string
                call WRITE_MSG
                ;Вывод серийного номера пользователя
                mov DI, OFFSET user_number_string
                add DI, 22
                mov AX, CX
                call WRD_TO_HEX
```

```
                mov AL, BL
                call BYTE_TO_HEX
                mov [DI + 4], AX
                mov DX, OFFSET user_number_string
                call WRITE_MSG
                ret
PRINT_DOS_VER ENDP
;-----------------------------------------
MAIN       PROC FAR
                sub   AX,AX
                push  AX
                mov   AX,DATA
                mov   DS,AX
                call GET_PC_TYPE
                call PRINT_PC_TYPE
                call LOAD_INFO
                call PRINT_DOS_VER
                xor AL,AL
                mov AH,4Ch
                int 21H
                ret
MAIN  ENDP
CODE ENDS
     END MAIN
```