

**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

<b>Название функции</b>	<b>Назначение</b>
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.

```
C:\>COM.COM
OS Type: AT
OS Version: 5.0
OEM: 255
Serial number: 000000
```

Рисунок 1 – Результат выполнения программы COM.COM

```
C:\>COM.EXE

        OS Type :

                                OS Type :          5 0

                                OS Type :          255

                                OS Type :          0

000000

                                OS Type :
```

Рисунок 2 – Результат выполнения программы COM.EXE

```
C:\>GOOD_EXE.EXE
OS Type: AT
OS Version: 5.0
OEM: 255
Serial number: 0000000
```

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 программных 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.



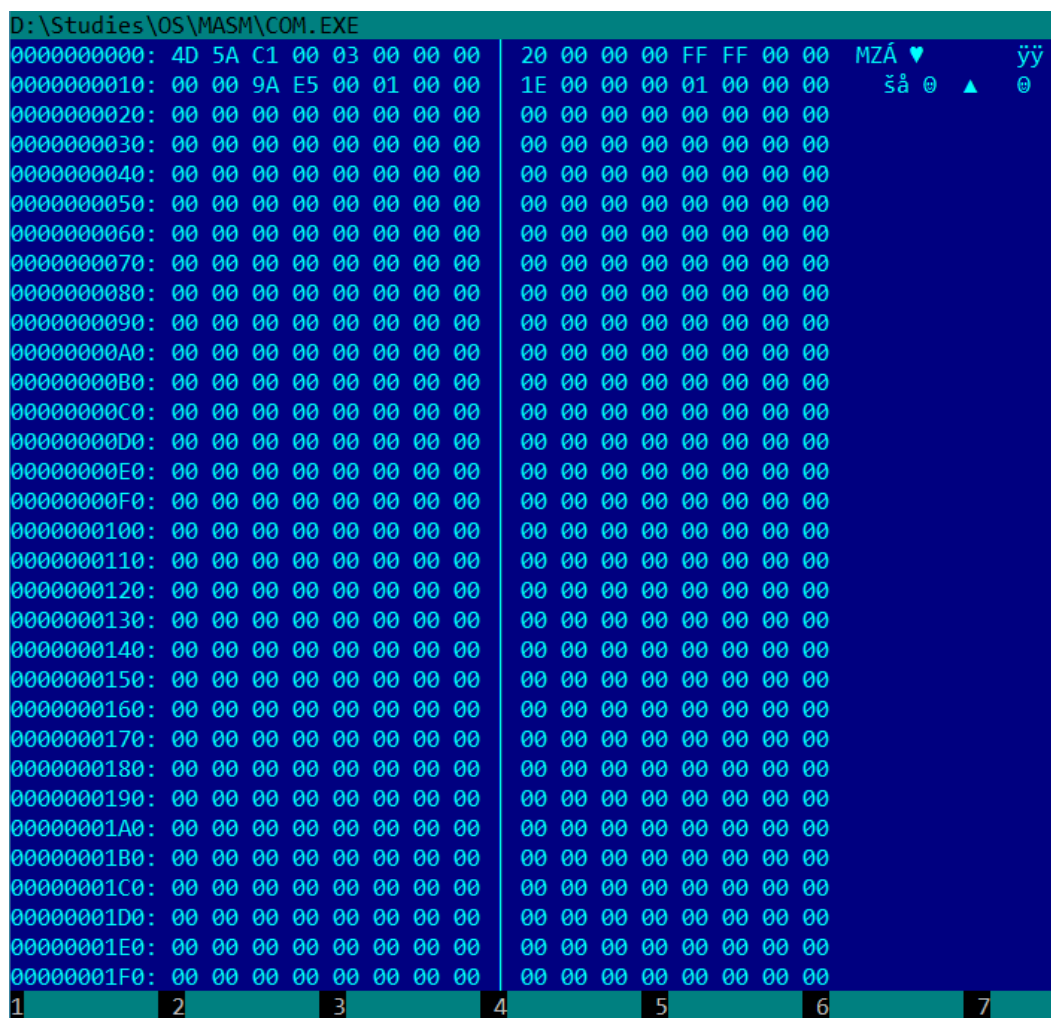


Рисунок 5 – Содержание «плохого» .EXE файла в FAR

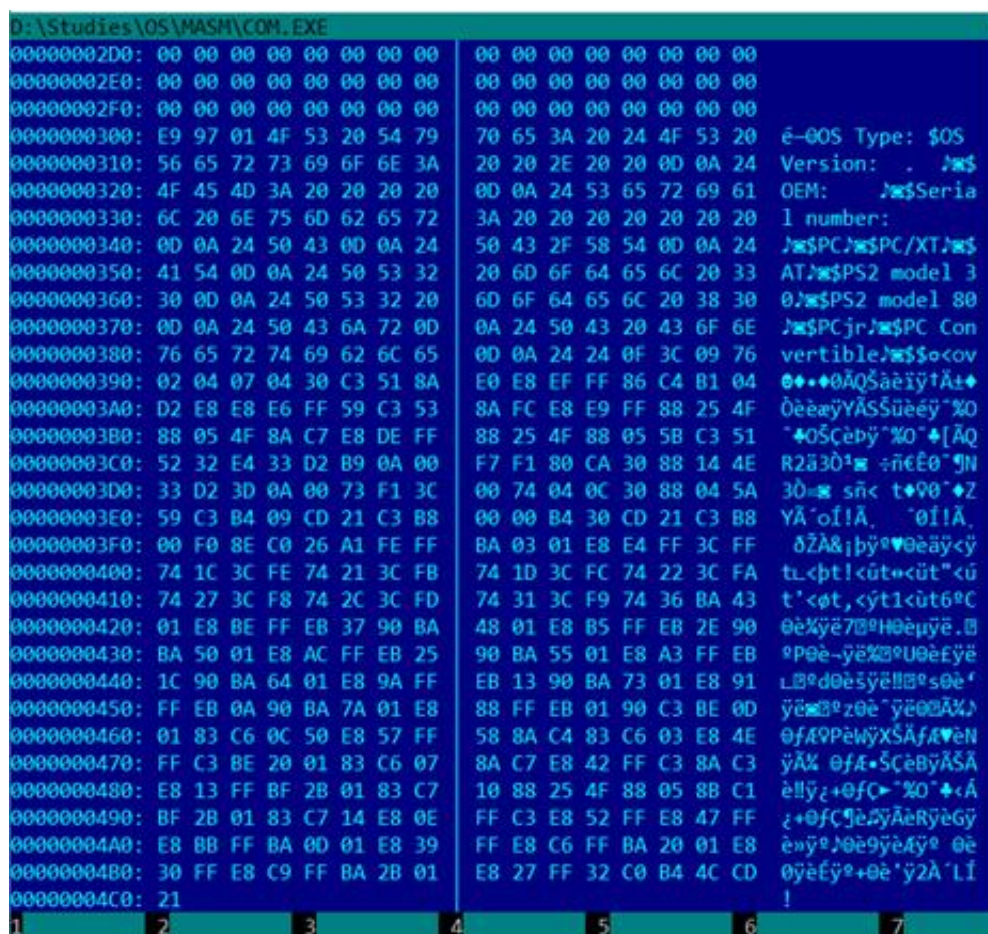


Рисунок 6 – Содержание «плохого» .EXE файла в FAR

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 представлена на рис. 7—8.



D:\Studies\OS\MASM\GOOD_EXE.EXE																		
000000000:	4D	5A	1E	00	03	00	1A	00	20	00	00	00	FF	FF	00	00	MZ▲♥→	ÿÿ
000000010:	20	00	A3	D5	31	01	0B	00	1E	00	00	00	01	00	11	00	£Ö10♠▲	0◀
000000020:	0B	00	1C	00	0B	00	26	00	0B	00	33	00	0B	00	7B	00	♠L♠&♠3♠{	
000000030:	0B	00	A3	00	0B	00	AE	00	0B	00	B9	00	0B	00	C4	00	♠£♠°♠¹♠Ä	
000000040:	0B	00	CF	00	0B	00	DA	00	0B	00	E5	00	0B	00	F5	00	♠İ♠Ú♠â♠õ	
000000050:	0B	00	00	01	0B	00	0E	01	0B	00	16	01	0B	00	2E	01	♠0♠♠0♠=0♠.0	
000000060:	0B	00	32	01	0B	00	39	01	0B	00	3E	01	0B	00	43	01	♠20♠90♠>0♠C0	
000000070:	0B	00	4B	01	0B	00	50	01	0B	00	58	01	0B	00	5D	01	♠K0♠P0♠X0♠]0	
000000080:	0B	00	65	01	0B	00	00	00	00	00	00	00	00	00	00	00	♠e0♠	
000000090:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000000F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000100:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000110:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000120:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000130:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000140:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000150:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000160:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000170:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000180:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000190:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000001A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000001B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000001C0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000001D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000001E0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000001F0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
1	2	3	4	5	6	7												

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

D:\Studies\OS\MASM\GOOD_EXE.EXE																	
0000000220:	4F	53	20	54	79	70	65	3A	20	24	4F	53	20	56	65	72	OS Type: \$OS Ver
0000000230:	73	69	6F	6E	3A	20	20	2E	20	20	0D	0A	24	4F	45	4D	sion: . ♠\$OEM
0000000240:	3A	20	20	20	20	0D	0A	24	53	65	72	69	61	6C	20	6E	: ♠\$Serial n
0000000250:	75	6D	62	65	72	3A	20	20	20	20	20	20	0D	0A	24		umber: ♠\$
0000000260:	50	43	0D	0A	24	50	43	2F	58	54	0D	0A	24	41	54	0D	PC)♠PC/XT)♠\$AT)
0000000270:	0A	24	50	53	32	20	6D	6F	64	65	6C	20	33	30	0D	0A	♠\$PS2 model 30)♠
0000000280:	24	50	53	32	20	6D	6F	64	65	6C	20	38	30	0D	0A	24	\$PS2 model 80)♠\$
0000000290:	50	43	6A	72	0D	0A	24	50	43	20	43	6F	6E	76	65	72	PCjr)♠\$PC Conver
00000002A0:	74	69	62	6C	65	0D	0A	24	00	00	00	00	00	00	00	00	tible)♠\$
00000002B0:	24	0F	3C	09	76	02	04	07	04	30	CB	51	8A	E0	9A	00	\$o<ov0♦♦0EQ\$ās
00000002C0:	00	0B	00	86	C4	B1	04	D2	E8	9A	00	00	0B	00	59	CB	♠†Ä±♦Öèš♠YĖ
00000002D0:	53	8A	FC	9A	0B	00	0B	00	88	25	4F	88	05	4F	8A	C7	SŠüš♠♠%0^+OŠC
00000002E0:	9A	0B	00	0B	00	88	25	4F	88	05	5B	CB	51	52	32	E4	š♠♠%0^+ EQR2ā
00000002F0:	33	D2	B9	0A	00	F7	F1	80	CA	30	88	14	4E	33	D2	3D	30^ā ÷ñĖĖ0^JN3D=
0000000300:	0A	00	73	F1	3C	00	74	04	0C	30	88	04	5A	59	CB	B4	ā sñ< t♦90^♦ZYĖ^
0000000310:	09	CD	21	CB	B8	00	00	B4	30	CD	21	CB	B8	00	F0	8E	oÍ!Ė. ^oÍ!Ė. āŽ
0000000320:	C0	26	A1	FE	FF	BA	00	00	9A	5F	00	0B	00	3C	FF	74	Å&ıbyē ſ_♠<yŧ
0000000330:	1C	3C	FE	74	23	3C	FB	74	1F	3C	FC	74	26	3C	FA	74	L<ıbt#<yŧ<ıŧt&<ıŧt
0000000340:	2D	3C	F8	74	34	3C	FD	74	3B	3C	F9	74	42	BA	40	00	-<ıpt4<yŧ;<ıŧtBıı
0000000350:	9A	5F	00	0B	00	EB	43	90	BA	45	00	9A	5F	00	0B	00	ſ_♠ĖCııĖ ſ_♠
0000000360:	EB	38	90	BA	4D	00	9A	5F	00	0B	00	EB	2D	90	BA	52	Ė8ıııM ſ_♠Ė-ıııR
0000000370:	00	9A	5F	00	0B	00	EB	22	90	BA	61	00	9A	5F	00	0B	ſ_♠Ė"ıııa ſ_♠
0000000380:	00	EB	17	90	BA	70	00	9A	5F	00	0B	00	EB	0C	90	BA	Ėııııp ſ_♠Ėıııı
0000000390:	77	00	9A	5F	00	0B	00	EB	01	90	CB	BE	0A	00	83	C6	w ſ_♠Ėııııııı fĖ
00000003A0:	0C	50	9A	3C	00	0B	00	58	8A	C4	83	C6	03	9A	3C	00	9Pš<♠XSĖfĖıı<
00000003B0:	0B	00	CB	BE	1D	00	83	C6	07	8A	C7	9A	3C	00	0B	00	♠Ė%+ fĖ•ŠCš<♠
00000003C0:	CB	8A	C3	9A	0B	00	0B	00	BF	28	00	83	C7	10	88	25	ĖŠĖš♠♠ı( fCıı%~
00000003D0:	4F	88	05	8B	C1	BF	28	00	83	C7	14	9A	20	00	0B	00	0^+<Ėı( fCııš♠
00000003E0:	CB	B8	02	00	8E	D8	9A	6C	00	0B	00	9A	64	00	0B	00	Ė.0 ŽÖš1♠šd♠
00000003F0:	9A	EB	00	0B	00	BA	0A	00	9A	5F	00	0B	00	9A	03	01	šĖ♠ıı ſ_♠šıı
0000000400:	0B	00	BA	1D	00	9A	5F	00	0B	00	9A	11	01	0B	00	BA	♠ıı ſ_♠šıııııııı
0000000410:	28	00	9A	5F	00	0B	00	32	C0	B4	4C	CD	21	CB			( ſ_♠2Ė^Lıı!Ė
1	2	3	4	5	6	7											

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.

The screenshot shows a debugger window with the following content:

[CPU 80486]			1=[↑][↓]	
cs:0100	E99701	jmp 029A ↓	ax	0000 c=0
cs:0103	4F	dec di	bx	0000 z=0
cs:0104	53	push bx	cx	0000 s=0
cs:0105	205479	and [si+79],dl	dx	0000 o=0
cs:0108	7065	jo 016F	si	0000 p=0
cs:010A	3A20	cmp ah,[bx+si]	di	0000 a=0
cs:010C	244F	and al,4F	bp	0000 i=1
cs:010E	53	push bx	sp	FFFE d=0
cs:010F	205665	and [bp+65],dl	ds	48DD
cs:0112	7273	jnb 0187	es	48DD
cs:0114	696F6E3A20	imul bp,[bx+6E],20	ss	48DD
cs:0119	202E2020	and [2020],ch	cs	48DD
cs:011D	0D0A24	or ax,240A	ip	0100

ds:0000	CD 20 FF 9F 00 EA FF FF	= f 0
ds:0008	AD DE E4 01 C9 15 AE 01	i 20 0 0
ds:0010	C9 15 80 02 24 10 92 01	0 0 0 0 0 0 0 0
ds:0018	01 01 01 00 02 FF FF FF	0 0 0 0 0 0 0 0

ss:0000	20CD
ss:FFFE	0000

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-FFFFh.

### Загрузка «хорошего» EXE модуля в основную память:

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

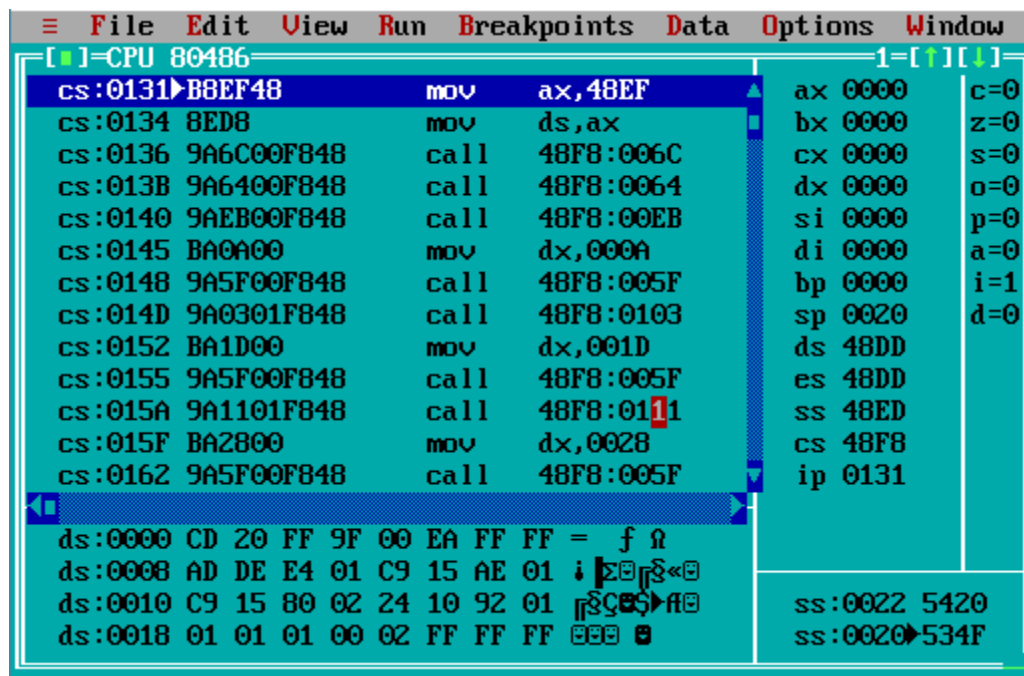


Рисунок 10 – Результат загрузки «хорошего» .EXE в основную память

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 выполняемого 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_1
        or AL,30h
        mov [SI],AL
end_1: 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_1
        or AL,30h
        mov [SI],AL
end_1:  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

```