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 _____ Ласковенко Е.А.

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.

Описание функций и структур данных, используемых в программе, представлены в табл. 1—2.

Таблица 1 — Описание функций

Название функции	**Назначение**
TETR_TO_HEX	Необходима для работы функции BYTE_TO_HEX
BYTE_TO_HEX	Перевод байта в регистре AL в два символа числа в 16-ой с/с в AX
WRD_TO_HEX	Перевод в 16с/с 16-ти разрядного числа. В AX - число, DI - адрес последнего символа
BYTE_TO_DEC	Перевод в 10с/с, SI – адрес поля младшей цифры
PRINT_STR	Вывод строки на экран
GET_OS_TYPE	Получение типа PC из предпоследнего байта ROM BIOS, вывод строки OS_TYPE и строки типа PC на экран
GET_OS_VERSION	Формирование строки OS_VERSION
GET_OS_OEM	Формирование строки OS_OEM
GET_SERIAL_NUM	Формирование строки SERIAL_NUM

Таблица 2 — Описание структур данных

Название	**Тип**	**Назначение**
OS_TYPE	db	Тип ОС
OS_VERSION	db	Версия ОС
OS_OEM	db	Серийный номер OEM
SERIAL_NUM	db	Серийный номер пользователя
PC	db	PC
PCXT	db	PC/XT
AT	db	AT
PS2_30	db	PS2 модель 30
PS2_80	db	PS2 модель 80
PCjr	db	PCjr
PC_Cnv	db	PC Convertible

Программа считывает необходимые значения, формирует соответствующие строки и выводит их на экран. Порядок вывода строк следующий: тип ОС, который определяется сравнением с табличными значениями, версия ОС, серийный номер OEM и серийный номер пользователя.

На рис. 1—3 приведены скриншоты, подтверждающие выполнение шагов лабораторной работы.

Рисунок 1 – Результат выполнения программы COM.COM

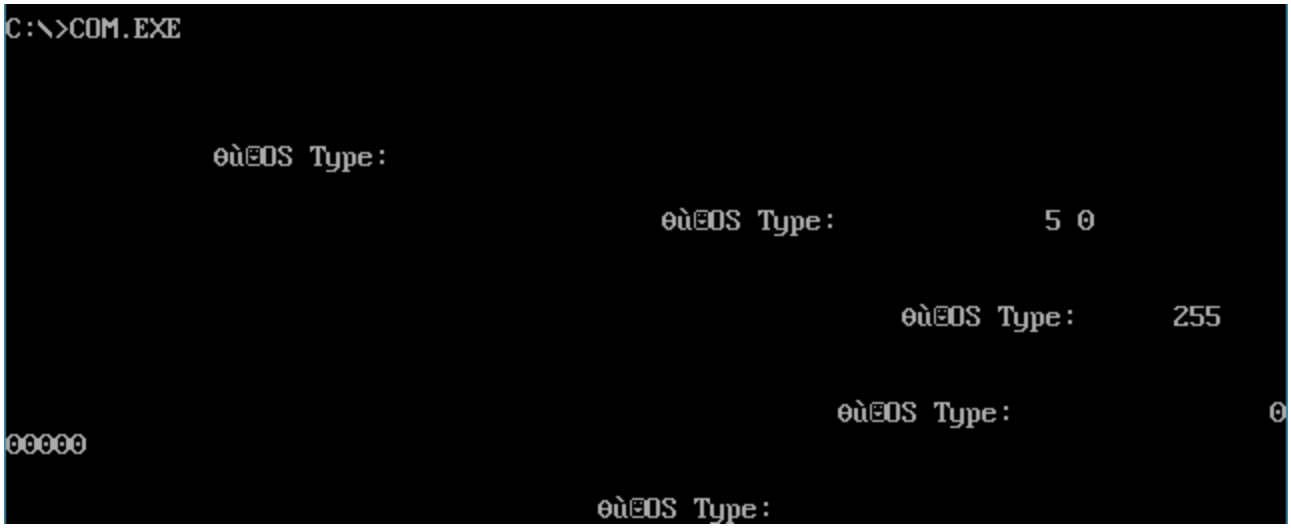

Рисунок 2 – Результат выполнения программы COM.EXE

Рисунок 3 – Результат выполнения программы GOOD_EXE.EXE

Выводы.

В результате выполнения данной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, различия структур файлов загрузочных модулей и способов 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 должна содержать только 1 сегмент.

2. EXE-программа?

Минимум 1 сегмент, но в отличие от COM-программы может состоять из нескольких сегментов.

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

Обязательными директивами являются: директива ASSUME, ставящая в соответствие сегментным регистрам соответствующие сегменты и директива

ORG 100h, которая резервирует 256 байт (100h) для PSP (заполнять PSP будет система).

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

В COM-программе нельзя использовать такие команды, как MOV регистр, сегмент (CODE, DATA). Также нельзя использовать команды, содержащие дальнюю адресацию, так как для выполнения этих команд используется таблица настройки, однако в COM-программе нет таблицы настройки, она есть только в EXE-файлах.

В отличие от программы типа .COM, в программе типа .EXE могут использоваться команды, использующие значения сегментных адресов програмных сегментов. Для выполнения этих команд требуется таблица настройки, содержащая всю необходимую информацию. Таблица настройки состоит из указателей вида “смещение: сегмент” на те слова в загрузочном модуле, которые содержат настраиваемые сегментные адреса. Размер таблицы настройки зависит от количества команд программы, в которых необходимо определять физический адрес того или иного программного сегмента.

Отличия форматов файлов COM и EXE модулей:

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

COM-файл содержит в себе только код и данные, в нем отсутствует таблица настроек, следовательно, в файле код располагается с нулевого адреса, как показано на рис. 4.

Рисунок 4 – Содержание .COM файла в FAR

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

В «плохом» EXE нет разделения по сегментам. С нулевого адреса располагается заголовок, как показано на рис. 5, за ним следует таблица настройки. Код располагается с адреса 300h, как показано на рис.6, (200h байт занимает заголовок + сдвиг на 100h дает директива ORG 100h).

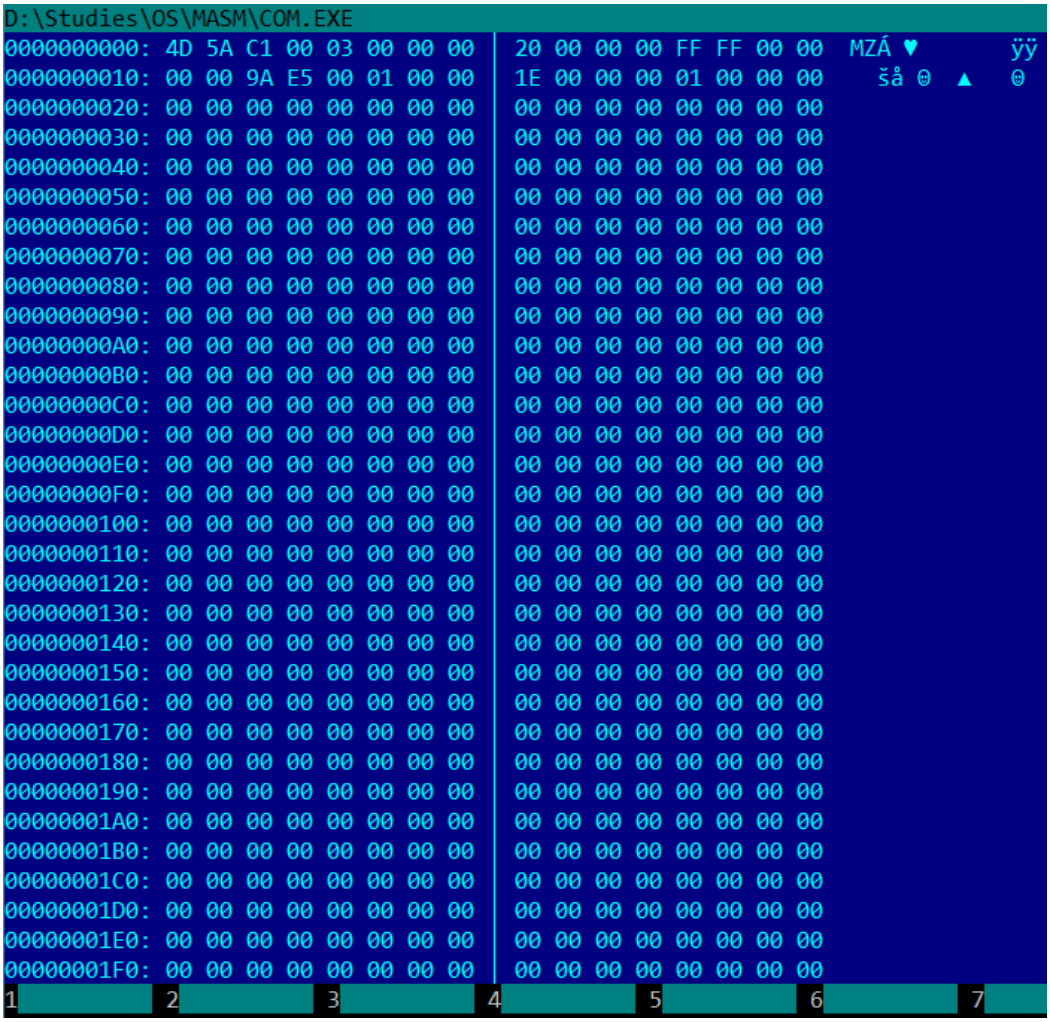

Рисунок 5 – Содержание «плохого» .EXE файла в FAR

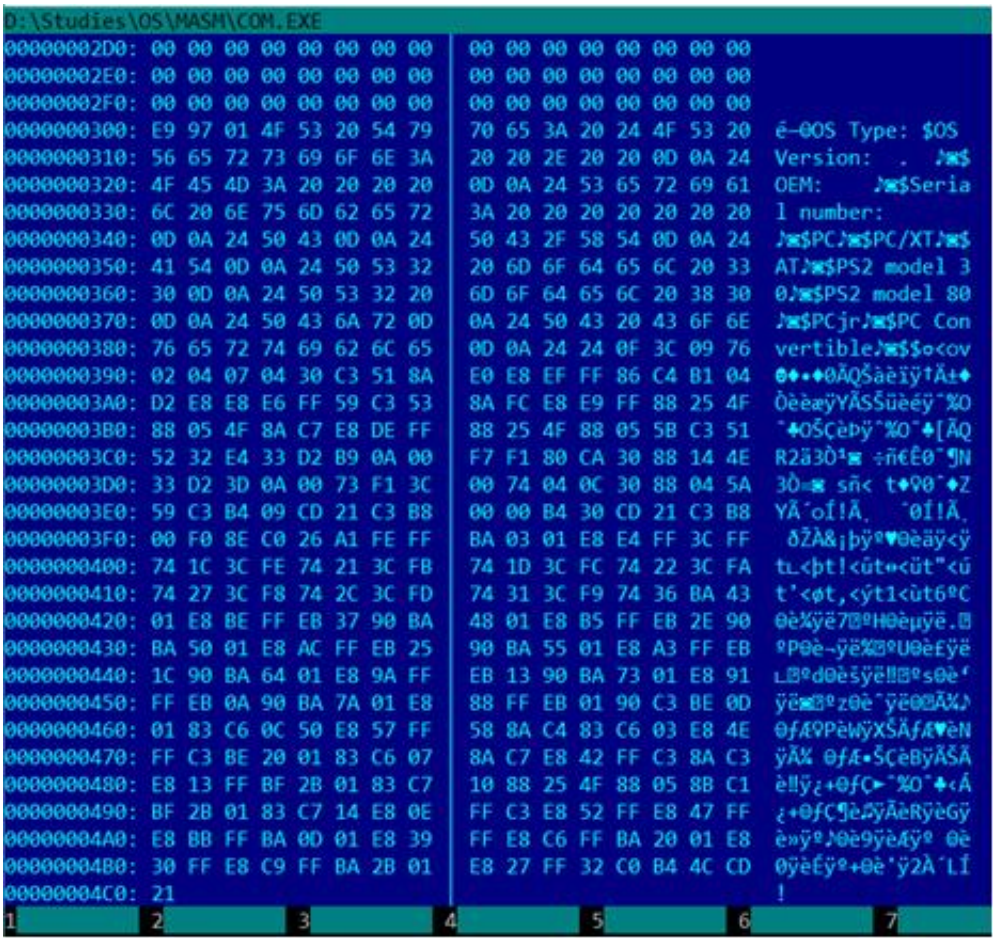

Рисунок 6 – Содержание «плохого» .EXE файла в FAR

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

В «хорошем» EXE код, в отличие от «плохого», стек и данные выделены в отдельные сегменты и дополнительно выделено под стек 010h. Соответственно структура файла следующая: 0-200h заголовок и таблица настроек, 200h–220h стек, 220h–410h сегмент данных и сегмент кода. В «хорошем» EXE не используется директива ORG 100h, так как загрузчик автоматически располагает программу после PSP. Структура файла «хорошего» EXE преставлена на рис. 7—8.

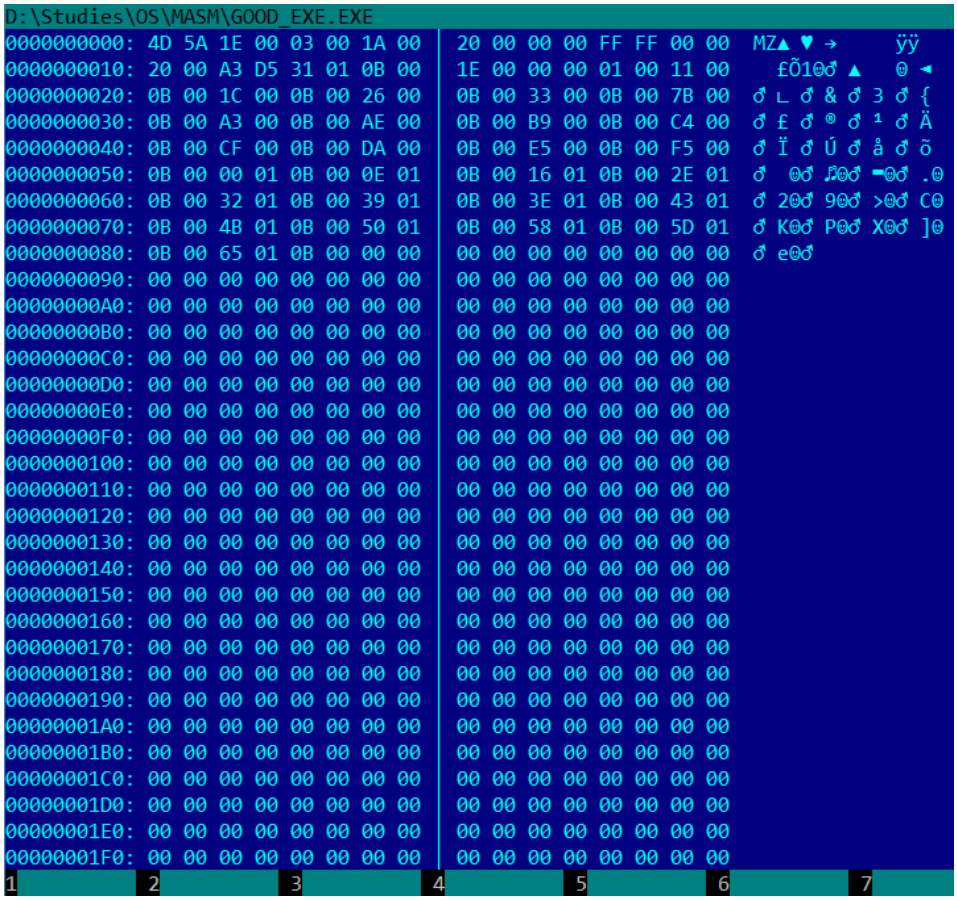

Рисунок 7 – Содержание «хорошего» .EXE файла в FAR

Рисунок 8 – Содержание «хорошего» .EXE файла в FAR

Загрузка COM модуля в основную память:

Результат запуска COM модуля под управлением отладчика представлен на рис. 9.

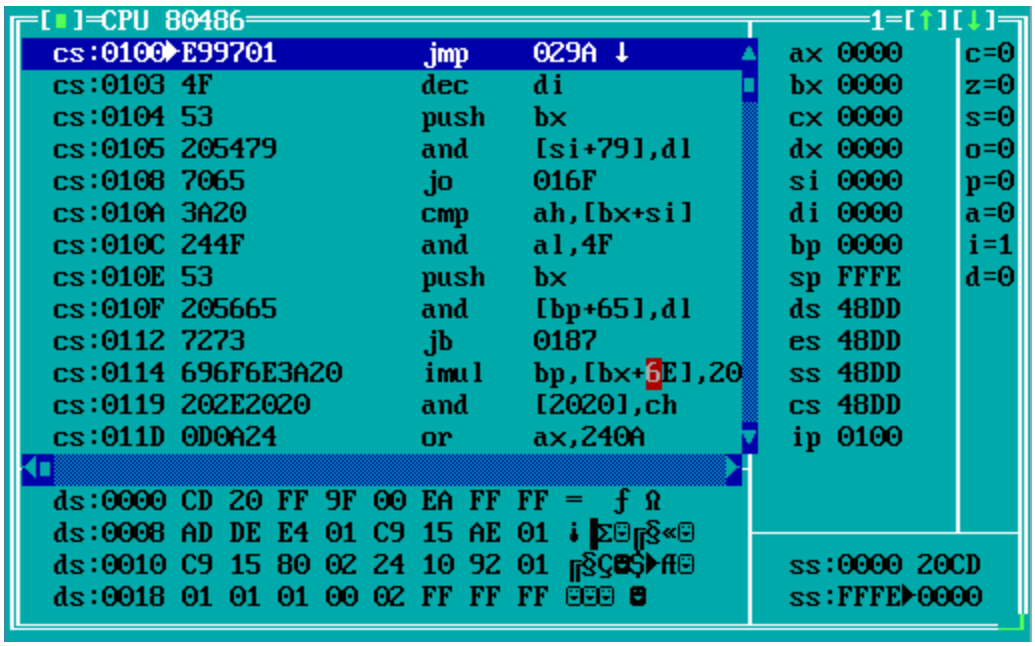

Рисунок 9 — Результат загрузки COM модуля в основную память

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

Сначала система выделяет сегмент памяти для модуля и все сегментные регистры устанавливает на начало выделенного сегмента памяти. В первых 100h байтах памяти, начиная с выделенного сегмента памяти, устанавливается PSP, потом загружается код COM-файла и IP присваивается значение 100h, а регистр SP устанавливается в конец сегмента памяти.

Код начинается с адреса 48DD – адрес CS в данном случае.

2. Что располагается с адреса 0?

С нулевого адреса располагается PSP.

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

Сегментные регистры имеют значение 48DD (см. рис.9). Они указывают на PSP.

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

Стек занимает весь сегмент памяти COM-программы. Стек определяется регистрами SS и SP: в SS находится адрес начала сегмента, а в SP — его конец.

Адреса, которые занимает сегмент стека, расположены в диапазоне 0000h-FFFEh.

Загрузка «хорошего» EXE модуля в основную память:

Результат запуска «хорошего» .EXE под управлением отладчика представлен на рис. 10.

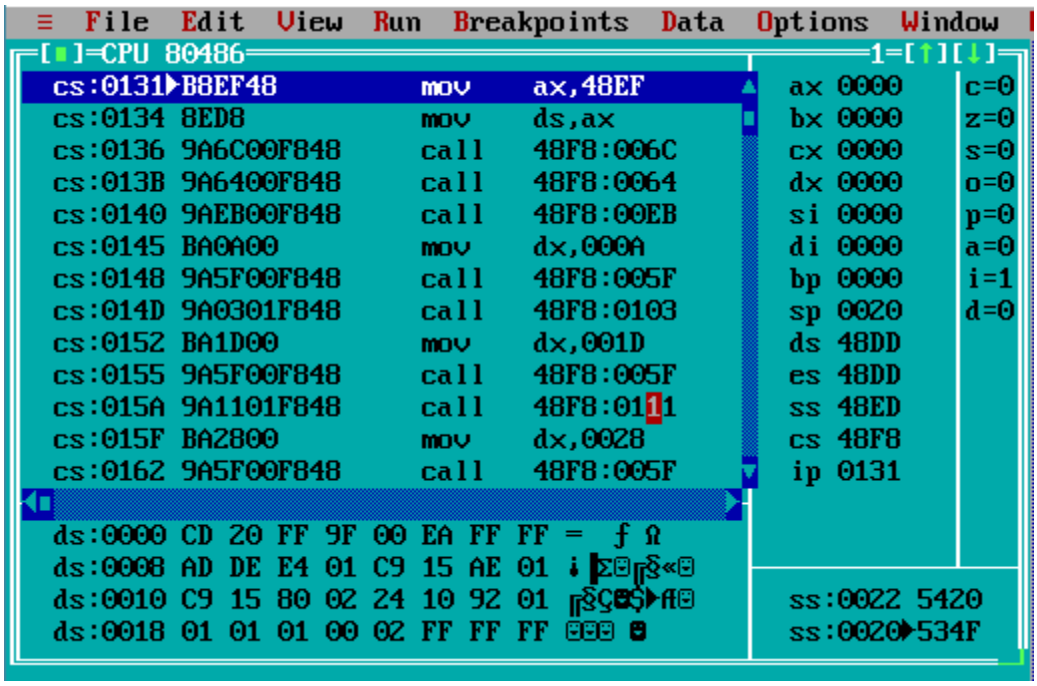

Рисунок 10 – Результат загрузки «хорошего» .EXE в основную память

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

При загрузке модуля программы в память система ставит в начало программы префикс программы PSP размером 256 байт. После загрузки система инициализирует сегментные регистры так, что регистры DS и ES указывают на начало PSP, CS – на начало сегмента команд, SS – на начало сегмента стека. В IP загружается смещение точки входа в программу, а в SP – смещение конца сегмента стека.

Как видно из рис. 10, адреса регистров в примере следующие: CS = 4936 – начало сегмента команд, SS = 48ED – начало сегмента стека, DS = ES = 48DD – начало PSP.

2. На что указывают регистры DS и ES?

Регистры DS и ES указывают на начало PSP.

3. Как определяется стек?

В исходном коде модуля стек определяется при помощи директивы STACK. При загрузке выполнимого модуля в регистр SS записывается адрес начала сегмента стека, а в SP – его вершины.

4. Как определяется точка входа?

Значение точки входа в программу берется из операнда директивы END. После этой директивы указывается метка, куда переходит программа при запуске.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММ

1. COM.ASM

```
TESTPC SEGMENT
       ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
       ORG 100H
START: JMP BEGIN

; ДАННЫЕ
OS_TYPE      db 'OS Type: $'
OS_VERSION   db 'OS Version:  .  ',0DH,0AH,'$'
OS_OEM             db 'OEM:    ',0DH,0AH,'$'
SERIAL_NUM   db 'Serial number:       ',0DH,0AH,'$'

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_80             db 'PS2 model 80',0DH,0AH,'$'
PCjr         db 'PCjr',0DH,0AH,'$'
PC_Cnv             db 'PC Convertible',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
       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_STR PROC near
	mov AH,09h
	int 21h
	ret
PRINT_STR ENDP

GET_OS_INFO PROC near
; Вызов функции 30h прерывания 21h
	mov AX,0
	mov AH,30h
	int 21h
	ret
GET_OS_INFO ENDP

GET_OS_TYPE PROC near
	; Загрузка в регистр AX данных по адресу предпоследнего бита ROM BIOS
	mov AX,0F000h
	mov ES,AX
	mov AX,ES:0FFFEh

	; Вывод строки OS_TYPE на экран
	mov DX,OFFSET OS_TYPE
	call PRINT_STR

	; Сравнение
	cmp AL,0FFh
	je PC_label
	cmp AL,0FEh
	je PCXT_label
	cmp AL,0FBh
	je PCXT_label
	cmp AL,0FCh
	je AT_label
```

```
	cmp AL,0FAh
	je PS2_30_label
	cmp AL,0F8h
	je PS2_80_label
	cmp AL,0FDh
	je PCjr_label
	cmp AL,0F9h
	je PC_Convertible_label
	PC_label:
		mov DX,OFFSET PC
		call PRINT_STR
		jmp end_label
	PCXT_label:
		mov DX,OFFSET PCXT
		call PRINT_STR
		jmp end_label
	AT_label:
		mov DX,OFFSET AT
		call PRINT_STR
		jmp end_label
	PS2_30_label:
		mov DX,OFFSET PS2_30
		call PRINT_STR
		jmp end_label
	PS2_80_label:
		mov DX,OFFSET PS2_80
		call PRINT_STR
		jmp end_label
	PCjr_label:
		mov DX,OFFSET PCjr
		call PRINT_STR
		jmp end_label
	PC_Convertible_label:
		mov DX,OFFSET PC_Cnv
		call PRINT_STR
		jmp end_label

end_label:
	ret
GET_OS_TYPE ENDP


GET_OS_VERSION PROC near
	; Формирование строки OS_VERSION: номер основной версии
	mov SI,OFFSET OS_VERSION
	add SI,12
	push AX
	call BYTE_TO_DEC

	; Формирование строки OS_VERSION: номер модификации
	pop AX
	mov AL,AH
	add SI,3
	call BYTE_TO_DEC

	ret
GET_OS_VERSION  ENDP


GET_OS_OEM PROC near
	; Формирование строки OS_OEM
```

```
            mov SI,OFFSET OS_OEM
            add SI,7
            mov AL,BH
            call BYTE_TO_DEC

            ret
      GET_OS_OEM   ENDP


      GET_SERIAL_NUM PROC near
            ; Формирование строки SERIAL_NUM: первые 8 бит номера
            mov AL,BL
            call BYTE_TO_HEX
            mov DI,OFFSET SERIAL_NUM
            add DI,16
            mov [DI],AH
            dec DI
            mov [DI],AL

            ; Формирование строки SERIAL_NUM: оставшиеся 16 бит номера
            mov AX,CX
            mov DI,OFFSET SERIAL_NUM
            add DI,20
            call WRD_TO_HEX

            ret
      GET_SERIAL_NUM     ENDP


      BEGIN:
            call GET_OS_TYPE

            call GET_OS_INFO

            call GET_OS_VERSION
            mov DX,OFFSET OS_VERSION
            call PRINT_STR

            call GET_OS_OEM
            mov DX,OFFSET OS_OEM
            call PRINT_STR

            call GET_SERIAL_NUM
            mov DX,OFFSET SERIAL_NUM
            call PRINT_STR

            xor AL,AL
            mov AH,4Ch
            int 21H
      TESTPC ENDS
 END START
```

2. GOOD_EXE.ASM

```
AStack SEGMENT STACK
      DW 010h DUP(?)
AStack ENDS

; ДАННЫЕ
DATA SEGMENT
```

```
OS_TYPE      db 'OS Type: $'
OS_VERSION   db 'OS Version:  .  ',0DH,0AH,'$'
OS_OEM              db 'OEM:    ',0DH,0AH,'$'
SERIAL_NUM   db 'Serial number:       ',0DH,0AH,'$'

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_80              db 'PS2 model 80',0DH,0AH,'$'
PCjr         db 'PCjr',0DH,0AH,'$'
PC_Cnv              db 'PC Convertible',0DH,0AH,'$'
DATA ENDS

; ПРОЦЕДУРЫ
CODE SEGMENT
       ASSUME CS:CODE, DS:DATA, SS:AStack

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 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 FAR
       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 FAR
       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_STR PROC FAR
       mov AH,09h
       int 21h
       ret
PRINT_STR ENDP

GET_OS_INFO PROC FAR
; Вызов функции 30h прерывания 21h
       mov AX,0
       mov AH,30h
       int 21h
       ret
GET_OS_INFO ENDP

GET_OS_TYPE PROC FAR
       ; Загрузка в регистр AX данных по адресу предпоследнего бита ROM BIOS
       mov AX,0F000h
       mov ES,AX
       mov AX,ES:0FFFEh

       ; Вывод строки OS_TYPE на экран
       mov DX,OFFSET OS_TYPE
       call PRINT_STR

       ; Сравнение
       cmp AL,0FFh
       je PC_label
       cmp AL,0FEh
       je PCXT_label
       cmp AL,0FBh
       je PCXT_label
       cmp AL,0FCh
       je AT_label
       cmp AL,0FAh
       je PS2_30_label
       cmp AL,0F8h
       je PS2_80_label
       cmp AL,0FDh
       je PCjr_label
       cmp AL,0F9h
       je PC_Convertible_label
```

```
	PC_label:
		mov DX,OFFSET PC
		call PRINT_STR
		jmp end_label
	PCXT_label:
		mov DX,OFFSET PCXT
		call PRINT_STR
		jmp end_label
	AT_label:
		mov DX,OFFSET AT
		call PRINT_STR
		jmp end_label
	PS2_30_label:
		mov DX,OFFSET PS2_30
		call PRINT_STR
		jmp end_label
	PS2_80_label:
		mov DX,OFFSET PS2_80
		call PRINT_STR
		jmp end_label
	PCjr_label:
		mov DX,OFFSET PCjr
		call PRINT_STR
		jmp end_label
	PC_Convertible_label:
		mov DX,OFFSET PC_Cnv
		call PRINT_STR
		jmp end_label

end_label:
	ret
GET_OS_TYPE ENDP


GET_OS_VERSION PROC FAR
	; Формирование строки OS_VERSION: номер основной версии
	mov SI,OFFSET OS_VERSION
	add SI,12
	push AX
	call BYTE_TO_DEC

	; Формирование строки OS_VERSION: номер модификации
	pop AX
	mov AL,AH
	add SI,3
	call BYTE_TO_DEC

	ret
GET_OS_VERSION  ENDP


GET_OS_OEM PROC FAR
	; Формирование строки OS_OEM
	mov SI,OFFSET OS_OEM
	add SI,7
	mov AL,BH
	call BYTE_TO_DEC

	ret
GET_OS_OEM   ENDP
```

```
GET_SERIAL_NUM PROC FAR
       ; Формирование строки SERIAL_NUM: первые 8 бит номера
       mov AL,BL
       call BYTE_TO_HEX
       mov DI,OFFSET SERIAL_NUM
       add DI,16
       mov [DI],AH
       dec DI
       mov [DI],AL

       ; Формирование строки SERIAL_NUM: оставшиеся 16 бит номера
       mov AX,CX
       mov DI,OFFSET SERIAL_NUM
       add DI,20
       call WRD_TO_HEX

       ret
GET_SERIAL_NUM     ENDP


START PROC FAR
       mov AX,DATA
       mov DS,AX

       call GET_OS_TYPE

       call GET_OS_INFO

       call GET_OS_VERSION
       mov DX,OFFSET OS_VERSION
       call PRINT_STR

       call GET_OS_OEM
       mov DX,OFFSET OS_OEM
       call PRINT_STR

       call GET_SERIAL_NUM
       mov DX,OFFSET SERIAL_NUM
       call PRINT_STR

       xor AL,AL
       mov AH,4Ch
       int 21H

       ret
START ENDP
CODE ENDS
END START
```