МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе №1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей

Студентка гр. 8381 _____ Звегинцева Е.Н.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Необходимые сведения для составления программы.

Тип IBM PC хранится в байте по адресу 0F000:0FFFE, в предпоследнем байте ROM BIOS. Соответствие кода и типа в таблице:

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**

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH:

Выходными параметрами являются:

AL – номер основной версии. Если 0, то <2.0;

AH – номер модификации;

BH – серийный номер OEM (Original Equipment Manufacturer);

BL:CX – 24-битовый серийный номер пользователя.

Постановка задачи.

Требуется реализовать текст исходного .COM модуля, который определяет тип PC и версию системы. 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 (Original Equipment Manufacturer) и серийным номером пользователя. Полученные строки выводятся на экран.

Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить и сравнить исходные тексты для .COM и .EXE модулей.

Выполнение работы.

Выполнение работы производилось на базе операционной системы Windows 10. Сборка и отладка модулей производилась с помощью компиляторов MASM и TASM и отладчика TD в эмуляторе DOSBox.

Типы исполняемых файлов в DOS исследовались на примере программы, которая выводит версию операционной системы. Программа содержит таблицу кодов, обозначающих различные типы PC. Она считывает код из памяти BIOS, ищет его по таблице и выводит строку с названием этого типа (функции LOOKUP_MODEL и PRINT_MODEL). Затем загружается информация о версии DOS, и отображается в функции

PRINT_VERSION_INFO. Вывод чисел выполняется при помощи функций печати чисел в строку BYTE_TO_DEC, BYTE_TO_HEX и WRD_TO_HEX из методички. Эта программа была написана таким образом, чтобы из неё можно было получить загрузочный модуль типа COM (LAB1COM.ASM). После этого она была модифицирована в программу EXE (LAB1EXE.ASM).

Пример сборки .COM модуля показан на рис. 1.

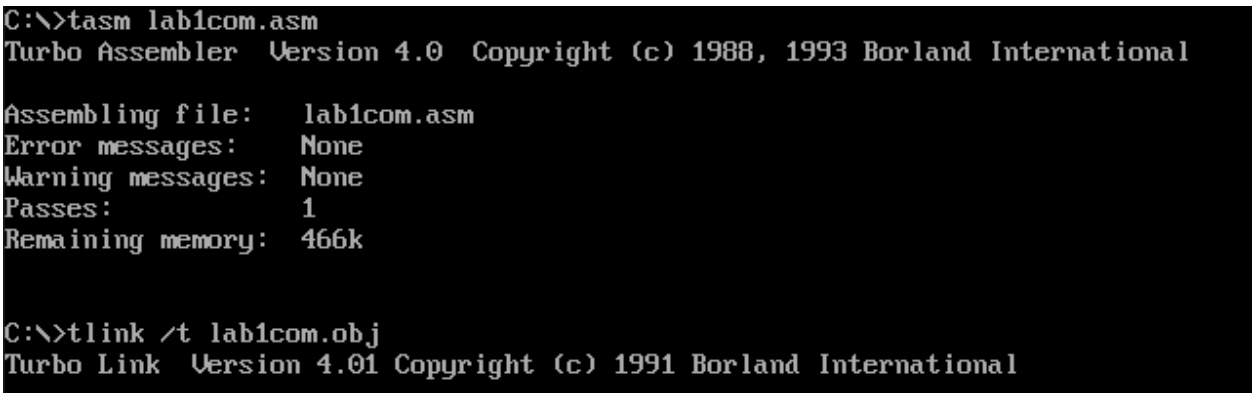

Рисунок 1 – Полученный .COM модуль

Во время линковки «плохого» .EXE модуля было выведено предупреждение об отсутствии сегмента стека, представлено на рис. 2.

Запуск «плохого» .EXE модуля.

Рисунок 2 – Предупреждение во время линковки «плохого» .EXE

Рисунок 3 – Вывод плохого .EXE модуля

Рсиунок 4 — Вывод .COM модуля

Рисунок 5 — Вывод «хорошего» .EXE модуля

Отличия исходных текстов COM и EXE программ.

1. **Сколько сегментов должна содержать COM-программа?**

Один: сегмент кода и данных. Сегмент стека создаётся при запуске автоматически.

2. **EXE программа?**

Произвольное число сегментов (в частности, стек нужно создавать вручную, т.к. он не создаётся автоматически).

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

Директива ORG 100h (смещение 100h), так как при загрузке COM-файла в память DOS занимает первые 256 байт (100h) блоком данных PSP и

располагает код программы только после этого блока. Также обязательна директива ASSUME, которая устанавливает значения регистров CS, DS на адрес сегмента программы.

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

Нельзя использовать команды, связанные с адресом сегмента, так как он неизвестен до загрузки этого сегмента в память, т.к. в .COM файле отсутствует таблица настройки с информацией о типе адресов и их местоположении.

Отличия форматов файлов COM и EXE модулей.

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

Вид файла COM в шестнадцатеричном формате представлен на рис. 6.

Рисунок 6 – Вид COM файла в шестнадцатеричном виде

COM файл состоит из одного сегмента и содержит данные и машинные команды. размер файла не превышает 64 КБ.

Файл начинается с E9 C1 01 – jmp к адресу 0x103 + 0x1c1 (0x103 – значение IP при выполнении этой команды), с которого начинается функция

MAIN. После кода программы в файле также ничего нет. По файлу карты памяти, который выдаёт линковщик, видно, что длина файла 1E316 равна длине единственного сегмента за вычетом длины PSP (10016 байт)

```
Start   Stop    Length Name                 Class

 00000H 002E2H 002E3H TEXT
```

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

Вид «плохого» EXE файла в шестнадцатеричном формате представлен на рис. 7.

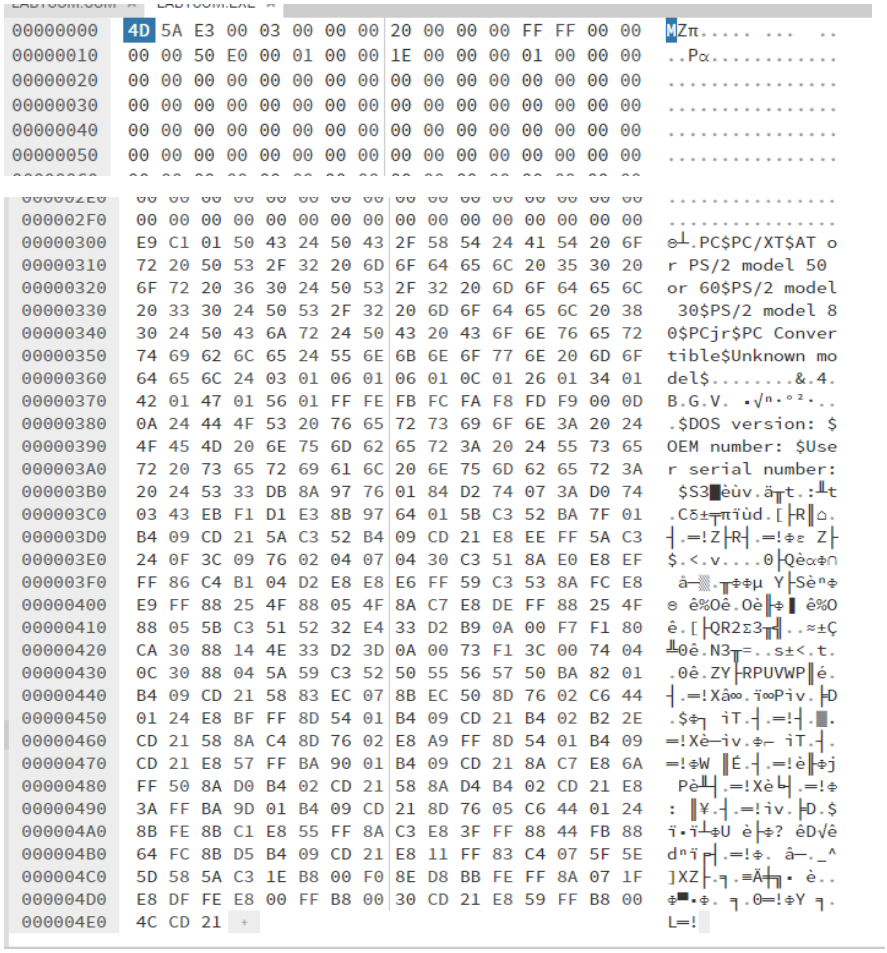

Рисунок 7 – Вид «плохого» EXE файла

В «плохом» EXE файле данные и код содержатся в одном сегменте. Код располагается с адреса 300h. С адреса 0h располагается Relocation Table

(таблица разметки). Также 100h резервируются командой ORG 100h (что вызовет отличие в структуре «плохого» и «хорошего» EXE).

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

Вид «хорошего» EXE в шестнадцатеричном виде представлен на рис.8

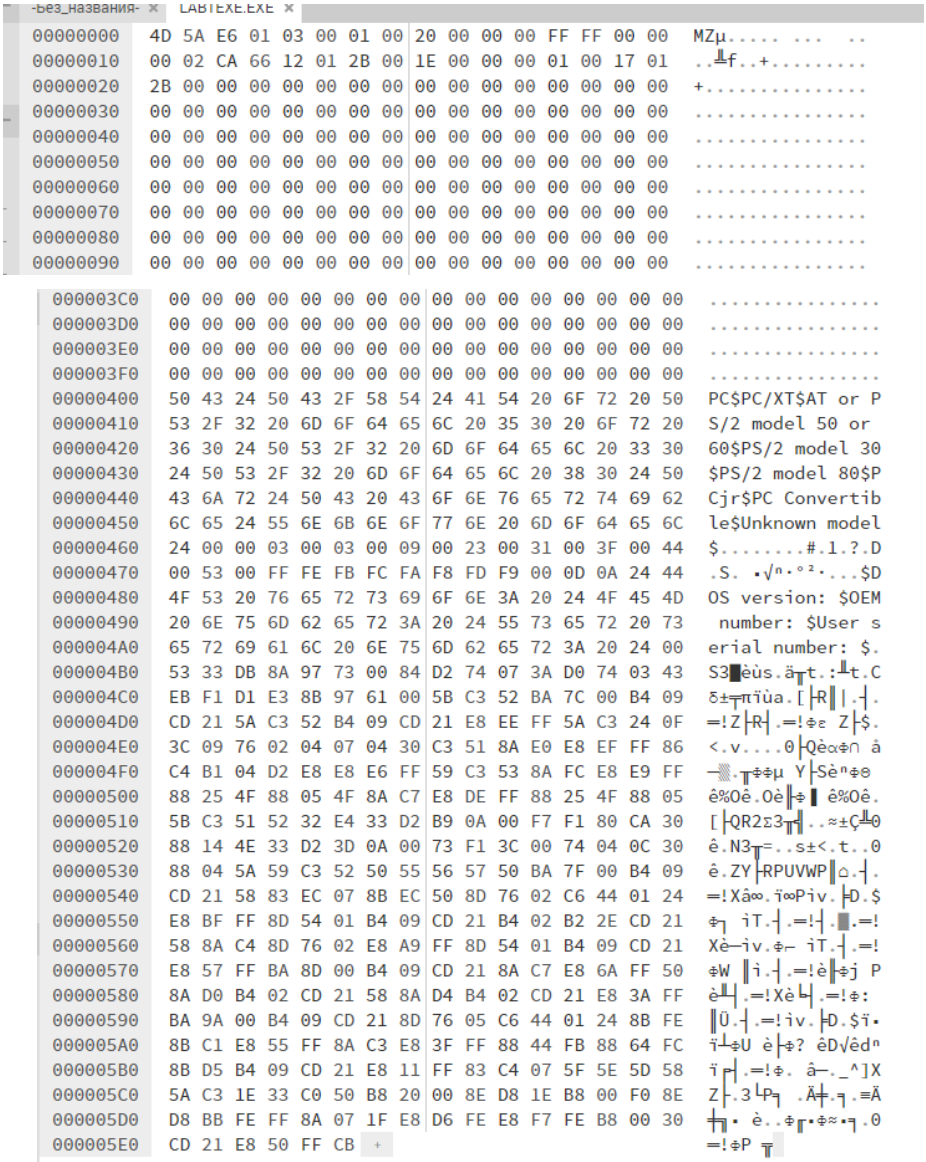

Рисунок 8 – Вид «хорошего» EXE файла

В «хорошем» файле EXE содержится информация для загрузчика, сегмент стека, сегмент данных и сегмент кода (3 сегмента вместо одного в «плохом» EXE). Если из исходного текста EXE-программы убрать сегмент стека, то код будет располагаться с адреса 200h. Отличие от «плохого» EXE в том, что в «хорошем» не резервируется дополнительно 100h, которые в COM

файле требовались для PSP, поэтому адреса начала кода отличаются на 100h + S, где S – размер стека.

Здесь сегмент стека начинается по адресу 0x200 от начала файла, данные – 0x400 и код – 0x4B0; все адреса будут на 20016 меньше, если их брать от начала образа программы. Это видно в файле карты памяти:

```
Start Stop Length Name Class
00000H 001FFH 00200H STACK
00200H 002AEH 000AFH DATA
002B0H 003E5H 00136H TEXT
```

Загрузка COM модуля в основную память.

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

Запуск файла .COM в отладчике TD.EXE представлен на рис 9.

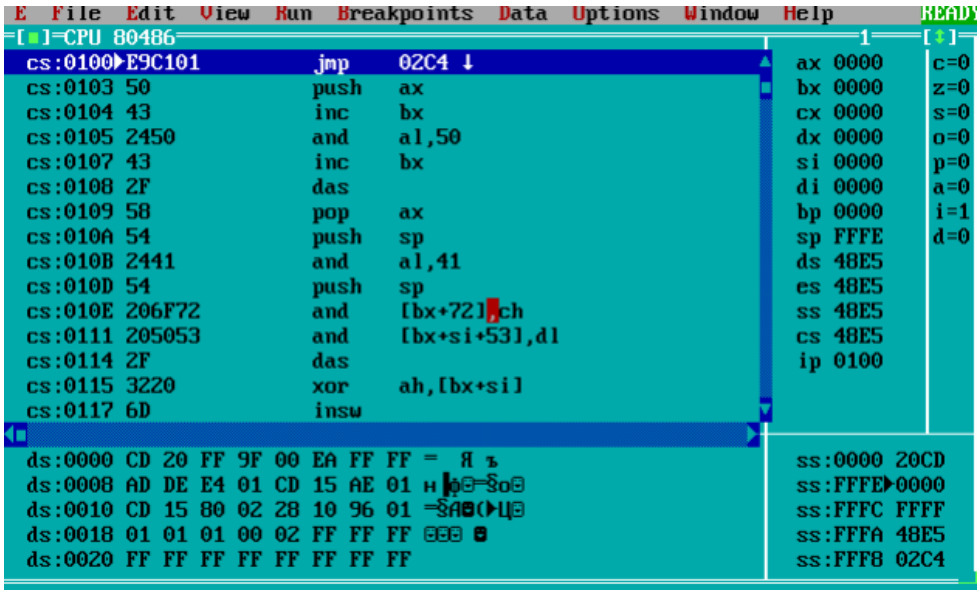

Рисунок 9 – Отладка файла .COM

Всё содержимое файла COM загружается в один и тот же сегмент по сдвигу 0x100; исполнение начинается с этого же адреса

2. **Что располагается с 0 адреса?**

PSP – Program Segment Prefix. Это структура, содержащая, в частности, команду int 20h (завершение программы), объём доступной памяти в

«параграфах», адреса подпрограмм, обрабатывающих некоторые события, 2 параметрические области и буфер передачи данных (DTA), который после запуска программы содержит остаток командной строки.

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

Сегментные регистры имеют значения 48DDh и указывают на PSP.

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

В COM модуле нельзя объявить стек, он создается автоматически. Указатель стека устанавливается на конец сегмента – FFFEh. Стек занимает оставшуюся память, а его адреса изменяются от больших к меньшим, то есть от FFFEh к 0000h.

Загрузка «хорошего» EXE. модуля в основную память.

Запуск «хорошего» EXE модуля в отладчике TD.EXE представлен на рис.10.

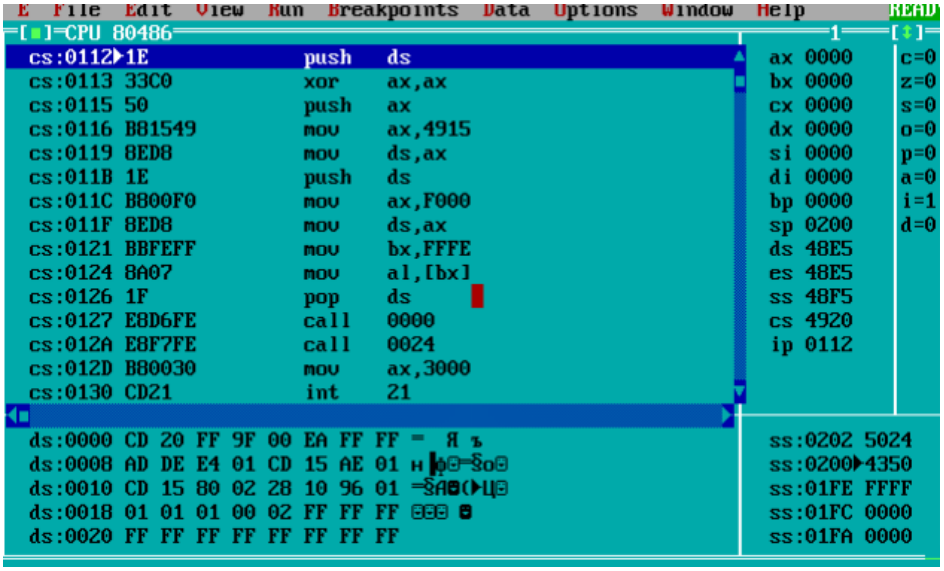

Рисунок 10 – Загрузка «хорошего» EXE модуля в память

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

В начале, модуль загружается в начальный сегмент. Потом, обрабатывается таблица настройки, т.е. изменяются адреса в программе, которые зависят от адреса начального сегмента. 8 Сегментный регистр CS

будет содержать адрес сегмента кода, регистр SS – адрес сегмента стека, а DS и ES – адрес сегмента, в котором по адресу 0 расположен PSP. Значение регистра DS, как правило, перезаписывается программой в начале выполнения адресом её сегмента данных. Перед этим его значение и слово, равное нулю добавляются в стек, чтобы из главной функции программы можно было выйти при помощи инструкции ret far. Это возможно, поскольку по адресу 0 в PSP лежит инструкция int 20h, которая завершает программу. В примере на скриншоте, образ программы загружен сразу за PSP. Начальный сегмент имеет адрес 0x48F5.

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

Они указывают на сегмент, в котором расположен PSP.

3. **Как определяется стек?**

Стек может быть объявлен при помощи директивы ASSUME, которая устанавливает сегментный регистр SS на начало сегмента стека, а также задает значение SP, указанное в заголовке. Также стек может быть объявлен с помощью директивы STACK. Если стек не объявлять, то он будет создан автоматически.

4. **Как определяется точка входа?**

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом (функцией или меткой, с которой необходимо начать программу) директивы END.

Вывод.

В ходе выполнения лабораторной работы были исследованы два формата загрузочных модулей в операционной системе DOS – COM и EXE (MZ). Исследованы различия в исходных текстах модулей, а также в их устройстве, формате и способе загрузки в память.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB1COM.ASM

```
TEXT SEGMENT
                ASSUME CS:TEXT, DS:TEXT
                ORG 100h

START:
                jmp MAIN

MODEL_FF        DB  'PC$'
MODEL_FE_FB     DB  'PC/XT$'
MODEL_FC        DB  'AT or PS/2 model 50 or 60$'
MODEL_FA        DB  'PS/2 model 30$'
MODEL_F8        DB  'PS/2 model 80$'
MODEL_FD        DB  'PCjr$'
MODEL_F9        DB  'PC Convertible$'
MODEL_UNKNOWN   DB  'Unknown model$'

MODEL_TAB       DW  OFFSET MODEL_FF
                    DW   OFFSET MODEL_FE_FB
                    DW   OFFSET MODEL_FE_FB
                    DW   OFFSET MODEL_FC
                    DW   OFFSET MODEL_FA
                    DW   OFFSET MODEL_F8
                    DW   OFFSET MODEL_FD
                    DW   OFFSET MODEL_F9
                    DW   OFFSET MODEL_UNKNOWN

MODEL_CHARS     DB  0FFh, 0FEh, 0FBh, 0FCh
                    DB   0FAh, 0F8h, 0FDh, 0F9h, 0

NEWLINE_STRING  DB  13, 10, '$'

DOS_VERSION_STRING  DB   'DOS version: $'
```

```
  OEM_NUMBER_STRING   DB   'OEM number: $'
  USER_SERIAL_STRING  DB   'User serial number: $'


  LOOKUP_MODEL PROC
                push     BX
                xor BX,  BX

                ;; we don't save DX as we return into it
anyway
  _lookup_model_loop:
                mov DL,  MODEL_CHARS[BX]

                ;; zero => unknown model
                test     DL,  DL
                jz  _lookup_model_break

                ;; model code matches => exit
                cmp DL,  AL
                jz  _lookup_model_break

                inc BX
                jmp _lookup_model_loop

  _lookup_model_break:
                ;; we're addressing 2-byte words
                shl BX,  1
                mov DX,  MODEL_TAB[BX]

                pop BX
                ret
  LOOKUP_MODEL ENDP

  NEWLINE PROC
                push     DX
```

```
                mov DX,  OFFSET NEWLINE_STRING
                mov AH,  09h
                int 21h
                pop DX
                ret
NEWLINE ENDP

PRINT_MODEL PROC
                push     DX

                ;; assume the model name is in DX
                mov AH,  09h
                int 21h

                call     NEWLINE

                pop DX
                ret
PRINT_MODEL 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
                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

PRINT_VERSION_INFO PROC
                push    DX
                push    AX
                push    BP
                push    SI
                push    DI

                push    AX

                mov DX,  OFFSET DOS_VERSION_STRING
                mov AH,  09h
                int 21h

                pop AX

                ;; reserver max buffer we're going to need
                sub SP,  7
                mov BP,  SP

                ;; print first line
                push    AX
```

```
lea SI,  [BP+2]
mov BYTE PTR [SI+1],    '$'
call     BYTE_TO_DEC

lea DX,  [SI+1]
mov AH,  09h
int 21h

mov AH,  02h
mov DL,  '.'
int 21h

pop AX

mov AL,  AH
lea SI,  [BP+2]
call     BYTE_TO_DEC

lea DX,  [SI+1]
mov AH,  09h
int 21h

call     NEWLINE

;; print second line
mov DX,  OFFSET OEM_NUMBER_STRING
mov AH,  09h
int 21h

mov AL,  BH
call     BYTE_TO_HEX

push     AX
mov DL,  AL
mov AH,  02h
```

```
int 21h

pop AX
mov DL,  AH
mov AH,  02h
int 21h

call     NEWLINE

;; print third line
mov DX,  OFFSET USER_SERIAL_STRING
mov AH,  09h
int 21h

lea SI,  [BP+5]
mov BYTE PTR [SI+1],    '$'
mov DI,  SI
mov AX,  CX
call     WRD_TO_HEX

mov AL,  BL
call     BYTE_TO_HEX
mov [SI-5],   AL
mov [SI-4],   AH

mov DX,  BP
mov AH,  09h
int 21h

call     NEWLINE

add SP,  7

pop DI
pop SI
```

```
                pop BP
                pop AX
                pop DX
                ret
  PRINT_VERSION_INFO ENDP

  MAIN PROC
                ;; AX <- 0F000:0FFFEh
                push    DS
                mov AX, 0F000h
                mov DS, AX
                mov BX, 0FFFEh
                mov AL, BYTE PTR [BX]
                pop DS

                ;; DX <- model name string offset (rel. to
DS)
                call    LOOKUP_MODEL

                call    PRINT_MODEL

                ;; AL == 00h => return OEM number
                ;; AL <- DOS version major
                ;; AH <- DOS version minor
                ;; BL:CX <- `user serial number'
                ;; BH <- MS-DOS OEM number
                mov AX, 3000h
                int 21h

                call    PRINT_VERSION_INFO

                mov AX, 4C00h
                int 21h
  MAIN ENDP
```

```
TEXT ENDS

        END START
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB1EXE.ASM

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

              ASSUME CS:TEXT, DS:DATA, SS:STACK

DATA SEGMENT

MODEL_FF        DB  'PC$'
MODEL_FE_FB     DB  'PC/XT$'
MODEL_FC        DB  'AT or PS/2 model 50 or 60$'
MODEL_FA        DB  'PS/2 model 30$'
MODEL_F8        DB  'PS/2 model 80$'
MODEL_FD        DB  'PCjr$'
MODEL_F9        DB  'PC Convertible$'
MODEL_UNKNOWN   DB  'Unknown model$'

MODEL_TAB       DW  OFFSET MODEL_FF
                    DW   OFFSET MODEL_FE_FB
                    DW   OFFSET MODEL_FE_FB
                    DW   OFFSET MODEL_FC
                    DW   OFFSET MODEL_FA
                    DW   OFFSET MODEL_F8
                    DW   OFFSET MODEL_FD
                    DW   OFFSET MODEL_F9
                    DW   OFFSET MODEL_UNKNOWN

MODEL_CHARS     DB  0FFh, 0FEh, 0FBh, 0FCh
                    DB   0FAh, 0F8h, 0FDh, 0F9h, 0

NEWLINE_STRING  DB  13, 10, '$'
```

```
  DOS_VERSION_STRING  DB   'DOS version: $'
  OEM_NUMBER_STRING   DB   'OEM number: $'
  USER_SERIAL_STRING  DB   'User serial number: $'

  DATA ENDS

  TEXT SEGMENT

  LOOKUP_MODEL PROC
                push     BX
                xor BX,  BX

                ;; we don't save DX as we return into it
anyway
  _lookup_model_loop:
                mov DL,  MODEL_CHARS[BX]

                ;; zero => unknown model
                test     DL,  DL
                jz  _lookup_model_break

                ;; model code matches => exit
                cmp DL,  AL
                jz  _lookup_model_break

                inc BX
                jmp _lookup_model_loop

  _lookup_model_break:
                ;; we're addressing 2-byte words
                shl BX,  1
                mov DX,  MODEL_TAB[BX]

                pop BX
                ret
```

```
LOOKUP_MODEL ENDP

NEWLINE PROC
                push     DX
                mov DX,  OFFSET NEWLINE_STRING
                mov AH,  09h
                int 21h
                pop DX
                ret
NEWLINE ENDP

PRINT_MODEL PROC
                push     DX

                ;; assume the model name is in DX
                mov AH,  09h
                int 21h

                call     NEWLINE

                pop DX
                ret
PRINT_MODEL 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
```

```
                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

PRINT_VERSION_INFO PROC
                push    DX
                push    AX
                push    BP
                push    SI
                push    DI

                push    AX

                mov DX,  OFFSET DOS_VERSION_STRING
                mov AH,  09h
                int 21h

                pop AX

                ;; reserver max buffer we're going to need
                sub SP,  7
                mov BP,  SP
```

```
;; print first line
push     AX

lea SI,  [BP+2]
mov BYTE PTR [SI+1],    '$'
call     BYTE_TO_DEC

lea DX,  [SI+1]
mov AH,  09h
int 21h

mov AH,  02h
mov DL,  '.'
int 21h

pop AX

mov AL,  AH
lea SI,  [BP+2]
call     BYTE_TO_DEC

lea DX,  [SI+1]
mov AH,  09h
int 21h

call     NEWLINE

;; print second line
mov DX,  OFFSET OEM_NUMBER_STRING
mov AH,  09h
int 21h

mov AL,  BH
call     BYTE_TO_HEX
```

```
push     AX
mov DL,  AL
mov AH,  02h
int 21h

pop AX
mov DL,  AH
mov AH,  02h
int 21h

call     NEWLINE

;; print third line
mov DX,  OFFSET USER_SERIAL_STRING
mov AH,  09h
int 21h

lea SI,  [BP+5]
mov BYTE PTR [SI+1],    '$'
mov DI,  SI
mov AX,  CX
call     WRD_TO_HEX

mov AL,  BL
call     BYTE_TO_HEX
mov [SI-5],   AL
mov [SI-4],   AH

mov DX,  BP
mov AH,  09h
int 21h

call     NEWLINE
```

```
                add SP,  7

                pop DI
                pop SI
                pop BP
                pop AX
                pop DX
                ret
PRINT_VERSION_INFO ENDP

MAIN PROC FAR
                push    DS
                xor AX,  AX
                push    AX

                mov AX,  DATA
                mov DS,  AX

                ;; AX <- 0F000:0FFFEh
                push    DS
                mov AX,  0F000h
                mov DS,  AX
                mov BX,  0FFFEh
                mov AL,  BYTE PTR [BX]
                pop DS

                ;; DX <- model name string offset (rel. to
DS)
                call    LOOKUP_MODEL

                call    PRINT_MODEL

                ;; AL == 00h => return OEM number
                ;; AL <- DOS version major
                ;; AH <- DOS version minor
```

```
                ;; BL:CX <- `user serial number'
                ;; BH <- MS-DOS OEM number
                mov AX,  3000h
                int 21h

                call     PRINT_VERSION_INFO

                ret
MAIN ENDP

TEXT ENDS

                END MAIN
```