**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Операционные системы»**

Тема: Исследование структур загрузочных модулей

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Никитин К.В. |
| Преподаватель |  | Губкин А. Ф. |

Санкт-Петербург

2018

# Постановка задачи.

* 1. **Цель работы.**

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

* 1. **Функции и структуры данных.**

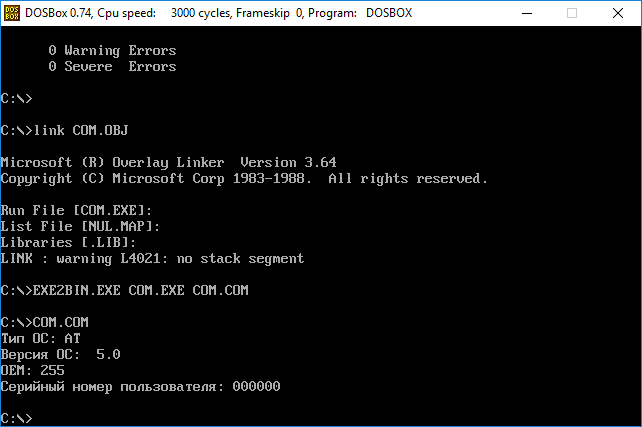
|  |  |
| --- | --- |
| **Название** | **Назначение** |
| Write | Вывод сообщения на экран |
| TETR\_TO\_HEX | Десятичная цифра переводится в код символа |
| BYTE\_TO\_HEX | Байт в AL переводится в два символа шестн. числа в AX |
| WRD\_TO\_HEX | Перевод в 16 с/с 16-ти разрядного числа |
| BYTE\_TO\_DEC | Перевод байта в 16-ной с/с в символьный код в 10-ной с/с |
| GetOSver | Определение версии ОС, ОЕМ и серийного номера |
| GetOStype | определение типа ОС |

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| OS | db | Тип ОС |
| OS\_VERS | db | Версия ОС |
| OS\_OEM | db | Серийный номер OEM |
| USER\_NUMBER | 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 |

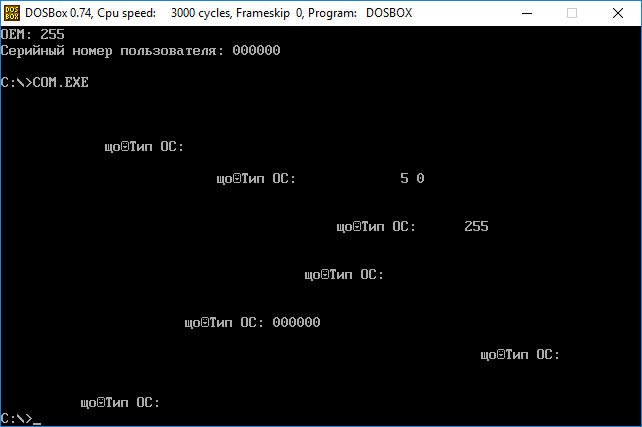
* 1. **Последовательность действий, выполняемых утилитой.**

1. Определение и вывод на экран версии IBM PC.
2. Определение и вывод на экран версии MS-DOS.
3. Определение и вывод на экран серийного номера OEM.
4. Определение и вывод на экран серийного номера пользователя.

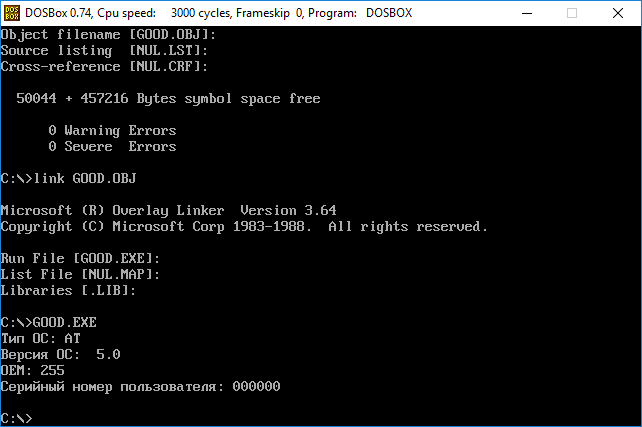
Результаты работы .СОМ файла, плохого .ЕХЕ файла, хорошего .EXE файла представлены на рис. 1, рис. 2, рис. 3 соответственно.



*Рисунок 1. Результат выполнения программы com.com*

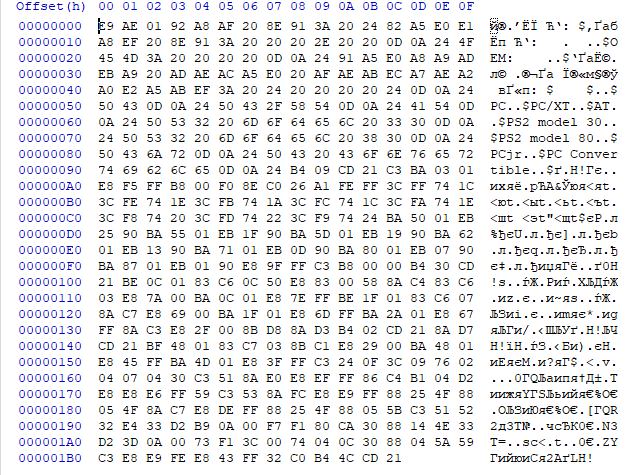


*Рисунок 2. Результат выполнения программы com.exe*

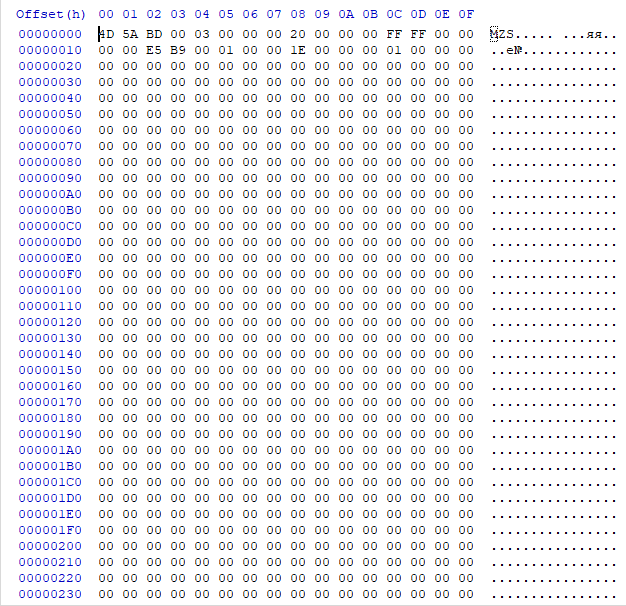


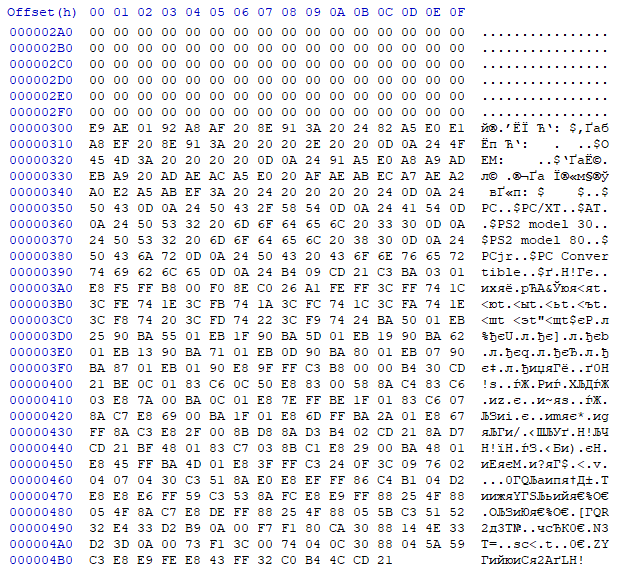
*Рисунок 3. Результат выполнения программы good.exe*

Шестнадцатеричные представления .COM файла, хорошего .ЕХЕ файла, плохого .ЕХЕ файла представлены на рис. 4, рис. 5, рис. 6 соответственно (см. след. страницу).

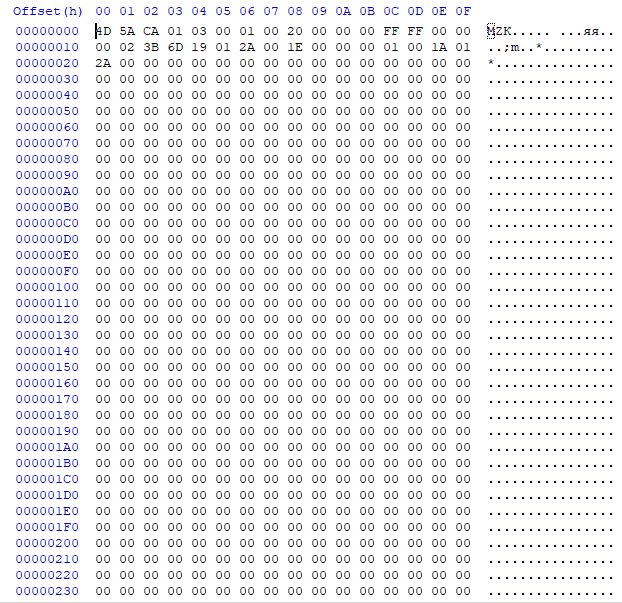


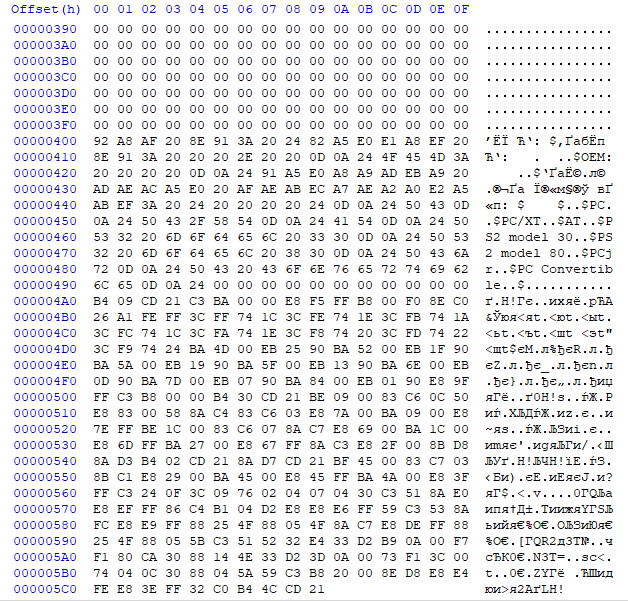
*Рисунок 4. HEX представление .СОМ файла.*





*Рисунок 5. HEX представление плохого .EXE файла.*





*Рисунок 6. HEX представление хорошего .EXE файла.*

**Вывод.**

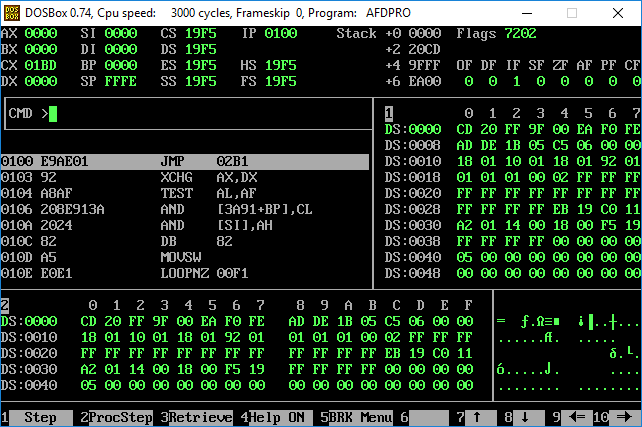
В результате выполнения лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .СОМ и .ЕХЕ, структур файлов загрузочных модулей и способов их загрузки в основную память. Текст реализованной программы laba\_1.asm приложен в Приложении A.

# Ответы на контрольные вопросы.

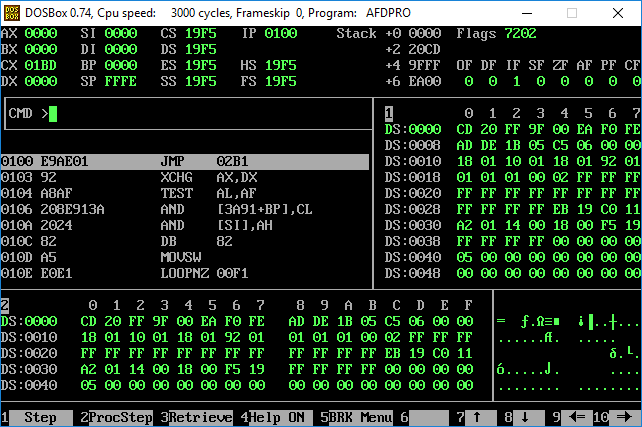
* 1. **Отличия исходных текстов COM и EXE программ**
     1. *Сколько сегментов должна содержать COM-программа?*
* Один сегмент.
  + 1. *EXE-программа?*
* Не менее одного сегмента.
  + 1. *Какие директивы должны обязательно быть в тексте COM-программы?*
* ORG 100h. Директива для задания смещение в 256 байт от нулевого адреса, так как при загрузке модуля в ОП в начале .COM – программы определяется 256 байтный префикс программного сегмента. Помимо этого должна быть директива ASSUME. Директива ASSUME используется для указания ассемблеру, что имя логического сегмента следует использовать для определенного сегмента.
  + 1. *Все форматы команд можно использовать в COM-программе?*
* Нельзя использовать команды MOV, SEGMENT,CODE, REGISTER и команды, использующие дальнюю адресацию.

В данных командах используется таблица разметки (relocation table), в которых содержатся адреса сегментов.

* 1. **Отличия форматов файлов COM и EXE программ**
     1. *Какова структура файла COM? С какого адреса располагается код?*
* Структура .СОМ файла – данные и команды. Код располагается с 0h, а при загрузке модуля в ОП устанавливается смещение в 100h
  + 1. *Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?*
* Данные и код содержатся в одном сегменте. Заголовок начинается с 0, код располагается с 300h. 200h идет на заголовок, 100h - сдвиг директивы ORG 100h.
  + 1. *Какова структура «хорошего» EXE? Чем он отличается от файла «плохого» EXE?*
* Данные, стек и код хорошего .EXE файла разделены по сегментам. Не требуется директива ORF 100h, так как загрузчик автоматически положит программу после PSP.
  1. **Загрузка COM модуля в основную память**



* + 1. *Какой формат загрузки модуля COM? С какого адреса располагается код?*
* После загрузки модуля в память сегментные регистры указывают на PSP, а код располагается с 100h.
  + 1. *Что располагается с адреса 0?*
* PSP
  + 1. *Какие значения имеют сегментные регистры? На какие области памяти они указывают?*
* Все сегментные регистры равны 19F5 и указывают на начало PSP
  + 1. *Как определяется стек? Какую область памяти он занимает? Какие адреса?*
* Cтек определяется автоматически, занимает весь сегмент программы. SP указывает на конец стека FFFEh. Диапазон адресов: 0h – FFFEh. Стек растет от больших адресов к меньшим.
  1. **Загрузка «хорошего» EXE модуля в основную память**



* + 1. *Как загружается «хороший» EXE? Какие значения имеют сегментные регистры?*
* DS, ES устанавливаются на начало сегмента PSP; SS – на начало сегмента стека; СS – начало сегмента команд; в IP загружаем смещение точки входа
  + 1. *На что указывают регистры DS и ES?*
* на начало PSP
  + 1. *Как определяется стек?*
* директивой .STACK, которая определяет начало сегмента стека. При исполнении в SS находится адрес начала сегмента стека, в SP - его вершины.
  + 1. *Как определяется точка входа?*
* директивой END, после нее ставится метка, на которую программа переходит при запуске.

# Приложение А.

# сom.asm

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

OS db 'Тип ОС: $'

OS\_VERSION db 'Версия ОС: . ',0DH,0AH,'$'

OS\_OEM db 'OEM: ',0DH,0AH,'$'

USER\_NUM db 'Серийный номер пользователя: ','$'

SPACE db ' $'

ENDSTR db 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,'$'

Write PROC near

mov AH,09h

int 21h

ret

Write ENDP

GetOStype PROC near

mov dx, OFFSET OS

call Write

mov ax,0F000h

mov es,ax

mov ax,es:0FFFEh

cmp al,0FFh

je PC\_

cmp al,0FEh

je PCXT\_

cmp al,0FBh

je PCXT\_

cmp al,0FCh

je AT\_

cmp al,0FAh

je PS2\_30\_

cmp al,0F8h

je PS2\_80\_

cmp al,0FDh

je PCjr\_

cmp al,0F9h

je PC\_CNV\_

PC\_:

mov dx, OFFSET PC

jmp endl

PCXT\_:

mov dx, OFFSET PCXT

jmp endl

AT\_:

mov dx, OFFSET \_AT

jmp endl

PS2\_30\_:

mov dx, OFFSET PS2\_30

jmp endl

PS2\_80\_:

mov dx, OFFSET PS2\_80

jmp endl

PCjr\_:

mov dx, OFFSET PCjr

jmp endl

PC\_CNV\_:

mov dx, OFFSET PC\_Cnv

jmp endl

endl:

call Write

ret

GetOStype ENDP

GetOSver PROC near

mov ax,0

mov ah,30h

int 21h

mov si,offset OS\_VERSION

add si,12

push ax

call BYTE\_TO\_DEC

pop ax

mov al,ah

add si,3

call BYTE\_TO\_DEC

mov dx,offset OS\_VERSION

call Write

mov si,offset OS\_OEM

add si,7

mov al,bh

call BYTE\_TO\_DEC

mov dx,offset OS\_OEM

call Write

mov dx,offset USER\_NUM

call Write

mov al,bl

call BYTE\_TO\_HEX

mov bx,ax

mov dl,bl

mov ah,02h

int 21h

mov dl,bh

int 21h

mov di,offset SPACE

add di,3

mov ax,cx

call WRD\_TO\_HEX

mov dx,offset SPACE

call Write

mov dx,offset ENDSTR

call Write

ret

GetOSver ENDP

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\_: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

BEGIN:

call GetOStype

call GetOSver

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START

# Приложение Б.

# exe.asm

STACK SEGMENT STACK

DW 0100h DUP(?)

STACK ENDS

DATA SEGMENT

OS db 'Тип ОС: $'

OS\_VERSION db 'Версия ОС: . ',0DH,0AH,'$' ;

OS\_OEM db 'OEM: ',0DH,0AH,'$'

USER\_NUM db 'Серийный номер пользователя: ','$'

SPACE db ' $'

ENDSTR db 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

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:STACK

Write PROC near

mov AH,09h

int 21h

ret

Write ENDP

GetOStype PROC near

mov dx, OFFSET OS

call Write

mov ax,0F000h

mov es,ax

mov ax,es:0FFFEh

cmp al,0FFh

je PC\_

cmp al,0FEh

je PCXT\_

cmp al,0FBh

je PCXT\_

cmp al,0FCh

je AT\_

cmp al,0FAh

je PS2\_30\_

cmp al,0F8h

je PS2\_80\_

cmp al,0FDh

je PCjr\_

cmp al,0F9h

je PC\_Cnv\_

PC\_:

mov dx, OFFSET PC

jmp endl

PCXT\_:

mov dx, OFFSET PCXT

jmp endl

AT\_:

mov dx, OFFSET \_AT

jmp endl

PS2\_30\_:

mov dx, OFFSET PS2\_30

jmp endl

PS2\_80\_:

mov dx, OFFSET PS2\_80

jmp endl

PCjr\_:

mov dx, OFFSET PCjr

jmp endl

PC\_Cnv\_:

mov dx, OFFSET PC\_Cnv

jmp endl

endl:

call Write

ret

GetOStype ENDP

GetOSver PROC near

mov ax,0

mov ah,30h

int 21h

mov si,offset OS\_VERSION

add si,12

push ax

call BYTE\_TO\_DEC

pop ax

mov al,ah

add si,3

call BYTE\_TO\_DEC

mov dx,offset OS\_VERSION

call Write

mov si,offset OS\_OEM

add si,7

mov al,bh

call BYTE\_TO\_DEC

mov dx,offset OS\_OEM

call Write

mov dx,offset USER\_NUM

call Write

mov al,bl

call BYTE\_TO\_HEX

mov bx,ax

mov dl,bl

mov ah,02h

int 21h

mov dl,bh

int 21h

mov di,offset SPACE

add di,3

mov ax,cx

call WRD\_TO\_HEX

mov dx,offset SPACE

call Write

mov dx,offset ENDSTR

call Write

ret

GetOSver ENDP

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\_: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

BEGIN:

mov ax,DATA

mov ds,ax

call GetOStype

call GetOSver

xor AL,AL

mov AH,4Ch

int 21H

CODE ENDS

END BEGIN