

**МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ**

**ОТЧЕТ
по практической работе №1
по дисциплине «Операционные системы»
Тема: Исследование структур загрузочного модуля**

Студентка гр. 8382

Наконечная А.Ю.

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.

Постановка задачи.

Необходимо написать текст исходного .COM модуля, который определяет тип РС и версию системы. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип РС и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводится в символьную строку, содержащую запись шестнадцатеричного числа и выводится 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. 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 исходного .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить и отладить. Таким образом, будет получен текст «хорошего» .EXE модуля. Затем необходимо сравнить тексты для .COM и .EXE модулей.

Выполнение работы.

Была создана функция TYPE_PC для определения типа IBM PC. В Таблице 2 указаны соответствия кода и типа PC. Данная информация была использована для написания функции.

Таблица 2 — соответствие кода и типа PC

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 типа MS DOS была написана функция MS_DOS. Она позволяет узнать: номер основной версии и модификации, серийный номер OEM, серийный номер пользователя.

В результате выполнения была получена информация, представленная на рисунках 1-3.

```
C:\>lr1_com.com
Type AT
Version 5.0
OEM 0
User 000000
C:\>
```

Рисунок 1 — «хороший» COM модуль

```
C:\>lr1_exe.exe
Type AT
Version 5.0
OEM 0
User 000000
C:\>
```

Рисунок 2 — «хороший» EXE модуль

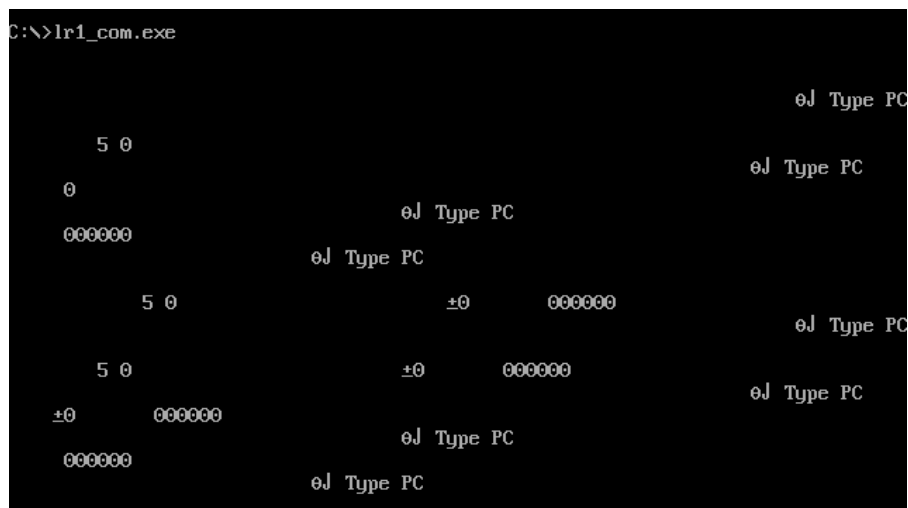


Рисунок 3 — «плохой» EXE модуль

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM-программа?

COM-программа должна содержать один сегмент. Весь код и данные располагаются в одном сегменте.

2) EXE-программа?

EXE-программы должны содержать не менее одного сегмента.

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

Обязательно должна присутствовать директива `ORG 100h`. Если опущен `ORG 100h`, то на данные в префиксе программного сегмента могут быть установлены неправильные ссылки с непредсказуемым результатом при выполнении.

Также необходима директива `ASSUME`, чтобы сегмент данных и сегмент кода указывали на один общий сегмент. (`ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING`)

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

Нет. Нельзя использовать команды, в которых участвует адрес сегмента, т. к. в COM файле отсутствует таблица настройки адресов и присутствует только один сегмент.

Отличия форматов файлов .COM и .EXE программ.

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

COM файл состоит из одного сегмента и занимает менее 64 Кб. COM-программа генерирует стек автоматически.

На рисунке 4 можно увидеть, что код располагается с адреса 0h.

00000000	e9 f5 00 54 79 70 65 20 50 43 0d 0a 24 54 79 70	...Type PC..\$Typ
00000010	65 20 50 43 2f 58 54 0d 0a 24 54 79 70 65 20 41	e PC/XT..\$Type A
00000020	54 0d 0a 24 54 79 70 65 20 50 53 32 20 6d 6f 64	T..\$Type PS2 mod
00000030	65 6c 20 33 30 0d 0a 24 54 79 70 65 20 50 53 32	el 30..\$Type PS2
00000040	20 6d 6f 64 65 6c 20 38 30 0d 0a 24 54 79 70 65	model 80..\$Type
00000050	20 50 43 6a 72 0d 0a 24 54 79 70 65 20 50 43 20	PCjr..\$Type PC
00000060	43 6f 6e 76 65 72 74 69 62 6c 65 0d 0a 24 56 65	Convertible..\$Ve
00000070	72 73 69 6f 6e 20 20 2e 20 20 0d 0a 24 56 65 72	rsion . ..\$Ver
00000080	73 69 6f 6e 20 3c 20 32 2e 30 0d 0a 24 4f 45 4d	sion < 2.0..\$OEM
00000090	20 20 0d 0a 24 55 73 65 72 20 20 20 20 20 20 20	..\$User
000000a0	24 24 0f 3c 09 76 02 04 07 04 30 c3 51 8a e0 e8	\$.<.v....0.Q...
000000b0	ef ff 86 c4 b1 04 d2 e8 e8 e6 ff 59 c3 53 8a fcY.S..
000000c0	e8 e9 ff 88 25 4f 88 05 4f 8a c7 e8 de ff 88 25%0..0.....%
000000d0	4f 88 05 5b c3 51 52 32 e4 33 d2 b9 0a 00 f7 f1	0..[.QR2.3.....
000000e0	80 ca 30 88 14 4e 33 d2 3d 0a 00 73 f1 3c 00 74	..0..N3.=...s.<.t
000000f0	04 0c 30 88 04 5a 59 c3 b8 00 f0 8e c0 26 a0 fe	..0..ZY.....&..
00000100	ff 3c ff 74 1c 3c fe 74 1e 3c fb 74 1a 3c fc 74	.<.t.<.t.<.t.<.t
00000110	1c 3c fa 74 1e 3c f8 74 20 3c fd 74 22 3c f9 74	.<.t.<.t <.t"<.t

Рисунок 4 — файл COM в шестнадцатеричном виде

00000120	24 ba 03 01 eb 25 90 ba 0d 01 eb 1f 90 ba 1a 01	\$....%......
00000130	eb 19 90 ba 24 01 eb 13 90 ba 38 01 eb 0d 90 ba\$.8....
00000140	4c 01 eb 07 90 ba 58 01 eb 01 90 b4 09 cd 21 e8	L....X.....!.
00000150	01 00 c3 b4 30 cd 21 50 3c 00 74 1c be 6e 01 830.!P<.t..n..
00000160	c6 08 e8 70 ff 58 8a c4 83 c6 03 e8 67 ff ba 6e	...p.X.....g..n
00000170	01 b4 09 cd 21 eb 0c 90 ba 7d 01 b4 09 cd 21 58!....}.!X
00000180	eb 01 90 be 8d 01 83 c6 05 8a c7 e8 47 ff ba 8dG...
00000190	01 b4 09 cd 21 eb 01 90 bf 95 01 83 c7 0a 8b c1!.....
000001a0	e8 1a ff 8a c3 e8 04 ff 83 ef 02 89 05 ba 95 01
000001b0	b4 09 cd 21 c3 e8 40 ff 32 c0 b4 4c cd 21	...!..@.2..L.!

Рисунок 5 — продолжение файла COM в шестнадцатеричном виде

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

В «плохом» EXE данные и код располагаются в одном сегменте, что приводит к некорректной работе. К тому же, в EXE-программе должен быть определён сегмент стека.

Код располагается с адреса 300h, что можно заметить на рисунке 7. Можно увидеть на рисунке 6, что с адреса 0h располагается заголовок файла.

00000000	4d 5a be 00 03 00 00 00 20 00 00 00 ff ff 00 00	MZ.....
00000010	00 00 00 00 00 01 00 00 3e 00 00 00 01 00 fb 50>.....P
00000020	6a 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00	jr.....
00000030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Рисунок 6 — файл «плохого» EXE в шестнадцатеричном виде

00000280	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000290	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000300	e9 f5 00 54 79 70 65 20 50 43 0d 0a 24 54 79 70	...Type PC...\$Typ
00000310	65 20 50 43 2f 58 54 0d 0a 24 54 79 70 65 20 41	e PC/XT...\$Type A

Рисунок 7 — часть файла «плохого» EXE в шестнадцатеричном виде с началом кода

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

В отличие от «плохого» EXE в «хорошем» EXE присутствуют три сегмента: сегмент кода, сегмент данных и сегмент стека, в «плохом» EXE данные и код расположены в одном сегменте.

Также в «плохом» EXE адресация кода начинается с 300h, в то время как в «хорошем» EXE адресация начинается с 200h + размер стека. В «хорошем» EXE с адреса 0h располагается заголовок и далее таблица настройки адресов. На рисунках 8 – 10 представлен «хороший» EXE в шестнадцатеричном виде.

00000000	4d 5a 53 01 03 00 01 00 20 00 00 00 ff ff 00 00 90 01 00 00 14 01 23 00 3e 00 00 00 01 00 fb	MZS.....#.>.....
0000001f	50 6a 72 00	Pjr.....
0000003e	16 01 23 00	..#.....
0000005d	00 00
0000007c	00 00
0000009b	00 00
000000ba	00 00
000000d9	00 00
000000f8	00 00
00000117	00 00
00000136	00 00
00000155	00 00
00000174	00 00
00000193	00 00
000001b2	00 00
000001d1	00 00
000001f0	00 00
0000020f	00 00
0000022e	00 00

Рисунок 8 — начало «хорошего» EXE в шестнадцатеричном виде

0000024d	00 00
0000026c	00 00
0000028b	00 00
000002aa	00 00
000002c9	00 00
000002e8	00 00
00000307	00 00
00000326	00 00
00000345	00 00
00000364	00 00
00000383	00 00 00 00 00 00 00 00 00 00 00 00 00 54 79 70 65 20 50 43 0d 0a 24 54 79 70 65 20 50 43 2fType PC..\$Type PC/
000003a2	58 54 0d 0a 24 54 79 70 65 20 41 54 0d 0a 24 54 79 70 65 20 50 53 32 20 6d 6f 64 65 6c 20 33	XT..\$Type AT..\$Type PS2 model 3
000003c1	30 0d 0a 24 54 79 70 65 20 50 53 32 20 6d 6f 64 65 6c 20 38 30 0d 0a 24 54 79 70 65 20 50 43	0..\$Type PS2 model 80..\$Type PC
000003e0	6a 72 0d 0a 24 54 79 70 65 20 50 43 20 43 6f 6e 76 65 72 74 69 62 6c 65 0d 0a 24 56 65 72 73	jrr..\$Type PC Convertible..\$Vers
000003ff	69 6f 6e 20 20 2e 20 20 0d 0a 24 56 65 72 73 69 6f 6e 20 3c 20 32 2e 30 0d 0a 24 4f 45 4d 20	ion . ..\$Version < 2.0..\$OEM
0000041e	20 0d 0a 24 55 73 65 72 20 20 20 20 20 20 20 20 24 0f 3c 09 76 02 04 07 04 30 c3 51 8a	..\$User \$..\$.<.v....0.Q.
0000043d	e0 e8 ef ff 86 c4 b1 04 d2 e8 e8 e6 ff 59 c3 53 8a fc e8 e9 ff 88 25 4f 88 05 4f 8a c7 e8 deY.S.....%0..0....
0000045c	ff 88 25 4f 88 05 5b c3 51 52 32 e4 33 d2 b9 0a 00 f7 f1 80 ca 30 88 14 4e 33 d2 3d 0a 00 73	..%0..[.QR2.3.....0..N3.=..s
0000047b	f1 3c 00 74 04 0c 30 88 04 5a 59 c3 b8 00 f0 8e c0 26 a0 fe ff 3c ff 74 1c 3c fe 74 1e 3c fb	<.t..0..ZY.....&...<.t.<.t.<.

Рисунок 9 — продолжение «хорошего» EXE в шестнадцатеричном виде

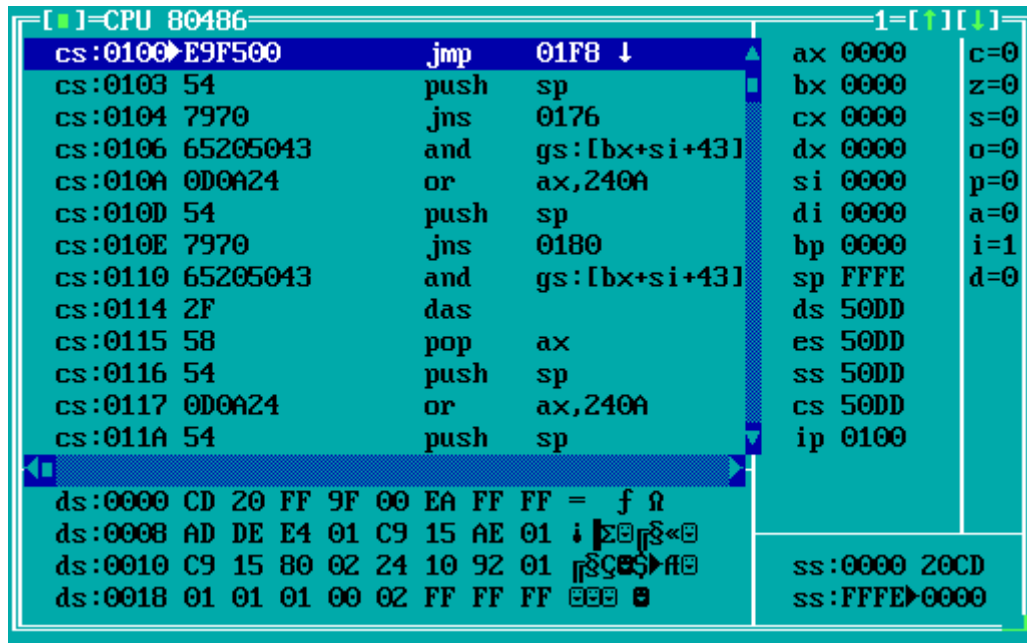
0000049a	74 1a 3c fc 74 1c 3c fa 74 1e 3c f8 74 20 3c fd 74 22 3c f9 74 24 ba 00 00 eb 25 90 ba 0a 00	t.<.t.<.t.<.t <.t"<.t\$.%....
000004b9	eb 1f 90 ba 17 00 eb 19 90 ba 21 00 eb 13 90 ba 35 00 eb 0d 90 ba 49 00 eb 07 90 ba 55 00 eb!.....5.....I.....U..
000004d8	01 90 b4 09 cd 21 e8 01 00 c3 b4 30 cd 21 50 3c 00 74 1c be 6b 00 83 c6 08 e8 70 ff 58 8a c4!.....0..!P<.t..k.....p.X..
000004f7	83 c6 03 e8 67 ff ba 6b 00 b4 09 cd 21 eb 0c 90 ba 7a 00 b4 09 cd 21 58 eb 01 90 be 8a 00 83g..k.....!.....z.....!X.....
00000516	c6 05 8a c7 e8 47 ff ba 8a 00 b4 09 cd 21 eb 01 90 bf 92 00 83 c7 0a 8b c1 e8 1a ff 8a c3 e8G.....!.....
00000535	04 ff 83 ef 02 89 05 ba 92 00 b4 09 cd 21 c3 50 b8 19 00 8e d8 e8 3a ff 32 c0 b4 4c cd 21!..P.....:2..L..!

Рисунок 10 — конец «хорошего» EXE в шестнадцатеричном виде

Загрузка COM модуля в основную память.

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

В начале определяется сегментный адрес участка ОП, у которого достаточно места для загрузки программы. Создаётся блок памяти для PSP и программы. Далее файл типа COM читается в память непосредственно выше PSP по смещению 100h. На рисунке 11 представлен результат работы отладчика для COM файла.



The screenshot shows a debugger window with the following content:

[CPU 80486]				1=[↑][↓]		
cs:0100	E9F500	jmp	01F8 ↓	ax	0000	c=0
cs:0103	54	push	sp	bx	0000	z=0
cs:0104	7970	jns	0176	cx	0000	s=0
cs:0106	65205043	and	gs:[bx+si+43]	dx	0000	o=0
cs:010A	0D0A24	or	ax,240A	si	0000	p=0
cs:010D	54	push	sp	di	0000	a=0
cs:010E	7970	jns	0180	bp	0000	i=1
cs:0110	65205043	and	gs:[bx+si+43]	sp	FFFE	d=0
cs:0114	2F	das		ds	50DD	
cs:0115	58	pop	ax	es	50DD	
cs:0116	54	push	sp	ss	50DD	
cs:0117	0D0A24	or	ax,240A	cs	50DD	
cs:011A	54	push	sp	ip	0100	

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

ss:0000	20CD
ss:FFFE	0000

Рисунок 11 – результат работы отладчика для COM файла

2) Что располагается с адреса 0?

С адреса 0 располагается PSP.

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

Сегментные регистры CS, DS, ES и SS указывают на PSP и имеют значения 50DD.

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

COM-программа генерирует стек автоматически. Регистр SS указывает на 0h, а регистр SP указывает на FFFEh. Адреса стека находятся в диапазоне от 0h до FFFEh.

Загрузка «хорошего» EXE модуля в основную память.

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

В начале определяется сегментный адрес, начиная с которого может быть загружена программа. По смещению 0h формируется PSP, сама программа загружается по смещению 0100h.

DS и ES устанавливаются на начало сегмента PSP, SS – на начало сегмента стека, CS – на начало сегмента команд.

На рисунке 12 представлен результат работы отладчика для «хорошего» EXE файла.

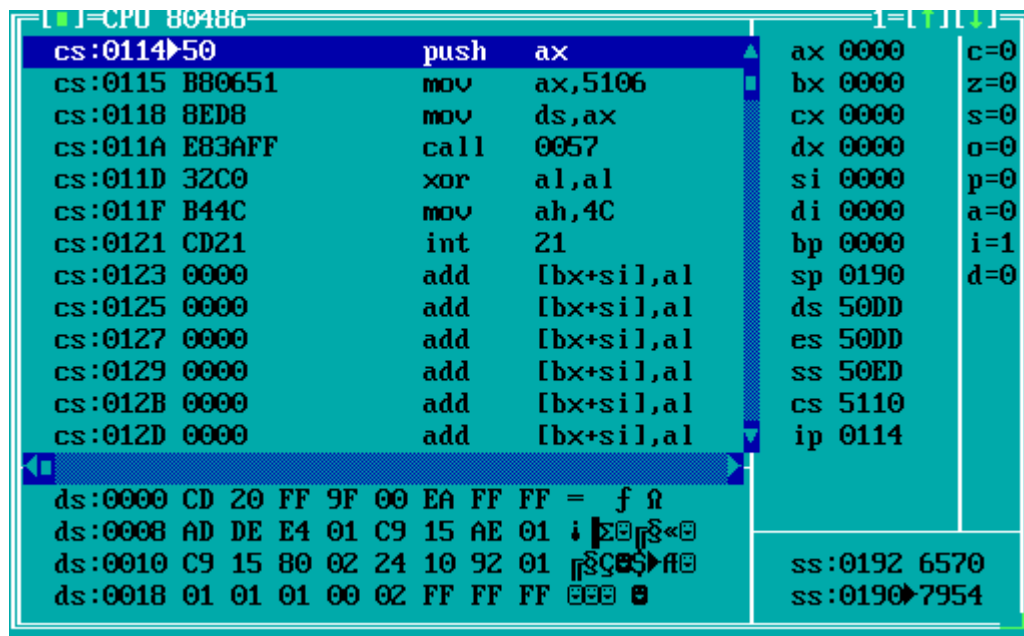


Рисунок 12 – результат работы отладчика для EXE файла

2) На что указывают регистры DS и ES?

Регистры DS и ES указывают на начало PSP.

3) Как определяется стек?

Стек определяется с помощью директивы ASSUME, которая установит регистр SS на начало сегмента стека.

4) Как определяется точка входа?

Точка входа определяется с помощью директивы END.

Выводы.

Были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А

Исходный код программы

Название файла: lr1_com.asm

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

; Данные
PC_STR db 'Type PC', 0DH,0AH,'$'
PC_XT_STR db 'Type PC/XT', 0DH,0AH,'$'
AT_STR db 'Type AT', 0DH,0AH,'$'
PS2_30_STR db 'Type PS2 model 30', 0DH,0AH,'$'
PS2_80_STR db 'Type PS2 model 80', 0DH,0AH,'$'
PCjr_STR db 'Type PCjr', 0DH,0AH,'$'
PC_C_STR db 'Type PC Convertible', 0DH,0AH,'$'
SYS_VER_STR db 'Version . ', 0DH,0AH,'$'
SYS_VER_STR_2 db 'Version < 2.0', 0DH,0AH,'$'
OEM_STR db 'OEM ', 0DH,0AH,'$'
USER_STR db 'User      $'

; 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 10с/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_1
        or AL, 30h
        mov [SI], AL
    end_1:
        pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP
;-----

; Код
BEGIN:

```

```

TYPE_PC PROC near
; Определение типа PC
    mov AX,0F000H
    mov ES,AX
    mov AL,ES:[0FFFEH]
    cmp AL,0FFH      ;это PC?
        je PC
    cmp AL,0FEH ;это PC/XT?
        je PC_XT
    cmp AL,0FBH ;это PC/XT?
        je PC_XT
    cmp AL,0FCH ;это AT?
        je AT
    cmp AL,0FAH ;это PS2 модель 30?
        je PS2_30
    cmp AL,0F8H ;это PS2 модель 80?
        je PS2_80
    cmp AL,0FDH ;это PCjr?
        je PCjr
    cmp AL,0F9H ;это PC Convertible?
        je PC_C

PC:
    mov DX,offset PC_STR
    jmp WRITE
PC_XT:
    mov DX,offset PC_XT_STR
    jmp WRITE
AT:
    mov DX,offset AT_STR
    jmp WRITE
PS2_30:
    mov DX,offset PS2_30_STR
    jmp WRITE
PS2_80:
    mov DX,offset PS2_80_STR
    jmp WRITE
PCjr:
    mov DX,offset PCjr_STR
    jmp WRITE
PC_C:
    mov DX,offset PC_C_STR
    jmp WRITE
WRITE:
    mov AH,09h
    int 21h
    call MS_DOS
ret

```

TYPE_PC ENDP

MS_DOS PROC near

; Определение версии MS_DOS

VER:

```
    mov AH,30h
    int 21h
    push AX
    cmp AL, 0
        je VER_2
    mov SI,offset SYS_VER_STR
    add SI,8
    call BYTE_TO_DEC
    pop AX
    mov AL,AH
    add SI,3
    call BYTE_TO_DEC
    mov DX,offset SYS_VER_STR
    mov AH,09h
    int 21h
    jmp OEM
```

VER_2:

```
    mov DX,offset SYS_VER_STR_2
    mov AH,09h
    int 21h
    pop AX
    jmp OEM
```

OEM:

```
    mov SI,offset OEM_STR
    add SI,5
    mov AL,BH
    call BYTE_TO_DEC
    mov DX,offset OEM_STR
    mov AH,09h
    int 21h
    jmp USER
```

USER:

```
    mov DI,offset USER_STR
    add DI,10
    mov AX,CX
    call WRD_TO_HEX
    mov AL,BL
    call BYTE_TO_HEX
    sub DI,2
    mov [DI],AX
    mov DX,offset USER_STR
```

```
        mov AH,09h
        int 21h
        ret
MS_DOS ENDP

call TYPE_PC

; Выход в DOS
xor AL,AL
mov AH,4Ch
int 21H

TESTPC ENDS
END START
```


ПРИЛОЖЕНИЕ В

Исходный код программы

Название файла: lr1_exe.asm

```
AStack      SEGMENT  STACK
              DW 200 DUP(?)
AStack      ENDS
```

DATA SEGMENT

; Данные

```
PC_STR db 'Type PC', 0DH,0AH,'$'
PC_XT_STR db 'Type PC/XT', 0DH,0AH,'$'
AT_STR db 'Type AT', 0DH,0AH,'$'
PS2_30_STR db 'Type PS2 model 30', 0DH,0AH,'$'
PS2_80_STR db 'Type PS2 model 80', 0DH,0AH,'$'
PCjr_STR db 'Type PCjr', 0DH,0AH,'$'
PC_C_STR db 'Type PC Convertible', 0DH,0AH,'$'
SYS_VER_STR db 'Version . ', 0DH,0AH,'$'
SYS_VER_STR_2 db 'Version < 2.0', 0DH,0AH,'$'
OEM_STR db 'OEM ', 0DH,0AH,'$'
USER_STR db 'User      $'
DATA ENDS
```

CODE SEGMENT

ASSUME CS:CODE,DS:DATA,SS:AStack

; 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 10с/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
;-----

```

```

; Код
BEGIN:

TYPE_PC PROC near
; Определение типа PC
    mov AX,0F000H
    mov ES,AX
    mov AL,ES:[0FFF0H]
    cmp AL,0FFH      ;это PC?
        je PC
    cmp AL,0FEH ;это PC/XT?
        je PC_XT
    cmp AL,0FBH ;это PC/XT?
        je PC_XT
    cmp AL,0FCH ;это AT?
        je AT
    cmp AL,0FAH ;это PS2 модель 30?
        je PS2_30
    cmp AL,0F8H ;это PS2 модель 80?
        je PS2_80
    cmp AL,0FDH ;это PCjr?
        je PCjr
    cmp AL,0F9H ;это PC Convertible?
        je PC_C

PC:
    mov DX,offset PC_STR
    jmp WRITE
PC_XT:
    mov DX,offset PC_XT_STR
    jmp WRITE
AT:
    mov DX,offset AT_STR
    jmp WRITE
PS2_30:
    mov DX,offset PS2_30_STR
    jmp WRITE
PS2_80:
    mov DX,offset PS2_80_STR
    jmp WRITE
PCjr:
    mov DX,offset PCjr_STR
    jmp WRITE
PC_C:
    mov DX,offset PC_C_STR
    jmp WRITE
WRITE:
    mov AH,09h

```

```

        int 21h
        call MS_DOS
    ret
TYPE_PC ENDP

MS_DOS PROC near
; Определение версии MS_DOS
VER:
    mov AH,30h
    int 21h
    push AX
    cmp AL, 0
        je VER_2
    mov SI,offset SYS_VER_STR
    add SI,8
    call BYTE_TO_DEC
    pop AX
    mov AL,AH
    add SI,3
    call BYTE_TO_DEC
    mov DX,offset SYS_VER_STR
    mov AH,09h
    int 21h
    jmp OEM

VER_2:
    mov DX,offset SYS_VER_STR_2
    mov AH,09h
    int 21h
    pop AX
    jmp OEM

OEM:
    mov SI,offset OEM_STR
    add SI,5
    mov AL,BH
    call BYTE_TO_DEC
    mov DX,offset OEM_STR
    mov AH,09h
    int 21h
    jmp USER

USER:
    mov DI,offset USER_STR
    add DI,10
    mov AX,CX
    call WRD_TO_HEX
    mov AL,BL
    call BYTE_TO_HEX

```

```

        sub DI,2
        mov [DI],AX
        mov DX,offset USER_STR
        mov AH,09h
        int 21h
        ret
MS_DOS ENDP

MAIN PROC far
    push AX
    mov AX,DATA
    mov DS,AX
    call TYPE_PC

    ; Выход в DOS
    xor AL,AL
    mov AH,4Ch
    int 21H
MAIN ENDP
CODE ENDS
END Main

```