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. 8383 _____ Максимова А.А.

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.

Выполнение работы.

Был написан текст исходного .COM модуля, код которого представлен в приложении А, который определяет тип PC и версию системы. Ассемблерная программа читает содержимое предпоследнего байта ROM BIOS. Полученный код сравнивается с известными значениями, выводится тип PC. В случае, когда код не совпадает ни с одним из известных значений, переводится в символьную строку, содержащую запись шестнадцатеричного числа и выводится на экран. После определяется версия системы: используя функцию 30H прерывания 21H, получаем выходные параметры. Формируются соответствующие текстовые строки, выводятся на экран. Строятся "плохой" и "хороший".EXE, код которого представлен в приложении Б. Результаты работы модулей представлены на рис. 1, 2 и 3 соответственно.

Процедуры:

Название:	**Предназначение:**
TETR_TO_HEX	Процедура перевода тетрады в шестнадцатеричную цифру (символ)
BYTE_TO_HEX	Процедура перевода байта AL в два символа шестнадцатеричного числа
WRD_TO_HEX	Процедура перевода слова в шестнадцатеричную систему счисления
BYTE_TO_DEC	Процедура перевода байта в десятичную систему счисления

PRINTF	Процедура печати
TYPE_PC	Процедура определения и вывода типа PC
CORRECT	Процедура для вывода версии системы в заданном формате xx.yy
VERSION_SYSTEM	Процедура определения и вывода версии системы, серийного номера OEM и номера пользователя

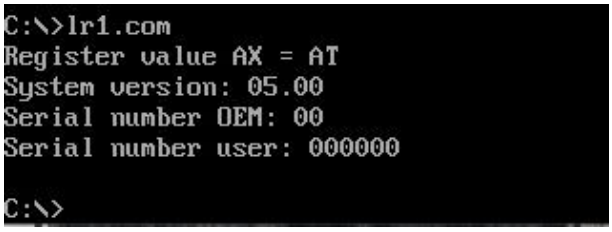

Рисунок 1 - результат работы .com

Рисунок 2 - результат работы "плохого" .exe

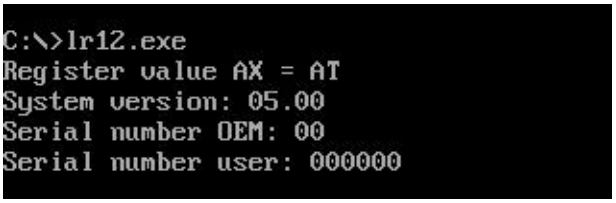

Рисунок 3 - результат работы "хорошего" .exe

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

COM программа должна содержать один сегмент - данные, код и стек объединены в одном сегменте, поэтому COM-файл ограничен размером одного сегмента и не превышает 64К.

2. Сколько сегментов должна содержать EXE-программа?

EXE-программы могут иметь любое число сегментов, минимум 1.

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

Директива ORG 100h, устанавливающая значение программного счетчика в 100h (при загрузке COM файла в память DOS занимает первые 256 байт блоком данных PSP и располагает код программы только после этого блока). Директива assume, показывающая соответствие между введенными сегментами программы и сегментными регистрами, обязательна должна быть в тексте Com-программы, состоящей из одного сегмента для привязки сегментных регистров кода (CS) и данных (DS) соответствующему сегменту (TESTPC). Без использования данной директивы программа не будет скомпилированна и будет выведена ошибка "Отсутствует или недоступен CS".

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

Невозможно использовать команды, содержащие seg <имя сегмента>, так как адрес сегмента до запуска программы неизвестен. Exe-программа берет адреса из таблицы настроек, в Com-программах таблица настроек отсутствует.

Был запущен FAR, в котором были открыты загрузочные модули файлов .COM и .EXE ("хорошего" и "плохого"). Соответствующие скриншоты представлены на рис. 4, 5 и 6.

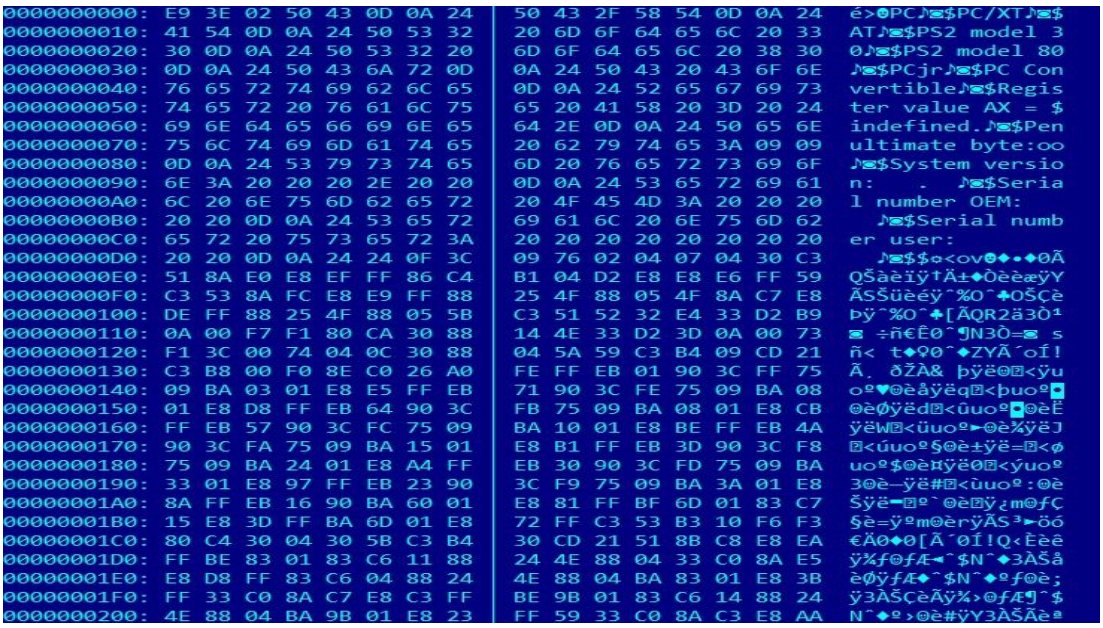

Рисунок 4 - загрузочный модуль .COM

Рисунок 5 - загрузочный модуль плохого .EXE

Рисунок 6 - загрузочный модуль хорошего .EXE

Отличия форматов файлов COM и EXE модулей

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

   Модуль com-файла содержит данные и код, которые располагаются в одном сегменте, начиная с нулевого адреса. При загрузке происходит смещение в 256 байт (org 100h), генерируется стек, который занимает все оставшуюся память.

2. Какова структура файла "плохого" EXE? С какого адреса располагается код?

   Модуль плохого exe-файла также содержит данные и код, которые располагаются в одном сегменте, но начиная с трехсотого (в 16 системе счисления) адреса. С 0 адреса до 300h располагаются заголовок - информация для загрузчика, расположенная в начале файла, и таблица настройки адресов, 100h отводится под PSP.

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

Модуль хорошего exe-файла содержит стэк, данные и код, расположенные в отдельных сегментах, в отличии от "плохого" EXE. Различается также расположение кода с 200h , так как в "хорошем" EXE нет смещения на 100h.

Загрузка COM модуля в основную память

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

   Во время загрузки COM - программы система выделяет первый свободный сегмент памяти и в его начале (первые 256 байт) размещается PSP. За PSP располагается код программы, то есть с адреса 100h. В IP - счетчик команд, устанавливается значение 100h. В стек записывается адрес 0000h, для возврата по ret.

2. Что располагается с адреса 0?

С адреса 0 располагается PSP, занимающий 256 байт.

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

   Все сегментные регистры содержат адрес PSP, то есть значение 48DD.

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

   Указатель стека устанавливается на конец сегмента программы (FFFE). Стек, располагающийся с старших адресов, занимает всю доступную память, не зарезервированную PSP и кодом.

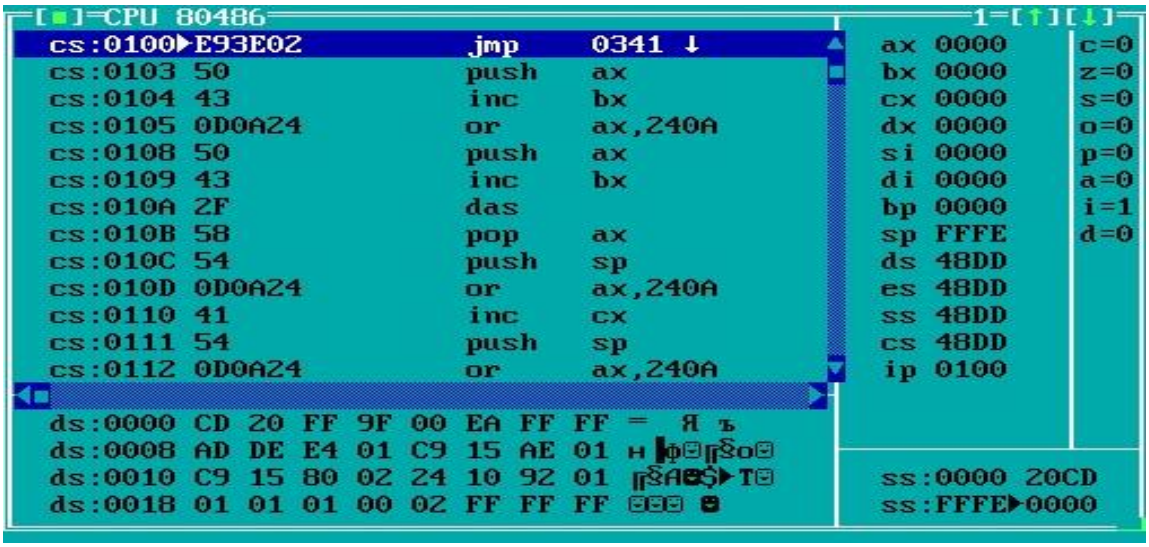

Рисунок 7 - Результат загрузки COM файла в память

Загрузка "хорошего" EXE модуля в основную память

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

Определяется сегментный адрес свободного участка памяти, размер которого достаточен для размещения программы. DS и ES указывают на начало PSP, CS указывает на начало сегмента кода, SS на начало сегмента стека (значения, указанные в заголовке). Управление передается по адресу, указанному в заголовке.

2. На что указывают регистры DS и ES?

Регистры DS и ES указывают на PSP.

3. Как определяется стек?

Стек определяется с помощью упрощенной директивы .stack (или стандартной имя SEGMENT STACK ... имя ENDS), задается размер стека (в соответствии с моделью памяти). SS - указывает на начало сегмента стека (значения, указанные в заголовке), SP - на его смещение.

4. Как определяется точка входа?

Точкой входа в программу может быть метка, объявленная в директиве END - директива, которой завершается программа, так как метка не является обязательным параметром этой директивы, то точкой входа, в таком случае, будет начало сегмента кода.

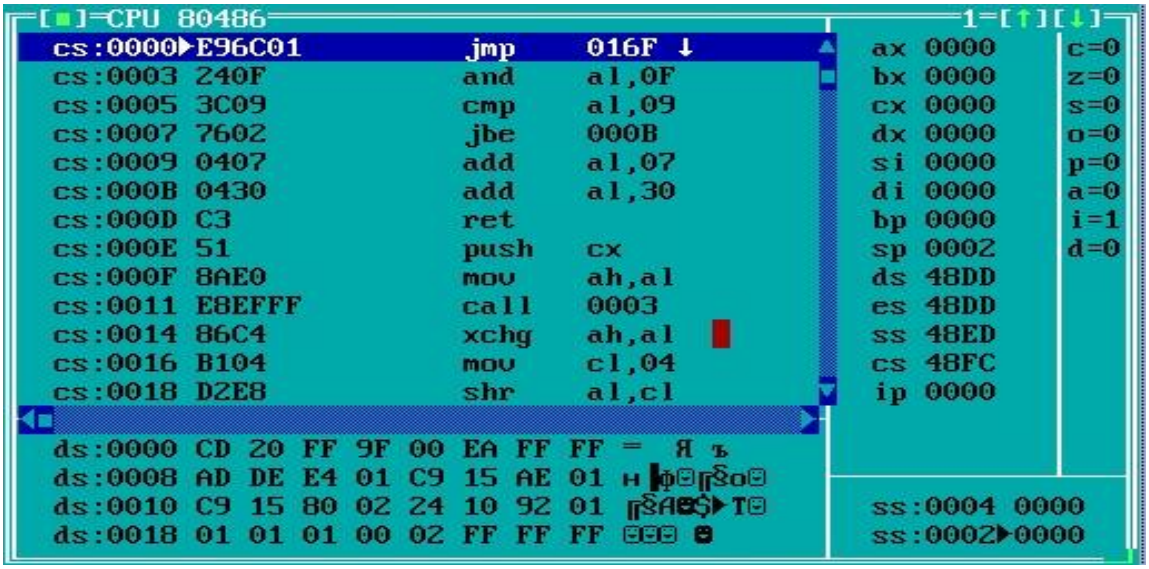

Рисунок 8 - Результат загрузки EXE файла в память

Выводы.

В результате выполнения лабораторной работы были исследованы различия в структурах исходных текстов и модулей COM и EXE файлов и способах их загрузки в основную память.

ПРИЛОЖЕНИЕ А

```
COMMENT *
МАКСИМОВА АНАСТАСИЯ, ГРУППА 8383
*

TESTPC      SEGMENT
            ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
            ORG 100H

START:      JMP         BEGIN

;ДАННЫЕ
TYPE1                   DB          'PC', 0DH, 0AH, '$'
TYPE2                   DB          'PC/XT', 0DH, 0AH, '$'
TYPE3                   DB          'AT', 0DH, 0AH, '$'    ;TYPE5 = TYPE3
TYPE4                   DB          'PS2 MODEL 30', 0DH, 0AH, '$'
TYPE6                   DB          'PS2 MODEL 80', 0DH, 0AH, '$'
TYPE7                   DB          'PCJR', 0DH, 0AH, '$'
TYPE8                   DB          'PC CONVERTIBLE', 0DH, 0AH, '$'
STRING                        DB          'REGISTER VALUE AX = ', '$'
VAR0_STR1               DB          'INDEFINED.', 0DH, 0AH, '$'
VAR0_STR2               DB          'PENULTIMATE BYTE:           ',  0DH,  0AH,
'$'
SYSTEM_VERSION          DB          'SYSTEM VERSION:    .  ', 0DH, 0AH, '$'
SERIAL_NUMB_OEM  DB          'SERIAL NUMBER OEM:      ', 0DH, 0AH, '$'
SERIAL_NUMB_USER DB          'SERIAL NUMBER USER:           ', 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
;-------------------------------------------------------------
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
                        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 10 С/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
;-------------------------------------------------------------
PRINTF            PROC  NEAR
            MOV         AH, 09H
          INT     21H
            RET
PRINTF            ENDP
;-------------------------------------------------------------
TYPE_PC           PROC  NEAR
                        MOV         AX, 0F000H
                        MOV         ES, AX
                        MOV   AL,   ES:[0FFFEH]
                        JMP   SEARCH_TYPE
SEARCH_TYPE:
                        CMP   AL, 0FFH
                        JNE         VAR2_1
                        MOV         DX, OFFSET TYPE1
                        CALL  PRINTF
                        JMP         EXIT

VAR2_1:                 CMP   AL, 0FEH
                        JNE         VAR2_2
                        MOV         DX, OFFSET TYPE2
                        CALL  PRINTF
                        JMP         EXIT

VAR2_2:                 CMP   AL, 0FBH
                        JNE         VAR3
                        MOV         DX, OFFSET TYPE2
                        CALL  PRINTF
                        JMP         EXIT

VAR3:             CMP   AL, 0FCH
                        JNE VAR4
                        MOV         DX, OFFSET TYPE3
```

```
                    CALL  PRINTF
                    JMP         EXIT

VAR4:         CMP   AL, 0FAH
                    JNE         VAR6
                    MOV         DX, OFFSET TYPE4
                    CALL  PRINTF
                    JMP         EXIT

VAR6:         CMP   AL, 0F8H
                    JNE         VAR7
                    MOV         DX, OFFSET TYPE6
                    CALL  PRINTF
                    JMP         EXIT

VAR7:         CMP   AL, 0FDH
                    JNE         VAR8
                    MOV         DX, OFFSET TYPE7
                    CALL  PRINTF
                    JMP         EXIT

VAR8:         CMP   AL, 0F9H
                    JNE         VAR0
                    MOV         DX, OFFSET TYPE8
                    CALL  PRINTF
                    JMP         EXIT

VAR0:

                    MOV         DX, OFFSET VAR0_STR1
                    CALL  PRINTF
                    MOV   DI,   OFFSET VAR0_STR2
                    ADD         DI, 21
                    CALL  WRD_TO_HEX
                    MOV         DX, OFFSET VAR0_STR2
                    CALL  PRINTF

EXIT:
                    RET
TYPE_PC       ENDP
;-------------------------------------------------------------
CORRECT             PROC  NEAR
```

```
                        PUSH  BX
                        MOV   BL, 16      ;BL - ДЕЛИТЕЛЬ
                        DIV         BL          ;AH - ОСТАТОК AL - ЦЕЛОЕ
                        ADD   AH, '0'
                        ADD   AL, '0'
                        POP   BX
                        RET
CORRECT                 ENDP
;-------------------------------------------------------------
VERSION_SYSTEM    PROC  NEAR
                        MOV         AH, 30H
                        INT         21H

                        PUSH CX
                        MOV   CX, AX            ;SAVE

                        ;AL
                        CALL  CORRECT
                        MOV   SI,   OFFSET SYSTEM_VERSION
                        ADD         SI, 17
                        MOV         [SI], AH

                        DEC         SI
                        MOV         [SI], AL

                        ;AH
                        XOR   AX, AX
                        MOV   AH, CH
                        CALL  CORRECT
                        ADD         SI, 4
                        MOV         [SI], AH

                        DEC         SI
                        MOV         [SI], AL

                        MOV         DX, OFFSET SYSTEM_VERSION
                        CALL  PRINTF

                        ;SERIAL_NUMB_OEM
                        XOR   AX, AX
                        MOV   AL, BH
```

```
CALL  CORRECT
MOV   SI,   OFFSET SERIAL_NUMB_OEM
ADD         SI, 20
MOV         [SI], AH

DEC         SI
MOV         [SI], AL

MOV         DX, OFFSET SERIAL_NUMB_OEM
CALL  PRINTF

;SERIAL_NUMB_USER
;BL
POP   CX
XOR   AX, AX
MOV   AL, BL

CALL  CORRECT
MOV   SI,   OFFSET SERIAL_NUMB_USER
ADD         SI, 21
MOV         [SI], AH

DEC         SI
MOV         [SI], AL

;CH
XOR   AX, AX
MOV   AL, CH

CALL  CORRECT
ADD         SI, 3
MOV         [SI], AH

DEC         SI
MOV         [SI], AL

;CL
XOR   AX, AX
MOV   AL, CL

CALL  CORRECT
ADD         SI, 3
```

```
                        MOV         [SI], AH

                        DEC         SI
                        MOV         [SI], AL

                        MOV         DX, OFFSET SERIAL_NUMB_USER
                        CALL  PRINTF


                        RET
VERSION_SYSTEM   ENDP
;--------------------------------------------------------
BEGIN:
;ВЫВОД СТРОКИ ТЕКСТА ИЗ ПОЛЯ STRING
                        MOV         DX, OFFSET STRING
                        MOV         AH, 09H
                        INT   21H

;ВЫПОЛНЕНИЕ ЗАДАНИЙ
                        CALL TYPE_PC
                        CALL VERSION_SYSTEM

;ВЫХОД В DOS
                        XOR         AL, AL
                        MOV         AH, 4CH
                        INT         21H
TESTPC      ENDS
            END   START             ;КОНЕЦ МОДУЛЯ, START - ТОЧКА ВХОДА
```

ПРИЛОЖЕНИЕ Б

```
      COMMENT *
      МАКСИМОВА АНАСТАСИЯ, ГРУППА 8383
      *

      ASTACK  SEGMENT  STACK
              DW 100H  DUP(?)                ;ОТВОДИТСЯ 100 СЛОВ ПАМЯТИ
      ASTACK  ENDS

      DATA SEGMENT
      ;ДАННЫЕ
      TYPE1                   DB          'PC', 0DH, 0AH, '$'
      TYPE2                   DB          'PC/XT', 0DH, 0AH, '$'
      TYPE3                   DB          'AT', 0DH, 0AH, '$'              ;TYPE5
= TYPE3
      TYPE4                   DB          'PS2 MODEL 30', 0DH, 0AH, '$'
      TYPE6                   DB          'PS2 MODEL 80', 0DH, 0AH, '$'
      TYPE7                   DB          'PCJR', 0DH, 0AH, '$'
      TYPE8                   DB          'PC CONVERTIBLE', 0DH, 0AH, '$'
      STRING                        DB          'REGISTER VALUE AX = ', '$'
      VAR0_STR1               DB          'INDEFINED.', 0DH, 0AH, '$'
      VAR0_STR2               DB          'PENULTIMATE BYTE:          ',  0DH,  0AH,
'$'
      SYSTEM_VERSION          DB          'SYSTEM VERSION:   .  ', 0DH, 0AH, '$'
      SERIAL_NUMB_OEM  DB          'SERIAL NUMBER OEM:     ', 0DH, 0AH, '$'
      SERIAL_NUMB_USER DB          'SERIAL NUMBER USER:          ', 0DH, 0AH, '$'
      DATA    ENDS

      CODE  SEGMENT
                ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:ASTACK

      ;START:     JMP         BEGIN


      ;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
;--------------------------------------------------------------
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
                      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 10 С/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
;-------------------------------------------------------------
PRINTF          PROC  NEAR
          MOV       AH, 09H
         INT    21H
          RET
PRINTF          ENDP
;-------------------------------------------------------------
TYPE_PC         PROC  NEAR
                    MOV         AX, 0F000H
                    MOV         ES, AX
                    MOV   AL,   ES:[0FFFEH]
                    JMP   SEARCH_TYPE
SEARCH_TYPE:
                    CMP   AL, 0FFH
                    JNE         VAR2_1
                    MOV         DX, OFFSET TYPE1
                    CALL  PRINTF
                    JMP         EXIT

VAR2_1:             CMP   AL, 0FEH
                    JNE         VAR2_2
                    MOV         DX, OFFSET TYPE2
                    CALL  PRINTF
                    JMP         EXIT
```

```
VAR2_2:             CMP   AL, 0FBH
                    JNE         VAR3
                    MOV         DX, OFFSET TYPE2
                    CALL  PRINTF
                    JMP         EXIT

VAR3:         CMP   AL, 0FCH
                    JNE VAR4
                    MOV         DX, OFFSET TYPE3
                    CALL  PRINTF
                    JMP         EXIT

VAR4:         CMP   AL, 0FAH
                    JNE         VAR6
                    MOV         DX, OFFSET TYPE4
                    CALL  PRINTF
                    JMP         EXIT

VAR6:         CMP   AL, 0F8H
                    JNE         VAR7
                    MOV         DX, OFFSET TYPE6
                    CALL  PRINTF
                    JMP         EXIT

VAR7:         CMP   AL, 0FDH
                    JNE         VAR8
                    MOV         DX, OFFSET TYPE7
                    CALL  PRINTF
                    JMP         EXIT

VAR8:         CMP   AL, 0F9H
                    JNE         VAR0
                    MOV         DX, OFFSET TYPE8
                    CALL  PRINTF
                    JMP         EXIT

VAR0:

                    MOV         DX, OFFSET VAR0_STR1
                    CALL  PRINTF
                    MOV   DI,   OFFSET VAR0_STR2
                    ADD         DI, 21
```

```
                        CALL  WRD_TO_HEX
                        MOV         DX, OFFSET VAR0_STR2
                        CALL  PRINTF

EXIT:
                        RET
TYPE_PC          ENDP
;-------------------------------------------------------------
CORRECT                 PROC  NEAR
                        PUSH  BX
                        MOV   BL, 16      ;BL - ДЕЛИТЕЛЬ
                        DIV         BL          ;AH - ОСТАТОК AL - ЦЕЛОЕ
                        ADD   AH, '0'
                        ADD   AL, '0'
                        POP   BX
                        RET
CORRECT                 ENDP
;-------------------------------------------------------------
VERSION_SYSTEM   PROC  NEAR
                        MOV         AH, 30H
                        INT         21H

                        PUSH CX
                        MOV   CX, AX            ;SAVE

                        ;AL
                        CALL  CORRECT
                        MOV   SI,   OFFSET SYSTEM_VERSION
                        ADD         SI, 17
                        MOV         [SI], AH

                        DEC         SI
                        MOV         [SI], AL

                        ;AH
                        XOR   AX, AX
                        MOV   AH, CH
                        CALL  CORRECT
                        ADD         SI, 4
                        MOV         [SI], AH

                        DEC         SI
```

```
MOV         [SI], AL

MOV         DX, OFFSET SYSTEM_VERSION
CALL  PRINTF

;SERIAL_NUMB_OEM
XOR   AX, AX
MOV   AL, BH

CALL  CORRECT
MOV   SI,   OFFSET SERIAL_NUMB_OEM
ADD         SI, 20
MOV         [SI], AH

DEC         SI
MOV         [SI], AL

MOV         DX, OFFSET SERIAL_NUMB_OEM
CALL  PRINTF

;SERIAL_NUMB_USER
;BL
POP   CX
XOR   AX, AX
MOV   AL, BL

CALL  CORRECT
MOV   SI,   OFFSET SERIAL_NUMB_USER
ADD         SI, 21
MOV         [SI], AH

DEC         SI
MOV         [SI], AL

;CH
XOR   AX, AX
MOV   AL, CH

CALL  CORRECT
ADD         SI, 3
MOV         [SI], AH
```

```
                        DEC         SI
                        MOV         [SI], AL

                        ;CL
                        XOR   AX, AX
                        MOV   AL, CL

                        CALL  CORRECT
                        ADD         SI, 3
                        MOV         [SI], AH

                        DEC         SI
                        MOV         [SI], AL

                        MOV         DX, OFFSET SERIAL_NUMB_USER
                        CALL  PRINTF


                        RET
VERSION_SYSTEM   ENDP
;-------------------------------------------------------------
BEGIN PROC FAR
                        PUSH  DS
                        PUSH    AX
                        XOR   AX, AX
                        PUSH  AX

                        MOV   AX, DATA
                        MOV   DS, AX
                        POP     AX
;ВЫВОД СТРОКИ ТЕКСТА ИЗ ПОЛЯ STRING
                        MOV         DX, OFFSET STRING
                        MOV         AH, 09H
                        INT   21H

;ВЫПОЛНЕНИЕ ЗАДАНИЙ
                        CALL TYPE_PC
                        CALL VERSION_SYSTEM

;ВЫХОД В DOS
                        XOR         AL, AL
                        MOV         AH, 4CH
```

```
                        INT         21H
BEGIN ENDP
CODE  ENDS
            END   BEGIN             ;КОНЕЦ МОДУЛЯ, START - ТОЧКА ВХОДА
```