

**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**

Студентка гр. 7381

Преподаватель

Кревчик А.Б.

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 загрузки в основную память.

### Ход работы.

TETR\_TO\_HEX – вспомогательная функция, переводит из двоичной в шестнадцатеричную систему.

BYTE\_TO\_HEX – переводит число из регистра AL в шестнадцатеричную систему.

WRD\_TO\_HEX – переводит число из регистра AX в шестнадцатеричную систему.

BYTE\_TO\_DEC – переводит число из регистра AL в десятичную систему.

PRINT – печатает сообщение на экран.

TYPE\_PC – получает тип ПК.

PRINT\_TYPE\_PC – определяет и выводит тип ПК.

VERSION\_DOS – определяет версию системы.

OEM\_NUM – определяет серийный номер OEM.

USER\_NUM – определяет серийный модуль пользователя.

Программа выводит на экран тип IBM PC, версию ОС, серийный номер OEM и серийный номер пользователя.

Результаты работы программы представлены на рис. 1-3.



Рисунок 1 – Результат выполнения «плохого» .EXE модуля

```
C:\>LR1.COM
Type PC: AT
Version MS DOS: 5.0
OEM serial number: 255
User serial number: 000000
```

Рисунок 2 – Результат выполнения «хорошего» .COM модуля

```
Type PC: AT
Version MS DOS: 5.0
OEM serial number: 255
User serial number: 000000
```

Рисунок 3 – Результат выполнения «хорошего» .EXE модуля

### **Выводы.**

В ходе лабораторной работы были изучены различия в структурах исходных текстов модулей типов .COM и .EXE, 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?

Ровно один сегмент – сегмент кода.

2. EXE-программа?

Один и больше.

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

ORG – сдвигает адресацию в программе на 256 бай для расположения PSP,

ASSUME – ставит сегментным регистрам в соответствие требуемые сегменты.

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

Нельзя использовать команды с дальней адресацией, поскольку в COM-

программе отсутствует таблица настроек, которая указывает, какие абсолютные адреса при загрузке должны быть изменены, так как до загрузки неизвестно, куда будет загружена программа.

### Отличия форматов файлов COM и EXE модулей.

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

COM-файл содержит данные и машинные команды. Код начинается с адреса 0h (см. рис. 4).

00000000: E9 14 02 54 79 70 65 20	50 43 3A 20 50 43 0D 0A	éType PC: PC
00000001: 24 54 79 70 65 20 50 43	3A 20 50 43 2F 58 54 0D	\$Type PC: PC/XT
00000002: 0A 24 54 79 70 65 20 50	43 3A 20 41 54 0D 0A 24	\$Type PC: AT
00000003: 54 79 70 65 20 50 43 3A	20 50 53 32 20 6D 6F 64	Type PC: PS2 mod
00000004: 65 6C 20 33 30 0A 0D 24	54 79 70 65 20 50 43 3A	el 30Type PC:
00000005: 20 50 53 32 20 6D 6F 64	65 6C 20 35 30 20 6F 72	PS2 model 50 or
00000006: 20 36 30 0A 0D 24 54 79	70 65 20 50 43 3A 20 50	60Type PC: P
00000007: 53 32 20 6D 6F 64 65 6C	20 38 30 0A 0D 24 54 79	S2 model 80
00000008: 70 65 20 50 43 3A 20 50	43 6A 72 0A 0D 24 54 79	pe PC: PCjr
00000009: 70 65 20 50 43 3A 20 50	43 20 43 6F 6E 76 65 72	pe PC: PC Conver
0000000A: 74 69 62 6C 65 0A 0D 24	56 65 72 73 69 6F 6E 20	tibleVersion
0000000B: 4D 53 20 44 4F 53 3A 20	20 2E 20 20 0A 0D 24 4F	MS DOS: .
0000000C: 45 4D 20 73 65 72 69 61	6C 20 6E 75 6D 62 65 72	EM serial number
0000000D: 3A 20 20 20 20 0A 0D 24	55 73 65 72 20 73 65 72	: User ser
0000000E: 69 61 6C 20 6E 75 6D 62	65 72 3A 20 20 20 20 20	ial number:

Рисунок 4 – Начало кода COM - файла

#### 2. Какова структура файла «плохого» EXE? С какого адреса располагается код?

Что располагается с адреса 0?

В «плохом» файле EXE данные и код содержатся в одном сегменте. Код располагается с адреса 300h(см. рис. 5). С адреса 0h располагается заголовок, таблица настроек, а также зарезервированные директивой ORG 100h байт.

0000002C0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000002D0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000002E0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000002F0: 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000300: E9 14 02 54 79 70 65 20	50 43 3A 20 50 43 0D 0A	éType PC: PC
000000310: 24 54 79 70 65 20 50 43	3A 20 50 43 2F 58 54 0D	\$Type PC: PC/XT
000000320: 0A 24 54 79 70 65 20 50	43 3A 20 41 54 0D 0A 24	\$Type PC: AT
000000330: 54 79 70 65 20 50 43 3A	20 50 53 32 20 6D 6F 64	Type PC: PS2 mod
000000340: 65 6C 20 33 30 0A 0D 24	54 79 70 65 20 50 43 3A	el 30Type PC:
000000350: 20 50 53 32 20 6D 6F 64	65 6C 20 35 30 20 6F 72	PS2 model 50 or
000000360: 20 36 30 0A 0D 24 54 79	70 65 20 50 43 3A 20 50	60Type PC: P

Рисунок 5 – Начало кода «плохого» EXE

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

В «хорошем» EXE код, стек и данные выделены в отдельные сегменты, тогда как в «плохом» всего один сегмент и для данных и для кода. В EXE программах нет необходимости в директиве ORG, поскольку загрузчик ставит программу после PSP. Код начинается с 400h.

00000003D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000003E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000003F0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000400:	54 79 70 65 20 50 43 3A	20 50 43 0D 0A 24 54 79	Type PC: PC
0000000410:	70 65 20 50 43 3A 20 50	43 2F 58 54 0D 0A 24 54	re PC: PC/XT
0000000420:	79 70 65 20 50 43 3A 20	41 54 0D 0A 24 54 79 70	ype PC: AT
0000000430:	65 20 50 43 3A 20 50 53	32 20 AC AE A4 A5 AB EC	e PC: PS2
0000000440:	20 33 30 0A 0D 24 54 79	70 65 20 50 43 3A 20 50	30
0000000450:	53 32 20 AC AE A4 A5 AB	EC 20 35 30 20 6F 72 20	S2
0000000460:	36 30 0A 0D 24 54 79 70	65 20 50 43 3A 20 50 53	60
0000000470:	32 20 AC AE A4 A5 AB EC	20 38 30 0A 0D 24 54 79	2
0000000480:	70 65 20 50 43 3A 20 50	43 6A 72 0A 0D 24 54 79	pe PC: PCjr
0000000490:	70 65 20 50 43 3A 20 50	43 20 43 6F 6E 76 65 72	pe PC: PC Conver
00000004A0:	74 69 62 6C 65 0A 0D 24	56 65 72 73 69 6F 6E 20	tibles

Рисунок 6 – Начало кода «хорошего» EXE

### Загрузка COM модуля в основную память.

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

После загрузки COM-программы в память, сегментные регистры указывают на начало PSP. Начало кода определяется директивой ORG от начала выделенного фрагмента (100h).

#### 2. Что располагается с адреса 0?

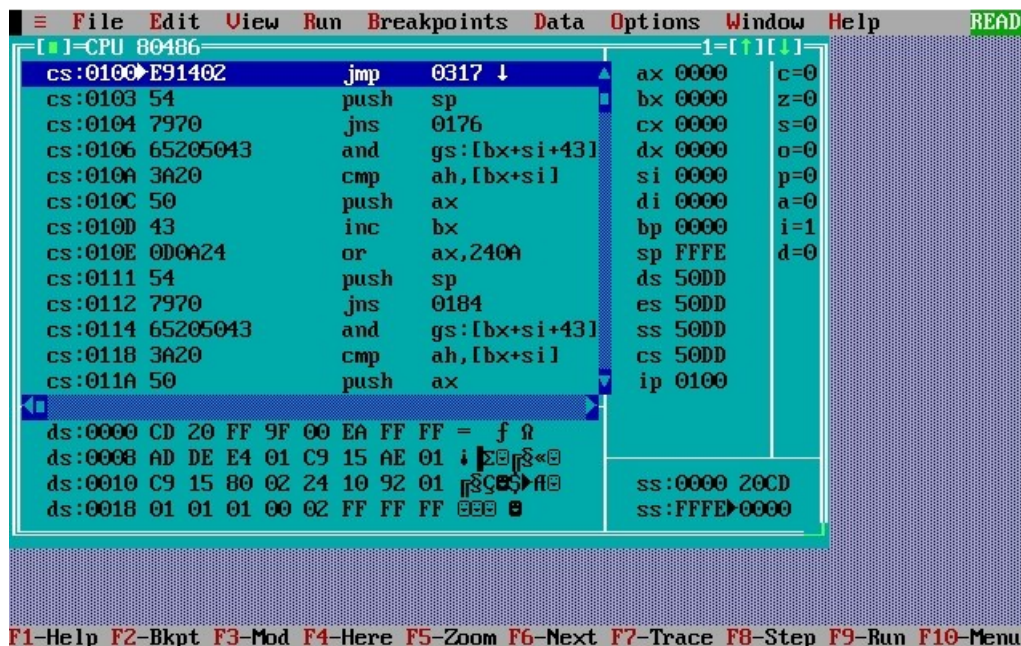
PSP.

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

Все сегментные регистры указывают на начало PSP.

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

Стек занимает весь фрагмент памяти, выделенный под программу и определяется регистрами SS и SP. Он занимает адреса 0000h-FFFFh.



### 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, SS – на начало сегмента стека, CS – на начало сегмента кода. В IP загружается смещение точки входа в программу, которая берётся из метки после директивы END.

2. На что указывают регистры DS и ES?

Начало сегмента PSP.

3. Как определяется стек?

Для стека в программе выделяется отдельный сегмент с параметром STACK. При запуске программы в SS заносится адрес сегмента стека, а в SP – адрес верхушки стека.

4. Как определяется точка входа?

С помощью директивы END, после которой указывается метка, куда переходит программа при запуске.

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД .COM МОДУЛЯ

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

```
;„ЪЌЌ›...
PC           DB 'TYPE PC: PC', 13, 10, '$'
PC_XT        DB 'TYPE PC: PC/XT', 13, 10, '$'
AT           DB 'TYPE PC: AT', 13, 10, '$'
PS2_1        DB 'TYPE PC: PS2 MODEL 30', 10, 13, '$'
PS2_2        DB 'TYPE PC: PS2 MODEL 50 OR 60', 10, 13, '$'
PS2_3        DB 'TYPE PC: PS2 MODEL 80', 10, 13, '$'
PCJR         DB 'TYPE PC: PCJR', 10, 13, '$'
PC_CONVERTIBLE DB 'TYPE PC: PC CONVERTIBLE', 10, 13, '$'
VERS         DB 'VERSION MS DOS: . ', 10, 13, '$'
OEM          DB 'OEM SERIAL NUMBER: ', 10, 13, '$'
USER         DB 'USER SERIAL NUMBER: ', 10, 13, '$'
```

```
;ЏЪЏ–...„Ъ›
```

```
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 ;Ў ©В Ў AL ЇЃАЃЎ®¤ЁВБП Ў ¢Ў БЁ¬Ў®«
ИЃБВ. ЗЁБ« Ў АХ
```

```
              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 ;ЇЃАЃЎ®¤ Ў 16 Б/Б 16-БЁ А §АП¤®Ј® ЗЁБ« , Ў АХ -
ЗЁБ«®, DI - ¢АЃБ Ї®Б«Ѓ¤ЃЈ® БЁ¬Ў®«
              PUSHBX
              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
        PUSHAX
        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
        POP     AX
        RET
BYTE_TO_DEC ENDP

PRINT PROC NEAR
        PUSH    AX
        MOV     AH,09H
        INT     21H
        POP     AX
        RET
PRINT ENDP

TYPE_PC PROC NEAR ;ї®«ГЗГ'ЕГ БЕЇ ЦЛЬ
        PUSHDS
        MOV     BX,0F000H
        MOV     DS,BX

```



```

SUB    AX,AX
MOV    AH,DS:[0FFFEH]
POP    DS
RET
TYPE_PC ENDP

```

```

PRINT_TYPE_PC PROC NEAR    ;@IAΓαΓ«ΓΈΓ Έ ŸΛŸ@α ΒΕΪ  ϘЉ

```

```

PUSH    AX
PUSHBX
PUSHDI

```

```

MOV  DX, OFFSET PC
CMP  AH, 0FFH
JE   PRINT_MSG

```

```

MOV  DX, OFFSET PC_XT
CMP  AH, 0FEH
JE   PRINT_MSG

```

```

MOV  DX, OFFSET PC_XT
CMP  AH, 0FBH
JE   PRINT_MSG

```

```

MOV  DX, OFFSET AT
CMP  AH, 0FCH
JE   PRINT_MSG

```

```

MOV  DX, OFFSET PS2_1
CMP  AH, 0FAH
JE   PRINT_MSG

```

```

MOV  DX, OFFSET PS2_2
CMP  AH, 0FCH
JE   PRINT_MSG

```

```

MOV  DX, OFFSET PS2_3
CMP  AH, 0F8H
JE   PRINT_MSG

```

```

MOV  DX, OFFSET PCJR
CMP  AH, 0FDH
JE   PRINT_MSG

```

```

MOV  DX, OFFSET PC_CONVERTIBLE
CMP  AH, 0F9H
JE   PRINT_MSG

```

```

MOV    AL,AH
CALL  BYTE_TO_HEX
MOV    DX, AX

```

```

PRINT_MSG:
    CALL PRINT
    POP DI
    POP BX
    POP AX
    RET
PRINT_TYPE_PC ENDP

VERSION_DOS PROC NEAR ;®ïAΓ¤Γ«ΓĖΓ ŸΓAБĖĖ БĖБBΓ¬Л
    PUSHAX
    PUSHSI
    MOV SI, OFFSET VERS
    ADD SI, 10H
    CALL BYTE_TO_DEC
    ADD SI, 3H
    MOV AL, AH
    CALL BYTE_TO_DEC
    MOV DX, OFFSET VERS
    CALL PRINT
    POP SI
    POP AX
    RET
VERSION_DOS ENDP

OEM_NUM PROC NEAR ;®ïAΓ¤Γ«ΓĖΓ БΓAĖ©®J® ®¬ΓA OEM
    PUSHAX
    PUSHBX
    PUSHSI
    MOV AL, BH
    MOV SI, OFFSET OEM
    ADD SI, 15H
    CALL BYTE_TO_DEC
    MOV DX, OFFSET OEM
    CALL PRINT
    POP SI
    POP BX
    POP AX
    RET
OEM_NUM ENDP

USER_NUM PROC NEAR ;®ïAΓ¤Γ«ΓĖΓ БΓAĖ©®J® ®¬ΓA ĩ®«M$®Ÿ BΓ«П
    PUSH CX
    PUSH DI
    PUSH AX
    MOV DI, OFFSET USER
    ADD DI, 19H
    MOV AX, CX
    CALL WRD_TO_HEX
    MOV AL, BL
    MOV DI, OFFSET USER
    ADD DI, 14H

```

```

CALL BYTE_TO_HEX
MOV     [DI], AX
MOV  DX, OFFSET USER
CALL PRINT
POP  AX
POP  DI
POP  CX
RET

```

```

USER_NUM ENDP

```

```

BEGIN:
CALL     TYPE_PC
CALL     PRINT_TYPE_PC
MOV      AH, 30H
INT      21H
CALL     VERSION_DOS
CALL     OEM_NUM
CALL     USER_NUM
XOR      AL,AL
MOV      AH,4CH
INT      21H
TESTPC ENDS
END START

```

## **ПРИЛОЖЕНИЕ Б** **ИСХОДНЫЙ КОД .EXE МОДУЛЯ**

ASTACK SEGMENT STACK

ASTACK ENDS

DATA SEGMENT

;,,ТбКК>...

PC

10,'\$'

PC\_XT

10,'\$'

AT

10,'\$'

PS2\_1

30', 10, 13, '\$'

PS2\_2

50 or 60', 10, 13, '\$'

PS2\_3

80', 10, 13, '\$'

PCjr

'\$'

PC\_CONVERTIBLE

13,'\$'

VERS

10, 13, '\$'

OEM

10, 13, '\$'

USER

', 10, 13, '\$'

DATA ENDS

CODE SEGMENT

ES:NOTHING, SS:ASTACK

;ЎђђТ-...,,“ђ»

START: JMP BEGIN

TETR\_TO\_HEX PROC NEAR

NEXT:

TETR\_TO\_HEX ENDP

BYTE\_TO\_HEX PROC NEAR

бѐ-ŷ®« иѓбв. зѐб« ŷ AX

DW 0100h DUP(?)

db 'Type PC: PC', 13,

db 'Type PC: PC/XT', 13,

db 'Type PC: AT', 13,

db 'Type PC: PS2 ¬®«М

db 'Type PC: PS2 ¬®«М

db 'Type PC: PS2 ¬®«М

db 'Type PC: PCjr',10, 13,

db 'Type PC: PC Convertible',10,

db 'Version MS DOS: . ',

db 'OEM serial number: ',

db 'User serial number:

ASSUME CS:CODE, DS:DATA,

and AL,0Fh

cmp AL,09

jbe NEXT

add AL,07

add AL,30h

ret

;ŷ ©в ŷ AL иѓаѓŷ®«ѐбп ŷ «ŷ

push CX

mov AH,AL

call TETR\_TO\_HEX

xchg AL,AH

бв аи п жёда

BYTE\_TO\_HEX ENDP

WRD\_TO\_HEX PROC NEAR ; и́агъ® ы 16 б/б 16-вё а шап®J® зёб« , ы AX - зёб«®, DI -  
агб и́б«г®J® бё¬ы®«

WRD\_TO\_HEX

BYTE\_TO\_DEC PROC NEAR  
и́«п ¬« ии́© жёдал

ёбе®ал© ы ©в

loop\_bd: div

end\_l: pop

```
mov CL,4
shr AL,CL
call TETR_TO_HEX ;ы AL

pop CX
;ы AH ¬« ии п
ret
```

```
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
ENDP
```

; и́агъ® ы ©в ы 10б/б, SI - агб

push AX ; AL б®а́аёв

```
push CX
push DX
xor AH,AH
xor DX,DX
mov CX,10
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
DX
pop CX
pop AX
```

BYTE\_TO\_DEC ENDP

PRINT PROC NEAR

PRINT ENDP

TYPE\_PC PROC NEAR

TYPE\_PC ENDP

PRINT\_TYPE\_PC PROC NEAR

ret

```
;ГЗ ВМ Б@@ЙГП нСа
push AX
mov AH, 09h
int 21h
pop AX
ret
```

```
;@«ГЗГЕГ ВЕИ ЦЛБ
push DS
mov BX, 0F000H
mov DS, BX
sub AX, AX
mov AH, DS:[0FFFEH]
pop DS
ret
```

```
;л҃@ ВЕИ ЦЛБ
push AX
push BX
push DI

mov DX, OFFSET PC
cmp AH, 0FFh
je print_msg

mov DX, OFFSET PC_XT
cmp AH, 0FEh
je print_msg

mov DX, OFFSET PC_XT
cmp AH, 0FBh
je print_msg

mov DX, OFFSET AT
cmp AH, 0FCh
je print_msg

mov DX, OFFSET PS2_1
cmp AH, 0FAh
je print_msg

mov DX, OFFSET PS2_2
cmp AH, 0FCh
je print_msg

mov DX, OFFSET PS2_3
```

PC\_CONVERTIBLE

```
cmp    AH, 0F8h
je      print_msg

mov     DX, OFFSET PCjr
cmp     AH, 0FDh
je      print_msg
```

```
mov     DX, OFFSET
```

```
cmp     AH, 0F9h
je      print_msg
```

```
mov     AL, AH
call    BYTE_TO_HEX
mov     DX, AX
```

```
print_msg:
call    PRINT
pop     DI
pop     BX
pop     AX
ret
```

PRINT\_TYPE\_PC ENDP

VERSION\_DOS PROC NEAR

```
push    AX
push    SI
mov     SI, OFFSET VERS
add     SI, 10h
call    BYTE_TO_DEC
add     SI, 3h
mov     AL, AH
call    BYTE_TO_DEC
mov     DX, OFFSET VERS
call    PRINT
pop     SI
pop     AX
ret
```

VERSION\_DOS ENDP

OEM\_NUM PROC NEAR

```
push    AX
push    BX
push    SI
mov     AL, BH
mov     SI, OFFSET OEM
add     SI, 15h
call    BYTE_TO_DEC
mov     DX, OFFSET OEM
call    PRINT
pop     SI
```

	pop	BX
	pop	AX
OEM_NUM ENDP	ret	
USER_NUM PROC NEAR		
	push	CX
	push	DI
	push	AX
	mov	DI, OFFSET USER
	add	DI, 19h
	mov	AX, CX
	call	WRD_TO_HEX
	mov	AL, BL
	mov	DI, OFFSET USER
	add	DI, 14h
	call	BYTE_TO_HEX
	mov	[DI], AX
	mov	DX, OFFSET USER
	call	PRINT
	pop	AX
	pop	DI
	pop	CX
	ret	
USER_NUM ENDP		
BEGIN:		
	mov	AX, DATA
	mov	DS, AX
	mov	BX, DS
	call	TYPE_PC
	call	PRINT_TYPE_PC
	mov	AH, 30h
	int	21h
	call	VERSION_DOS
	call	OEM_NUM
	call	USER_NUM
	xor	AL, AL
	mov	AH, 4Ch
	int	21h
CODE ENDS		
END START		