

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. 0382

Сергеев Д.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

Задание.

Шаг 1. Написать текст исходно .COM модуля, который определяет тип РС и версию системы. Построить «плохой» .EXE модуль, полученный из исходного текста для .COM модуля.

Шаг 2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в Шаге 1, построить и отладить его. Таким образом будет получен «хороший» .EXE.

Шаг 3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на контрольные вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 4. Запустить FAR и открыть файлы загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем открыть файл загрузочного модуля «хорошего» .EXE и сравнить его с предыдущими файлами. Ответить на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».

Шаг 5. Открыть отладчик TD.EXE и загрузить .COM. Ответить на контрольные вопросы «Загрузка COM модуля в основную память». Представить в отчете план загрузки модуля .COM в основную память.

Шаг 6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE модуля в основную память».

Выполнение работы.

В ходе работы был взят шаблон из методических материалов, также были написаны две процедуры: CHECK_PC_TYPE и CHECK_OS_VERSIONS.

В процедуре CHECK_PC_TYPE проверялся предпоследний байт ROM BIOS, в котором хранится информация о типе РС. И в зависимости от его

значения, на экран выводится информация и типе PC с помощью процедуры WRITESTRING.

В процедуре CHECK_OS_VERS с помощью функции 30h прерывания 21h мы получаем информация о версии MS DOS, а затем с помощью процедур BYTE_TO_DEC, WRD_TO_HEX и BYTE_TO_HEX в заготовленные строчки вносится актуальная информация, и затем они выводятся на терминал.

В ходе работы было написано два файла progcom.asm(“Хороший” .com модуль и “плохой” .exe модуль(полученный использованием .com шаблона)) и progexe.asm(“Хороший” .exe модуль). Ниже представлены результаты запусков “хорошего” и “плохого” .exe модуля и “хорошего” .com модуля.

```
C:\>progcom.com
Your PC type is -> AT
Version MS-DOS: 5.0
Serial number OEM: 255
User serial number: 000000H
C:\>
```

Рисунок 1 – “Хороший” .com модуль

```
C:\>progcom.exe
                                00000000
s -> PC/XT
pe is -> AT
pe is -> PS2  00000000
C:\>
```

Рисунок 2 – “Плохой” .exe модуль

Для того, чтобы получить “хороший” .exe модуль следует выделить сегменты стека, кода и данных, убрать ненужные директивы и установить сегмент данных.

```
C:\>progexe.exe
Your PC type is -> AT
Version MS-DOS: 5.0
Serial number OEM: 255
User serial number: 000000H
C:\>
```

Рисунок 3 – “Хороший” .exe модуль

Исходный программный код смотрите в приложении А.

Контрольные вопросы.

1. Отличие исходных текстов COM и EXE программ

- Сколько сегментов должна содержать COM-программа?

Ответ: один

- EXE?

Ответ: один и больше(CS=DS, ES,SS)

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

Ответ: директива `org 100h`, которая смещает адресацию на размер PSP, а именно на 256 байтов, а также `ASSUME`, т.к. без этой директивы компилятор не будет знать какой сегмент относится к какому сегментному регистру.

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

Ответ: нет, не все, в число запрещенных входят команды, которые непосредственно берут адрес сегмента, так как в COM-файле отсутствует таблица настроек.

2. Отличия форматов файлов COM и EXE модулей.

- Какова структура файла COM? С какого адреса располагается код?

Ответ: COM файлы состоят из одного сегмента, адресация которого начинается с `0h`, но при загрузке программы в память, код сместиться на `100h`, и будет начинаться сразу после PSP (рисунок 4).

- Какова структура файла “плохого” EXE? С какого адреса располагается код? Что располагается с адреса `0`?

Ответ: “Плохой” EXE файл состоит из одного сегмента. Код начинается с адреса `300h`, а до этого располагается заголовок с MZ байтами, relocation table и смещение `100h`, заданное в коде (рисунок 5).

- Какова структура файла “хорошего” EXE? Чем он отличается от файла “плохого” EXE?

Ответ: В “хорошем” EXE модуле файл начинается с заголовка и relocation table, далее идёт программа, разбитая на сегменты, сначала

сегмент стека, сегмент данных и сегмент кода, в отличие от “плохого” EXE, где код и данные располагаются в одном сегменте. Кроме этого в отличие от “плохого” EXE отсутствует ненужное смещение на 256 байт (рисунок 6).

C:\Users\pyuko\Desktop\ЛАБЫ ОС\PROGCOM.COM																h 1252 678	
0000000000:	E9	97	02	59	6F	75	72	20	50	43	20	74	79	70	65	20	é-0Your PC type
0000000010:	69	73	20	2D	3E	20	50	43	0D	0A	24	59	6F	75	72	20	is -> PC)0\$Your
0000000020:	50	43	20	74	79	70	65	20	69	73	20	2D	3E	20	50	43	PC type is -> PC
0000000030:	2F	58	54	0D	0A	24	59	6F	75	72	20	50	43	20	74	79	/XT)0\$Your PC ty
0000000040:	70	65	20	69	73	20	2D	3E	20	41	54	0D	0A	24	59	6F	pe is -> AT)0\$Yo
0000000050:	75	72	20	50	43	20	74	79	70	65	20	69	73	20	2D	3E	ur PC type is ->
0000000060:	20	50	53	32	20	D0	BC	D0	BE	D0	B4	D0	B5	D0	BB	D1	PS2 0%D%D'Dµ0»N
0000000070:	8C	20	33	30	0D	0A	24	59	6F	75	72	20	50	43	20	74	E 30)0\$Your PC t
0000000080:	79	70	65	20	69	73	20	2D	3E	20	50	53	32	20	D0	BC	ype is -> PS2 0%
0000000090:	D0	BE	D0	B4	D0	B5	D0	BB	D1	8C	20	35	30	20	D0	B8	0%D'Dµ0»N'E 50 D,
00000000A0:	D0	BB	D0	B8	20	36	30	0D	0A	24	59	6F	75	72	20	50	0»D, 60)0\$Your P
00000000B0:	43	20	74	79	70	65	20	69	73	20	2D	3E	20	50	53	32	C type is -> PS2
00000000C0:	20	D0	BC	D0	BE	D0	B4	D0	B5	D0	BB	D1	8C	20	38	30	0%D%D'Dµ0»N'E 80
00000000D0:	0D	0A	24	59	6F	75	72	20	50	43	20	74	79	70	65	20)0\$Your PC type
00000000E0:	69	73	20	2D	3E	20	50	D0	A1	6A	72	0D	0A	24	59	6F	is -> PDijr)0\$Yo
00000000F0:	75	72	20	50	43	20	74	79	70	65	20	69	73	20	2D	3E	ur PC type is ->
0000000100:	20	50	43	20	43	6F	6E	76	65	72	74	69	62	6C	65	0D	PC Convertible)
0000000110:	0A	24	59	6F	75	72	20	50	43	20	74	79	70	65	20	69	0\$Your PC type i
0000000120:	73	20	2D	3E	20	20	0D	0A	24	56	65	72	73	69	6F	6E	s ->)0\$Version
0000000130:	20	4D	53	2D	44	4F	53	3A	20	20	2E	20	20	0D	0A	24	MS-DOS: .)0\$
0000000140:	53	65	72	69	61	6C	20	6E	75	6D	62	65	72	20	4F	45	Serial number 0E
0000000150:	4D	3A	20	20	20	0D	0A	24	55	73	65	72	20	73	65	72	M:)0\$User ser
0000000160:	69	61	6C	20	6E	75	6D	62	65	72	3A	20	20	20	20	20	ial number:
0000000170:	20	20	48	20	24	24	0F	3C	09	76	02	04	07	04	30	C3	H \$ \$o<ov0♦♦0Ã
0000000180:	51	8A	E0	E8	EF	FF	86	C4	B1	04	D2	E8	E8	E6	FF	59	Q5âëïÿtÃ±♦0èèæÿY
0000000190:	C3	53	8A	FC	E8	E9	FF	88	25	4F	88	05	4F	8A	C7	E8	ÃS5üèëÿ~%0^*0\$Cè
00000001A0:	DE	FF	88	25	4F	88	05	5B	C3	51	52	32	E4	33	D2	B9	bÿ~%0^*[ÃQR2ã30¹
00000001B0:	0A	00	F7	F1	80	CA	30	88	14	4E	33	D2	3D	0A	00	73	0 ÷ñ€f0^¶N30=0 s

Рисунок 4 – “Хороший” СОМ модуль в 16-ричном виде

```
C:\Users\pyuko\Desktop\ЛАБЫ ОС\PROGCOM.EXE h 1252 1446 Col 0 0% 3:4
00000000: 4D 5A A6 01 03 00 00 00 20 00 00 00 FF FF 00 00 MZ!@  yÿ
00000001: 00 00 1E 05 00 01 00 00 1E 00 00 00 01 00 00 00 ▲+ @ ▲ @
00000002: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000003: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000004: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000006: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000007: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000009: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000011: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000012: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000013: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000014: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000015: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000017: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000019: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1Help 2Text 3Quit 4Dump 5 6Edit 7Search 8ANSI 9 10Quit 11Plugins 12Screen
```

Рисунок 5- “Плохой” EXE модуль в 16-ричном виде

```
0000000A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000011: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000012: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000013: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000014: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000015: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000017: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000019: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000021: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000022: 59 6F 75 72 20 50 43 20 74 79 70 65 20 69 73 20 Your PC type is
00000023: 2D 3E 20 50 43 0D 0A 24 59 6F 75 72 20 50 43 20 -> PC$Your PC
00000024: 74 79 70 65 20 69 73 20 2D 3E 20 50 43 2F 58 54 type is -> PC/XT
00000025: 0D 0A 24 59 6F 75 72 20 50 43 20 74 79 70 65 20 $Your PC type
```

Рисунок 6 – “Хороший” EXE модуль в 16-ричном виде

3. Загрузка COM модуля в основную память

- Какой формат загрузки модуля COM? С какого адреса располагается код?

Ответ: В основной памяти находится свободное место, начиная с которого сначала загружается PSP (256 байт), а затем сам код программы. Код располагается сразу после PSP, а именно 48DD:0100

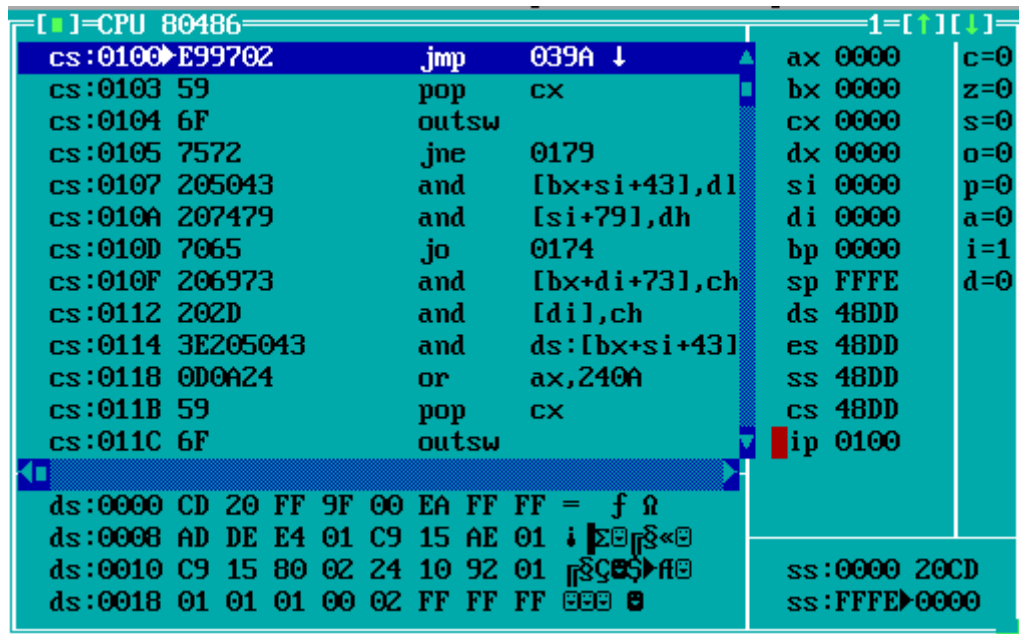


Рисунок 7 – “Хорошая” COM программа в отладчике

- Что располагается с адреса 0?

Ответ: PSP (Program Segment Prefix).

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

Ответ: Все сегментные регистры имеют значение 48DD и указывают на начало PSP.

- Как определяется стек? Какую область памяти он занимает? Какие адреса?

Ответ: Под стек отведён весь сегмент, в который загружена программа, регистр SP указывает на конец стека FFFE, стек может увеличиваться и даже дойти до кода и начать стирать его, это приведет к некорректной работе программы. Таким образом, стек занимает адреса SS:0000h-SS:FFFEh.

4. Загрузка “хорошего” EXE модуля в основную память

- Как загружается “хороший” EXE? Какие значения имеют сегментные регистры?

Ответ: Аналогично COM модулю EXE загружается со смещением относительно PSP – 100h. Значения регистров DS=ES=48DD, CS=4907, SS=48ED.

- На что указывают регистры DS и ES?

Ответ: На начало сегменты PSP.

- Как определяется стек?

Ответ: он определяется вручную с помощью директивы SEGMENT STACK, в которой указывается размер стека.

- Как определяется точка входа?

Ответ: с помощью директивы END.

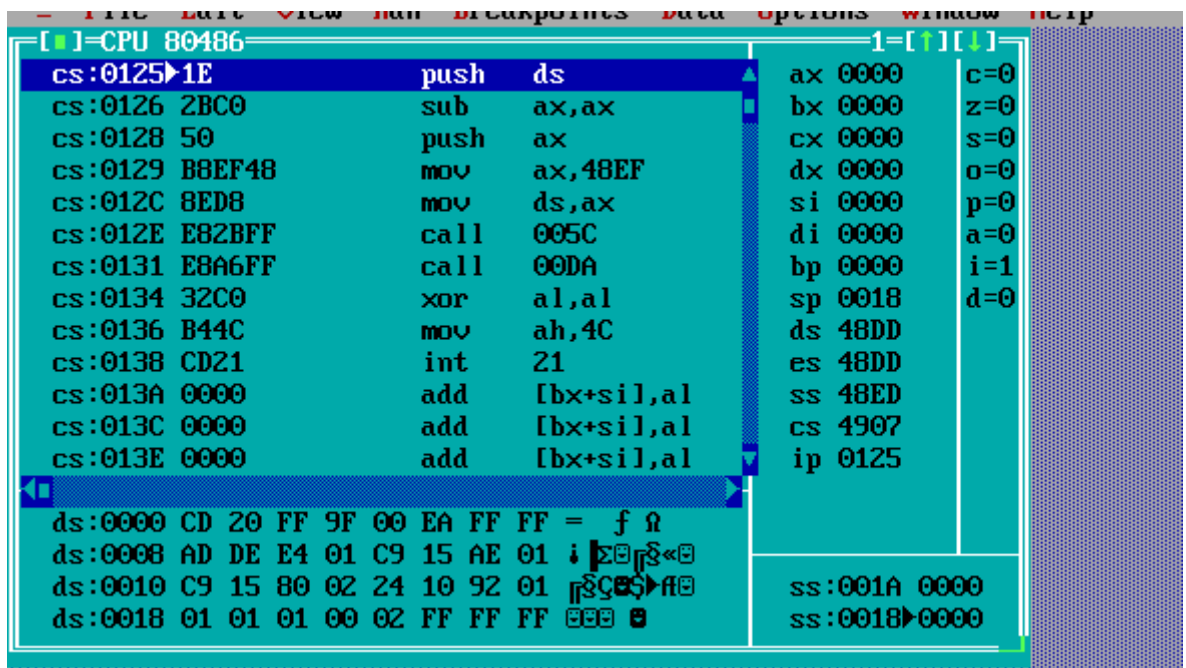


Рисунок 8 – “Хороший” EXE модуль в отладчике

Выводы.

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: progcom.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
; Данные
PC db 'Your PC type is -> PC',0DH,0AH,'$'
PC_XT db 'Your PC type is -> PC/XT',0DH,0AH,'$'
AT db 'Your PC type is -> AT',0DH,0AH,'$'
PS2_30 db 'Your PC type is -> PS2 модель 30',0DH,0AH,'$'
PS2_50_60 db 'Your PC type is -> PS2 модель 50 или 60',0DH,0AH,'$'
PS2_80 db 'Your PC type is -> PS2 модель 80',0DH,0AH,'$'
PCJR db 'Your PC type is -> PCjr',0DH,0AH,'$'
PC_CONVERTIBLE db 'Your PC type is -> PC Convertible',0DH,0AH,'$'
PC_UNK db 'Your PC type is -> ',0DH,0AH,'$'

VERSIONS db 'Version MS-DOS: . ',0DH,0AH,'$'
SERIAL_NUMBER db 'Serial number OEM: ',0DH,0AH,'$'
USER_NUMBER db 'User serial number:      H $'

; 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 шест. 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
;-----
WRITESTRING PROC near
    mov AH,09h
    int 21h
    ret
WRITESTRING ENDP

CHECK_PC_TYPE PROC near
    mov ax,0F000h
    mov es,ax
    mov ah,es:[0FFFEh]
    ;mov ah,0EBh

    cmp ah,0FFh
    je pc_lab

    cmp ah,0FEh
    je xt_lab

    cmp ah, 0FBh
    je xt_lab

    cmp ah, 0FCh

```

```

je at_lab

cmp ah, 0FAh
je ps230_lab

cmp ah, 0FCh
je ps25060_lab

cmp ah, 0F8h
je ps280_lab

cmp ah, 0FDh
je pcjr_lab

cmp ah, 0F9h
je pcconv_lab

unk:
    mov di,offset PC_UNK
    add di,19
    mov al,ah
    call BYTE_TO_HEX
    mov [di], ax
    mov dx,offset PC_UNK
    jmp final_1

pc_lab:
    mov dx,offset PC
    jmp final_1

xt_lab:
    mov dx,offset PC_XT
    jmp final_1

at_lab:
    mov dx,offset AT
    jmp final_1

ps230_lab:
    mov dx,offset PS2_30
    jmp final_1

ps25060_lab:
    mov dx,offset PS2_50_60
    jmp final_1

ps280_lab:
    mov dx,offset PS2_80
    jmp final_1

pcjr_lab:
    mov dx,offset PCJR
    jmp final_1

pcconv_lab:
    mov dx,offset PC_CONVERTIBLE
    jmp final_1

```

```

final_1:
    call WRITESTRING
    ret
CHECK_PC_TYPE ENDP

CHECK_OS_VERS PROC near
    mov ah,30h
    int 21h
    push ax

    mov si,offset VERSIONS
    add si,16
    call BYTE_TO_DEC
    pop ax
    add si,3
    mov al,ah
    call BYTE_TO_DEC
    mov dx,offset VERSIONS
    call WRITESTRING

    mov si,offset SERIAL_NUMBER
    add si,21
    mov al,bh
    call BYTE_TO_DEC
    mov dx,offset SERIAL_NUMBER
    call WRITESTRING

    mov di, offset USER_NUMBER
    add di, 25
    mov ax, cx
    call WRD_TO_HEX
    mov al, bl
    call BYTE_TO_HEX
    sub di,2
    mov [di], ax
    mov dx, offset USER_NUMBER
    call WRITESTRING
    ret

final_2:
    ret
CHECK_OS_VERS ENDP

; Код
BEGIN:
    call CHECK_PC_TYPE
    call CHECK_OS_VERS
    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
END START

```

Название файла: progexe.asm

```

AStack SEGMENT STACK
    DW 12 DUP(?)

```

AStack ENDS

DATA SEGMENT

```
PC db 'Your PC type is -> PC',0DH,0AH,'$'
PC_XT db 'Your PC type is -> PC/XT',0DH,0AH,'$'
AT db 'Your PC type is -> AT',0DH,0AH,'$'
PS2_30 db 'Your PC type is -> PS2 модель 30',0DH,0AH,'$'
PS2_50_60 db 'Your PC type is -> PS2 модель 50 или 60',0DH,0AH,'$'
PS2_80 db 'Your PC type is -> PS2 модель 80',0DH,0AH,'$'
PCJR db 'Your PC type is -> PCjr',0DH,0AH,'$'
PC_CONVERTIBLE db 'Your PC type is -> PC Convertible',0DH,0AH,'$'
PC_UNK db 'Your PC type is -> ',0DH,0AH,'$'
```

```
VERSIONS db 'Version MS-DOS: . ',0DH,0AH,'$'
SERIAL_NUMBER db 'Serial number OEM: ',0DH,0AH,'$'
USER_NUMBER db 'User serial number: H $'
```

DATA ENDS

CODE SEGMENT

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

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 шест. 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
;-----
WRITESTRING PROC near
    mov AH,09h
    int 21h
    ret
WRITESTRING ENDP

CHECK_PC_TYPE PROC near
    mov ax,0F000h
    mov es,ax
    mov ah,es:[0FFFEh]
    ;mov ah,0EBh

    cmp ah,0FFh
    je pc_lab

    cmp ah,0FEh
    je xt_lab

    cmp ah, 0FBh
    je xt_lab

    cmp ah, 0FCh
    je at_lab

    cmp ah, 0FAh
    je ps230_lab

```

```

    cmp ah, 0FCh
    je ps25060_lab

    cmp ah, 0F8h
    je ps280_lab

    cmp ah, 0FDh
    je pcjr_lab

    cmp ah, 0F9h
    je pcconv_lab

unk:
    mov di,offset PC_UNK
    add di,19
    mov al,ah
    call BYTE_TO_HEX
    mov [di], ax
    mov dx,offset PC_UNK
    jmp final_1

pc_lab:
    mov dx,offset PC
    jmp final_1

xt_lab:
    mov dx,offset PC_XT
    jmp final_1

at_lab:
    mov dx,offset AT
    jmp final_1

ps230_lab:
    mov dx,offset PS2_30
    jmp final_1

ps25060_lab:
    mov dx,offset PS2_50_60
    jmp final_1

ps280_lab:
    mov dx,offset PS2_80
    jmp final_1

pcjr_lab:
    mov dx,offset PCJR
    jmp final_1

pcconv_lab:
    mov dx,offset PC_CONVERTIBLE
    jmp final_1

final_1:
    call WRITESTRING
    ret
CHECK_PC_TYPE ENDP

```

```

CHECK_OS_VERS PROC near
    mov ah,30h
    int 21h
    push ax

    mov si,offset VERSIONS
    add si,16
    call BYTE_TO_DEC
    pop ax
    add si,3
    mov al,ah
    call BYTE_TO_DEC
    mov dx,offset VERSIONS
    call WRITESTRING

    mov si,offset SERIAL_NUMBER
    add si,21
    mov al,bh
    call BYTE_TO_DEC
    mov dx ,offset SERIAL_NUMBER
    call WRITESTRING

    mov di, offset USER_NUMBER
    add di, 25
    mov ax, cx
    call WRD_TO_HEX
    mov al, bl
    call BYTE_TO_HEX
    sub di,2
    mov [di], ax
    mov dx, offset USER_NUMBER
    call WRITESTRING
    ret

final_2:
    ret
CHECK_OS_VERS ENDP

Main PROC FAR
    push DS
    sub AX,AX
    push AX
    mov AX,DATA
    mov DS,AX

    call CHECK_PC_TYPE
    call CHECK_OS_VERS
    xor AL,AL
    mov AH,4Ch
    int 21H

Main ENDP
CODE ENDS
    END Main

```