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. 7383 _____ Медведев И.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2019

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 работы

В ходе работы были написаны некоторые функции и структуры данных, которые описаны в табл. 1-2.

Таблица 1 – Используемые функции

Название функции	Выполняемая задача
TETR_TO_HEX	Перевод числа из 2-ой в 16-ричную систему счисления.
BYTE_TO_HEX	Осуществляет перевод байта, помещенного в AL, в два символа в шестнадцатеричной системе счисления, помещая результат в AX.
WRD_TO_HEX	Переводит числа размером 2 байта в 16-ричную систему счисления. В AX – число, в DI – адрес последнего символа.
BYTE_TO_DEC	Переводит в 10-ную систему счисления, SI – адрес поля младшей цифры.
Output	Вывод сообщения на экран.
PC_VERSION	Вывод строки, содержащей версию ПК.
VERSION_MS_DOS	Определение версии ОС

Окончание таблицы 1

SERIAL_NUMBER	Определение серийного номера пользователя
NUMBER_OEM	Определение номера OEM

Таблица 2 – Структуры данных

Название	Тип	Назначение
VersPC	db	Строка «Version PC»
VersMSDOS		Строка «Version MS-DOS: . »
NumOEM		Строка «Number OEM: »
SerNum		Строка «Serial Number: »
PC		Строка «PC»
PCXT		Строка «PC/XT»
AT		Строка «AT»
PS2_30		Строка «PS2 model 30»
PS2_80		Строка «PS2 model 80»
PC2_50_60		Строка «PS2 model 50/60»
PCjr		Строка «PCjr»
PC_Convert		Строка «PC Convertible»

Также в ходе работы были созданы .COM файл, «хороший» и «плохой» .EXE файлы. Результаты работы представлены на рис. 1-3.

Рисунок 1 – Запуск .COM файла.

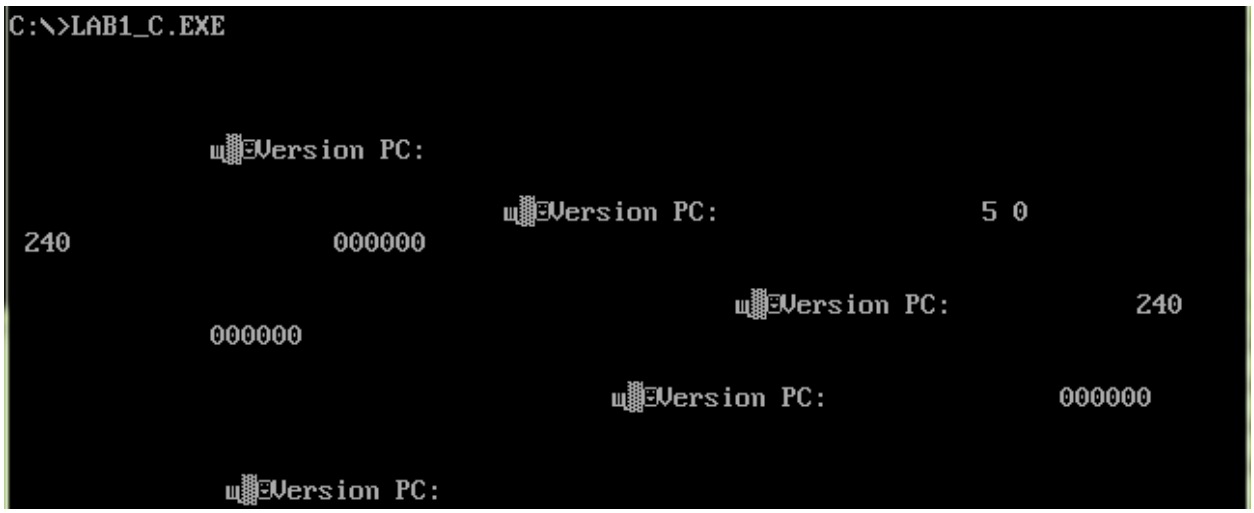

Рисунок 2 – Запуск «плохого» .EXE файла.

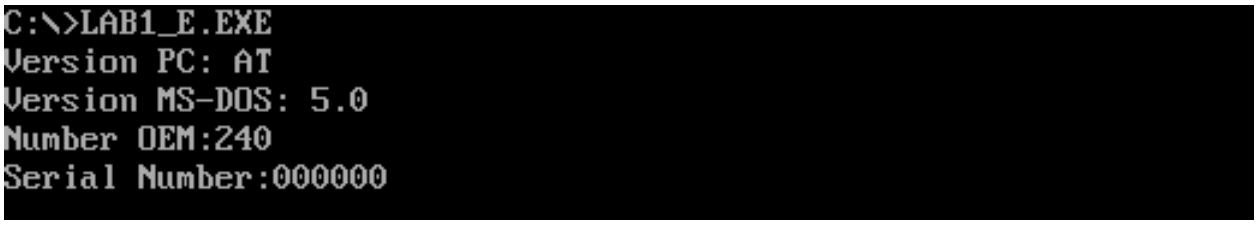

Рисунок 3 – Запуск «хорошего» .EXE файла.

Также были рассмотрены структуры файлов. Структуры представлены на рис. 4 – 6. В конце работы .COM и .EXE файлы был запущен с помощью отладчика, как показано на рис. 7 – 8.

Рисунок 4 – Структура .COM файла.

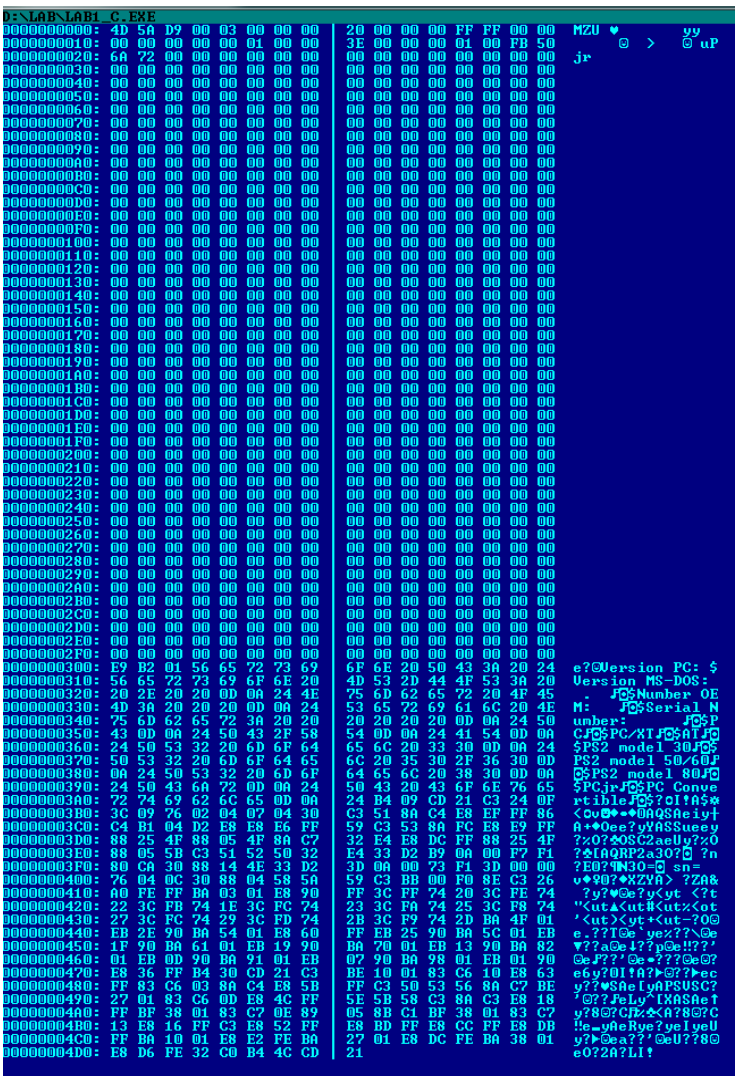

Рисунок 5 – Структура «плохого» .EXE файла.

Рисунок 5 – Структура «хорошего» .EXE файла.

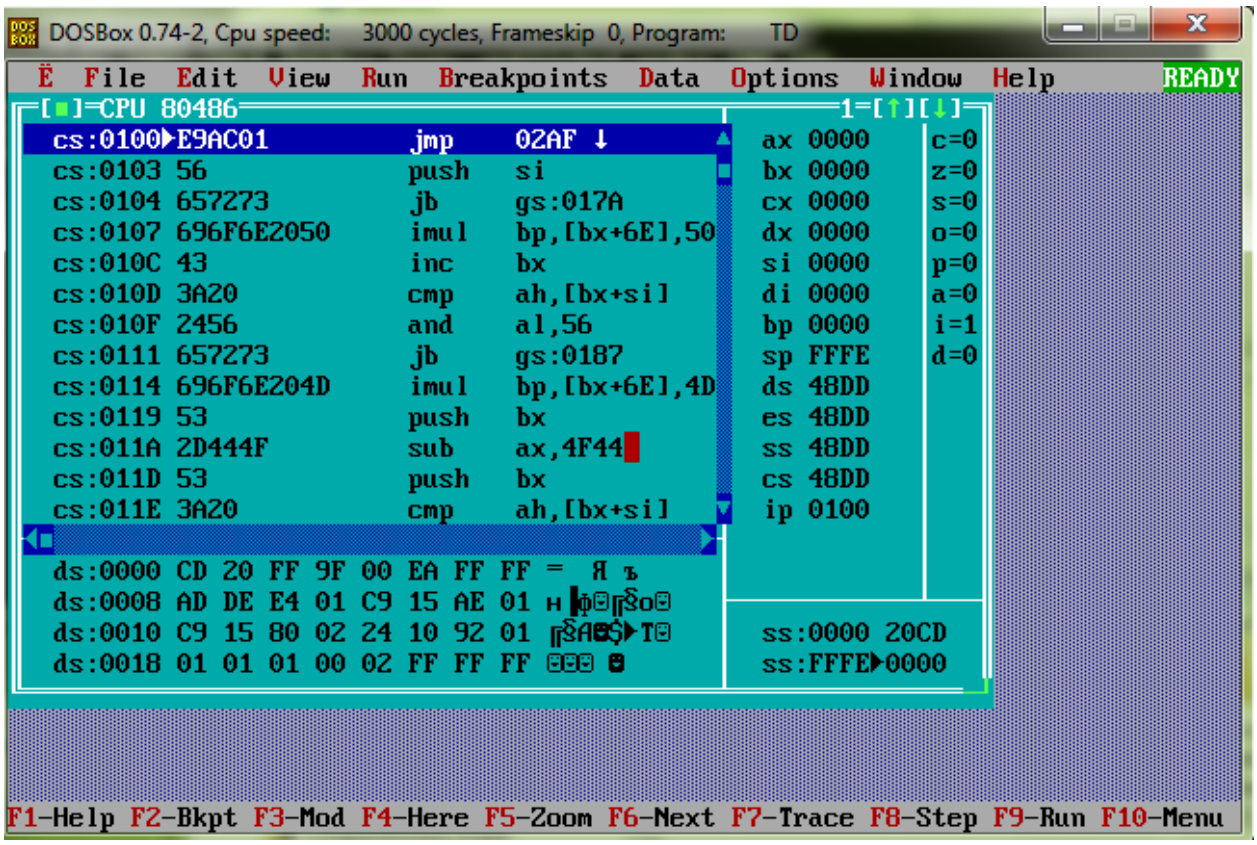

Рисунок 7 – Запуск .COM файла в дебагере.

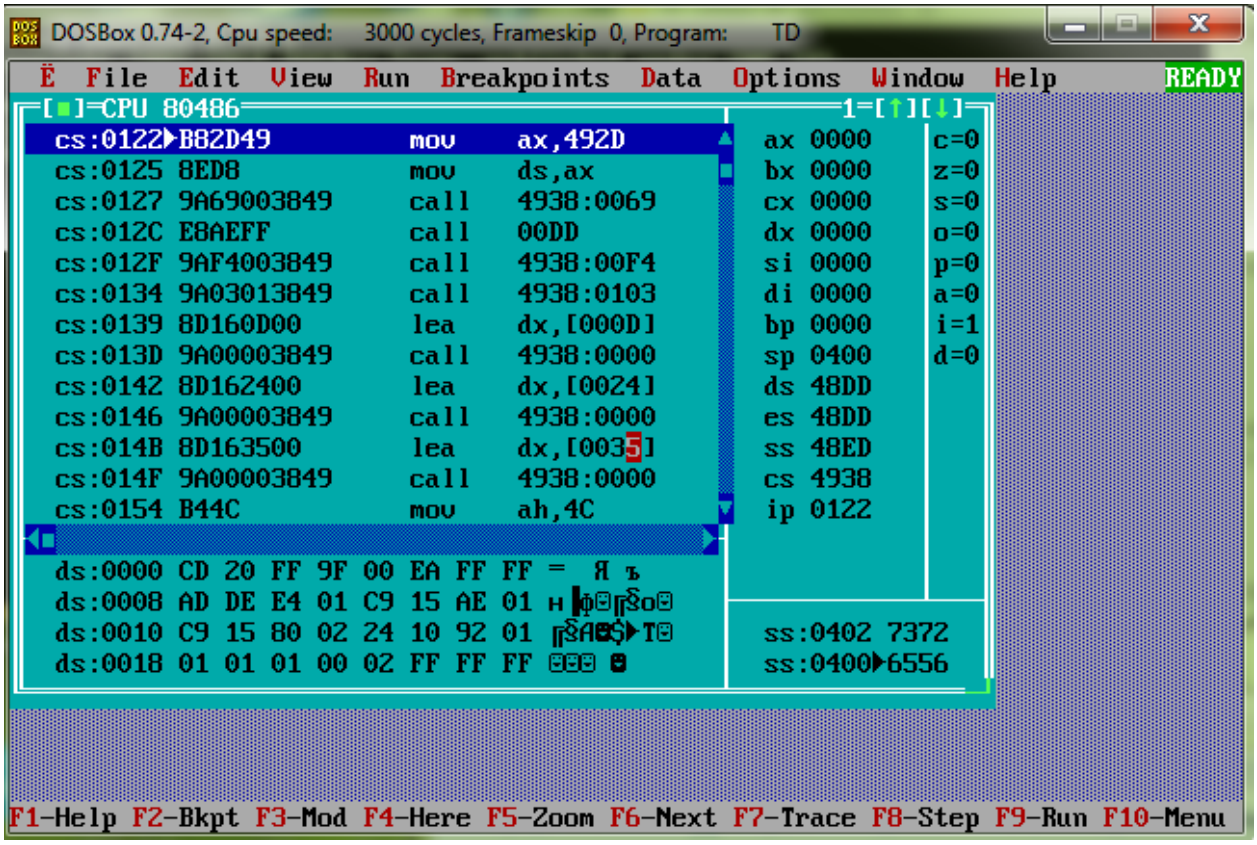

Рисунок 8 – Запуск .EXE файла в дебагере.

Просмотрев структуры и запустив структуры файлов, мы узнали, как располагаются те или иные сегменты.

Вывод

В ходе выполнения данной лабораторной работы были исследованы различия в структурах исходных текстов загрузочный модулей типов .COM и .EXE, структур файлов этих модулей и способах их загрузки в основную память.

Ответы на контрольные вопросы

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

COM-программа содержит 1 сегмент.

2) Сколько сегментов должна содержать EXE-программа?

Три сегмента (сегмент стека, сегмент кода и сегмент данных).

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

Обязательная должна быть директива ORG 100h. Она нужна по той причине, что при загрузке модуля в ОП в начале COM-программы резервируется 256 байт для PSP. Так же обязательно должна быть директива ASSUME. Эта директива указывает, что необходимо привязать сегментный регистр к данному сегменту. MASM может обойтись и без этой директивы, а TASM выдаст сообщение об ошибке “Near jump or call to different CS”.

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

Нет. Нельзя использовать команды, которые используют адреса сегментов, т.к. отсутствует таблица настроек, которая состоит из длинных указателей (смещение: сегмент), которые занимают по два байта. Они указывают слова в загрузочном модуле, содержащие адрес, который должен быть настроен на место памяти, в которое загружается задача.

Отличия форматов файлов COM и EXE модулей

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

COM-файл содержит единственный сегмент, в котором данные и машинные команды. Код начинается с адреса 100h.

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

В «плохом» EXE данные и код содержатся в одном сегменте. С адреса 0 до 1B располагается информация для загрузчика и образует так называемый заголовок. После располагается таблица настроек (см. 4 вопрос раздела «Отличия исходных текстов COM и EXE программ»), размер которой записан в байтах 06-07 заголовка. Затем начинается сегмент кода с адреса 300h.

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

Как и в плохом, с адреса 0 располагается заголовок и таблица настроек. Затем располагается сегмент стека длиной 400h, после сегмент данных с адреса 600h, а затем сегмент кода с длиной 660h.

Загрузка COM модуля в основную память

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

В ОП определяется адрес участка ОП, где достаточно места для загрузки модуля. Первые 100h байт занимает PSP, регистр SP устанавливается по адресу FFEEh, в регистр IP записывается значение 100h, сегмент кода CS начинается с адреса 48DDh.

2) Что располагается с адреса 0?

PSP.

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

При загрузке программы они указывают на начало PSP (48DDh).

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

Регистр SP указывает на конец стека (FFFFh), SS – на начало (0h). Адреса расположены в диапазоне 0h – FFFEh

Загрузка «хорошего» EXE модуля в основную память

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

DS и ES устанавливаются на начало сегмента PSP (48DDh), SS – на начало сегмента стека (48Edh), CS – на начало сегмента команд (4938h).

2) На что указывают регистры DS и ES?

На начало PSP.

3) Как определяется стек?

Стек определяется с помощью директивы STACK.

4) Как определяется точка входа?

Точка входа определяется директивой END. Смещение точки входа загружается в регистр IP.

ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ

lab1_c.asm

```
TESTPC SEGMENT
            ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
            ORG 100H
START:      JMP   BEGIN
;Данные
VersPC      db    'Version PC: $'
VersMSDOS   db    'Version MS-DOS:  .  ',0DH,0AH,'$'
NumOEM      db    'Number OEM:   ',0DH,0AH,'$'
SerNum      db    'Serial Number:      ',0DH,0AH,'$'

PC          db 'PC',0DH,0AH,'$'
PCXT        db 'PC/XT',0DH,0AH,'$'
AT                db 'AT',0DH,0AH,'$'
PS2_30            db 'PS2 model 30',0DH,0AH,'$'
PS2_50_60   db 'PS2 model 50/60',0DH,0AH,'$'
PS2_80            db 'PS2 model 80',0DH,0AH,'$'
PCjr        db 'PCjr',0DH,0AH,'$'
PC_Convert  db 'PC Convertible',0DH,0AH,'$'



Output            PROC near
            mov   ah,09h
            int   21h
            ret
Output            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 шестн. числа в AX
            push  cx
            mov   al,ah
            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
            xor   ah,ah
            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
            push  ax
            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   ax,00h
            jbe   end_l
            or    al,30h
            mov   [si],al
end_l:            pop   ax
            pop   dx
            pop   cx
            ret
BYTE_TO_DEC       ENDP



PC_VERSION  PROC near

            mov   bx,0F000h
            mov   es,bx
            mov   al,es:[0FFFEh]
            mov   dx,offset VersPC
            call Output
            cmp al,0FFh
            je PC_lab

            cmp al,0FEh
            je PCXT_lab

            cmp al,0FBh
```

```
je PCXT_lab

cmp al,0FCh
je AT_lab

cmp al,0FAh
je PS2_30_lab

cmp al,0F8h
je PS2_50_60_lab

cmp al,0FCh
je PS2_80_lab

cmp al,0FDh
je PCjr_lab

cmp al,0F9h
je PCConvert_lab

PC_lab:
      mov dx, offset PC
      jmp end_lab
PCXT_lab:
      mov dx, offset PCXT
      call Output
      jmp end_lab
AT_lab:
      mov dx, offset AT
      jmp end_lab
PS2_30_lab:
      mov dx, offset PS2_30
      jmp end_lab
PS2_50_60_lab:
      mov dx, offset PS2_50_60
      jmp end_lab

PS2_80_lab:
      mov dx, offset PS2_80
      jmp end_lab
PCjr_lab:
      mov dx, offset PCjr
      jmp end_lab
PCConvert_lab:
      mov dx, offset PC_Convert
      jmp end_lab

end_lab:
      call Output
      mov   ah,30h
      int   21h
      ret
```

```
PC_VERSION  ENDP


VERSION_MS_DOS          PROC near

            mov   si,offset VersMSDOS
            add   si,16
            call  BYTE_TO_DEC
            add   si,3
            mov   al,ah
            call        BYTE_TO_DEC

            ret
VERSION_MS_DOS    ENDP


NUMBER_OEM              PROC near

            mov   al,bh
            lea   si,NumOEM
            add   si,13
            call  BYTE_TO_DEC

            ret
NUMBER_OEM              ENDP

SERIAL_NUMBER           PROC near

            mov   al,bl
            call  BYTE_TO_HEX
            lea   di,SerNum
            add   di,14
            mov   [di],ax
            mov   ax,cx
            lea   di,SerNum
            add   di,19
            call  WRD_TO_HEX

            ret
SERIAL_NUMBER           ENDP


BEGIN:
            call  PC_VERSION
            call  VERSION_MS_DOS
            call  NUMBER_OEM
            call  SERIAL_NUMBER

            lea   dx,VersMSDOS
            call  Output
            lea   dx,NumOEM
            call  Output
            lea   dx,SerNum
            call  Output
```

```
            xor   al,al
            mov   ah,4ch
            int   21h

TESTPC      ENDS
            END   START
```

lab1_e.asm

```
EOL EQU '$'
AStack SEGMENT STACK
            DW 512 DUP(?)
AStack ENDS

DATA  SEGMENT
      VersPC      db    'Version PC: $'
      VersMSDOS   db    'Version MS-DOS:  .  ',0DH,0AH,'$'
      NumOEM      db    'Number OEM:   ',0DH,0AH,'$'
      SerNum      db    'Serial Number:       ',0DH,0AH,'$'

      PC                db 'PC',0DH,0AH,'$'
      PCXT        db 'PC/XT',0DH,0AH,'$'
      AT                db 'AT',0DH,0AH,'$'
      PS2_30            db 'PS2 model 30',0DH,0AH,'$'
      PS2_50_60   db 'PS2 model 50/60',0DH,0AH,'$'
      PS2_80            db 'PS2 model 80',0DH,0AH,'$'
      PCjr        db 'PCjr',0DH,0AH,'$'
      PC_Convert  db 'PC Convertible',0DH,0AH,'$'
DATA  ENDS


CODE  SEGMENT
      ASSUME CS:CODE,   DS:DATA, SS:AStack
Output            PROC FAR
            mov   ah,09h
            int   21h
            ret
Output            ENDP

TETR_TO_HEX       PROC  FAR
            and   al,0fh
            cmp   al,09
            jbe   NEXT
            add   al,07
NEXT:       add   al,30h
            ret
TETR_TO_HEX       ENDP

BYTE_TO_HEX       PROC FAR
            push  cx
            mov   al,ah
            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  FAR
            push  bx
            mov   bh,ah
            call  BYTE_TO_HEX
            mov   [di],ah
            dec   di
            mov   [di],al
            dec   di
            mov   al,bh
            xor   ah,ah
            call  BYTE_TO_HEX
            mov   [di],ah
            dec   di
            mov   [di],al
            pop   bx
            ret
WRD_TO_HEX        ENDP

BYTE_TO_DEC       PROC  FAR
            push  cx
            push  dx
            push  ax
            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   ax,00h
            jbe   end_l
            or    al,30h
            mov   [si],al
end_l:            pop   ax
            pop   dx
            pop   cx
            ret
BYTE_TO_DEC       ENDP

PC_VERSION        PROC FAR

            mov   bx,0F000h
            mov   es,bx
            mov   al,es:[0FFFEh]
            mov   dx,offset VersPC
            call Output
```

```
            cmp al,0FFh
            je PC_lab
            cmp al,0FEh
            je PCXT_lab
            cmp al,0FBh
            je PCXT_lab
            cmp al,0FCh
            je AT_lab
            cmp al,0FAh
            je PS2_30_lab
            cmp al,0F8h
            je PS2_80_lab
            cmp al,0FDh
            je PCjr_lab
            cmp al,0F9h
            je PCConvert_lab
            PC_lab:
                  mov dx, offset PC
                  jmp end_lab
            PCXT_lab:
                  mov dx, offset PCXT
                  jmp end_lab
            AT_lab:
                  mov dx, offset AT
                  jmp end_lab
            PS2_30_lab:
                  mov dx, offset PS2_30
                  jmp end_lab
            PS2_80_lab:
                  mov dx, offset PS2_80
                  jmp end_lab
            PCjr_lab:
                  mov dx, offset PCjr
                  jmp end_lab
            PCConvert_lab:
                  mov dx, offset PC_Convert
                  jmp end_lab

            end_lab:
                  call Output
                  mov   ah,30h
                  int   21h
                  ret


PC_VERSION  ENDP


VERSION_MS_DOS    PROC NEAR

            lea   si,VersMSDOS
            add   si,16
            call  BYTE_TO_DEC
            add   si,3
            mov   al,ah
```

```
            call        BYTE_TO_DEC

            ret
VERSION_MS_DOS    ENDP


NUMBER_OEM              PROC FAR

            mov   al,bh
            lea   si,NumOEM
            add   si,13
            call  BYTE_TO_DEC

            ret
NUMBER_OEM  ENDP


SERIAL_NUMBER     PROC FAR
            mov   al,bl
            call  BYTE_TO_HEX
            lea   di,SerNum
            add   di,14
            mov   [di],ax
            mov   ax,cx
            lea   di,SerNum
            add   di,19
            call  WRD_TO_HEX

            ret
SERIAL_NUMBER           ENDP



Main              PROC FAR

            mov   ax, DATA
            mov   ds,ax
            call  PC_VERSION
            call  VERSION_MS_DOS
            call  NUMBER_OEM
            call  SERIAL_NUMBER

            lea   dx,VersMSDOS
            call  Output
            lea   dx,NumOEM
            call  Output
            lea   dx,SerNum
            call  Output

            mov   ah,4ch
            int   21h
            ret
Main              ENDP
CODE              ENDS
                  END Main
```