**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по практической работе №2**

**по дисциплине «Операционные системы»**

**Тема**: **Исследование интерфейсов программных модулей**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8381 |  | Киреев К.А. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2020

**Цель работы.**

Исследование интерфейса управляющей программы и загрузочных модулей. Этот интерфейс состоит в передаче запускаемой программе управляющего блока, содержащего адреса и системные данные. Так загрузчик строит префикс сегмента программы (PSP) и помещает его адрес в сегментный регистр. Исследование префикса сегмента программы (PSP) и среды, передаваемой программе.

**Основные теоретические положения.**

При начальной загрузке программы формируется PSP, который размещается в начале первого сегмента программы. PSP занимает 256 байт и располагается с адреса, кратного границе сегмента. При загрузке модулей типа .COM все сегментные регистры указывают на адрес PSP. При загрузке модуля типа .EXE сегментные регистры DS и ES указывают на PSP. Именно по этой причине значения этих регистров в модуле .EXE следует переопределять.

Формат PSP:

Область среды содержит последовательность символьных строк вида: имя=параметр

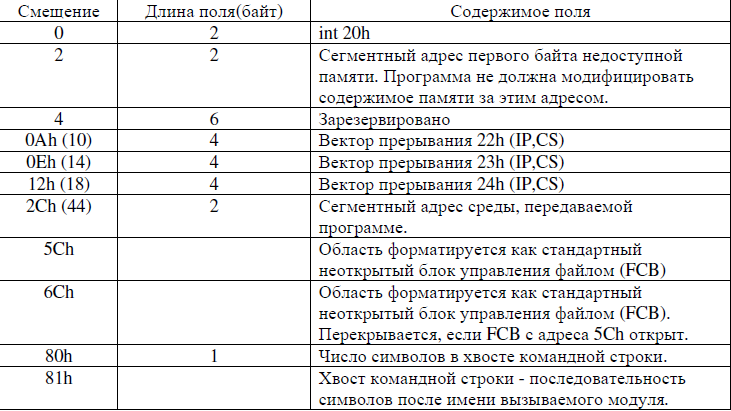
Каждая строка завершается байтом нулей.

В первой строке указывается имя COMSPEC, которая определяет используемый командный процессор и путь к COMMAND.COM. Следующие строки содержат информацию, задаваемую командами PATH, PROMPT, SET.

Среда заканчивается также байтом нулей. Таким образом, два нулевых байта являются признаком конца переменных среды. Затем идут два байта, содержащих 00h, 01h, после которых располагается маршрут загруженной

программы. Маршрут также заканчивается байтом 00h.

Таблица 1 – Формат PSP

****

**Постановка задачи.**

Требуется реализовать текст исходного .COM модуля, который определяет тип PC 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 переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться 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. Далее производится отладка полученного исходного модуля. Результатом выполнения этого действия будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля. Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить и сравнить исходные тексты для .COM и .EXE модулей, ответить на контрольные вопросы.

**Выполнение работы.**

Написан текст исходного .COM модуля, который определяет тип PC и версию системы. Полученный исходный модуль был отлажен. В результате был получен «плохой» .EXE модуль и построен «хороший» .COM модуль с помощью программы EXE2BIN.COM. Во время линковки получено предупреждение об отсутствии сегмента стека, представленное на рис. 1.

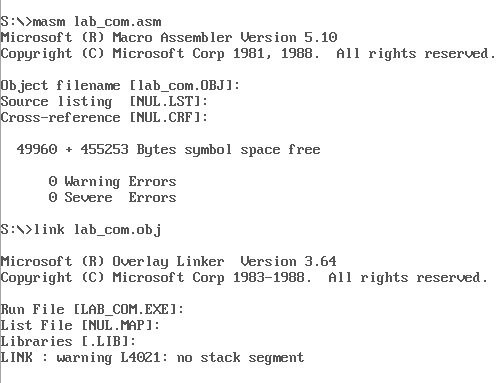


Рисунок 1 – Предупреждение об отсутствии стека

Запуск «хорошего» .COM модуля представлен на рис. 2.

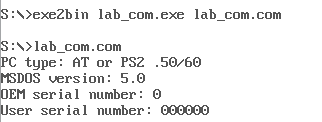


Рисунок 2 – «Хороший» .COM модуль

Запуск «плохого» .EXE модуля представлен на рис. 3.

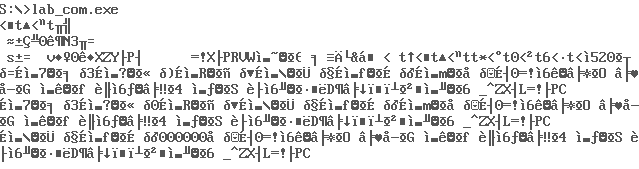


Рисунок 3 – «Плохой» .EXE модуль

Написан текст исходного .EXE модуля, который выполняет те же функции, что и предыдущий модуль. Был получен «хороший» .EXE модуль, запуск которого представлен на рис. 4.

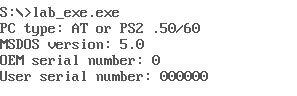


Рисунок 4 – «Хороший» .EXE модуль

**Отличия исходных текстов COM и EXE программ.**

**1. Сколько сегментов должна содержать COM-программа?**

Программы типа .СОМ состоят из единственного сегмента, в котором размещаются программные коды, данные и стек.

**2. EXE-программа?**

В программах типа .ЕХЕ для собственно программы, данных и стека предусматривают отдельные сегменты.

1. **Какие директивы должны обязательно быть в тексте COM программы?**

Директива ORG 100h, которая резервирует 256 байт для PSP. Заполнять PSP будет по-прежнему система, но место под него в начале сегмента должен отвести программист, код программы располагается только после этого блока.

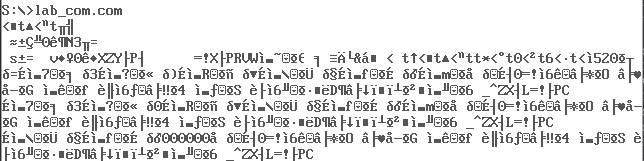


Рисунок 5 – .COM модуль без директивы 100h

Директива ASSUME, ставящая в соответствие адрес сегмента программы сегментам кода и данных.

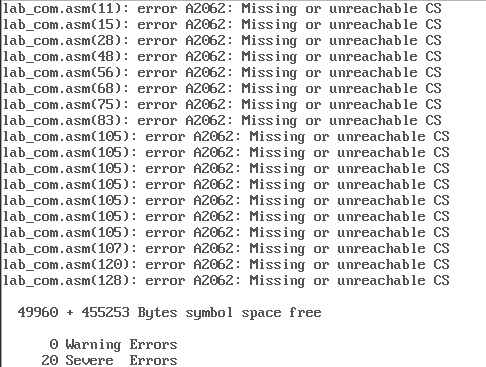


Рисунок 6 – .COM модуль без директивы ASSUME

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

Так как в COM-программе все сегментные регистры определяются в момент запуска программы, а не в момент компиляции (ассемблирования), то невозможно использование, например, таких конструкций:

mov ax, DATA или mov ax, CODE

Нельзя использовать команды вида mov <регистр>, seg <имя сегмента>

Например mov ax, seg CODE

**Отличия форматов файлов COM и EXE модулей.**

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

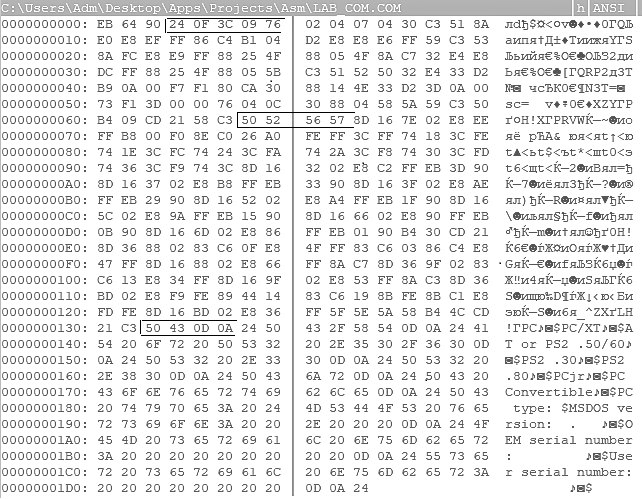


Рисунок 7 - .COM модуль в шестнадцатеричном виде

COM файл состоит из одного сегмента и, независимо от фактического размера программы, ей выделяется 64 Кбайт адресного пространства, Код начинается с адреса 0h, но при загрузке модуля устанавливается смещение в 100h.

Сначала идут байты, отвечающие за код. Например, коды команд, представленные на рис. 8. Данные начинаются на 130h, второй байт.

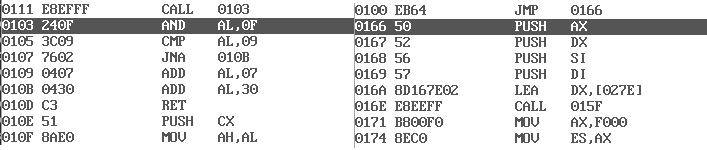


Рисунок 8 – команды .COM модуль

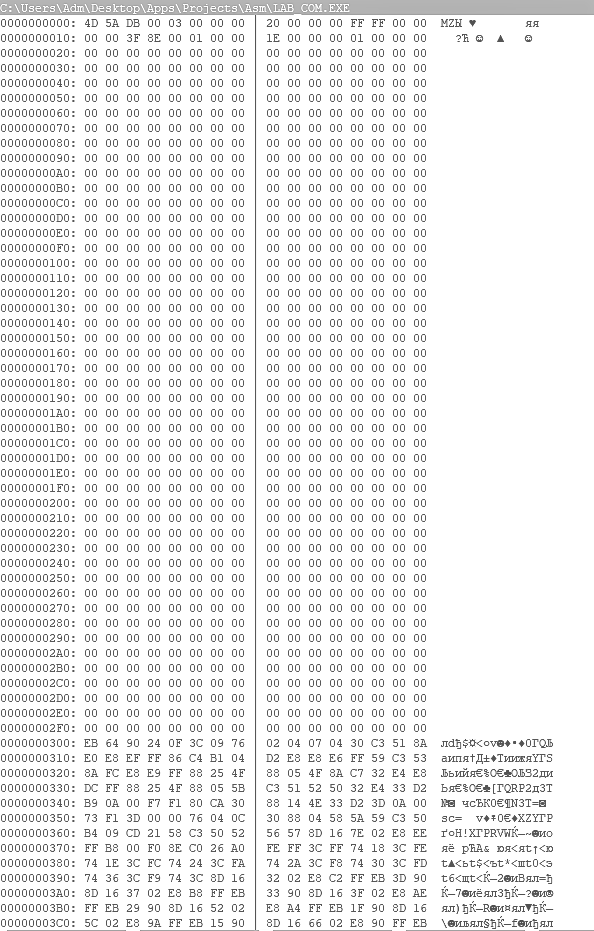


Рисунок 9 - «Плохой» .EXE модуль в шестнадцатеричном виде

1. **Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с 0 адреса?**

В «плохом» EXE файле данные и код содержатся в одном сегменте. Код располагается с адреса 300h. С адреса 0h располагается таблица разметки. Также 100h резервируются командой ORG 100h.

1. **Какова структура файла «хорошего» EXE? Чем он отличается от «плохого» EXE файла?**

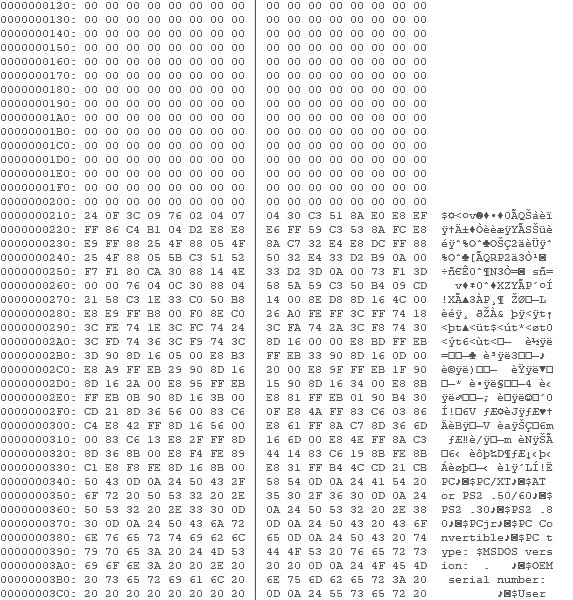


Рисунок 10 - «Хороший» .EXE модуль в шестнадцатеричном виде

В «хорошем» файле EXE содержится информация для загрузчика, сегмент стека, сегмент данных и сегмент кода. Код располагается с адреса 210h в отличие от 300h в «плохом» .EXE файле.

**Загрузка COM модуля в основную память.**

****

Рисунок 11 – Образ памяти программы типа .COM

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

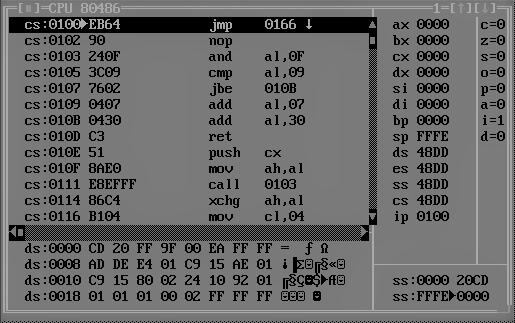


Рисунок 12 – Загрузка .COM модуля в память

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h, IP = 0100h.

1. **Что располагается с 0 адреса?**

С нулевого адреса располагается адрес начала PSP.

1. **Какие значения имеют сегментные регистры? На какие области памяти они указывают?**

Сегментные регистры имеют значение 48DDh. Они указывают на начало PSP, что продемонстрировано на рис. 11.

1. **Как определяется стек? Какую область памяти он занимает? Какие адреса?**

Стек определяется автоматически, указатель стека устанавливается на конец сегмента. Если для программы размер сегмента в 64КБ является достаточным, то DOS устанавливает в регистре SP адрес конца сегмента – FFFEh, что также можно увидеть на рис. 11. Адреса расположены в диапазоне от FFFEh до 0000h.

**Загрузка «хорошего» EXE модуля в основную память.**

****

Рисунок 13 – Образ памяти программы типа .EXE

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

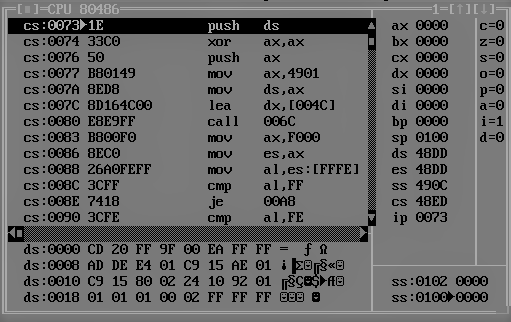


Рисунок 14 – Загрузка «хорошего».EXE модуля в память

Для PSP и программы выделяется блок памяти. Система, загрузив программу в память, инициализирует сегментные регистры, так что регистры DS и ES указывают на начало PSP(48DDh), CS - на начало сегмента команд(48EDh), a SS - на начало сегмента стека(490Ch). В указатель стека SP кладется смещение конца сегмента стека, а в указатель команд IP загружается смещение точки входа в программу. Таким образом, после загрузки программы в память адресуемыми оказываются все сегменты, кроме сегмента (или сегментов) данных.

1. **На что указывают регистры DS и ES?**

DS и ES указывают на начало PSP. Инициализация регистра DS в первых строках программы позволяет сделать адресуемым и сегмент данных.

1. **Как определяется стек?**

В исходном коде модуля стек определяется при помощи директивы .STACK, а при исполнении в регистр SS записывается адрес начала сегмента стека, а в указатель стека SP - смещение конца сегмента стека.

1. **Как определяется точка входа?**

При загрузке программы в указатель команд IP загружается смещение точки входа в программу (которое берется из операнда директивы END).

**Вывод.**

В ходе работы было проведено исследование различий в структурах исходных текстов модулей .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB\_COM.COM**

lab segment

assume cs:lab, ds:lab, es:nothing, ss:nothing

org 100h

main: jmp processing

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 16 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 цифра

ret

byte\_to\_hex endp

word\_to\_hex proc near

;перевод в 16 сс 16 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

xor ah, ah

call byte\_to\_hex

mov [di], ah

dec di

mov [di], al

pop bx

ret

word\_to\_hex endp

byte\_to\_dec proc near

;перевод в 10 сс, si - адрес поля младшей цифры

push cx

push dx

push ax

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 ax, 00h

jbe end\_l

or al, 30h

mov [si], al

end\_l:

pop ax

pop dx

pop cx

ret

byte\_to\_dec endp

print proc near

push ax

mov ah, 09h

int 21h

pop ax

ret

print endp

processing:

push ax

push dx

push si

push di

lea dx, pc\_arg

call print

mov ax, 0F000h ;указывает ES на ПЗУ

mov es, ax

mov al, es:[0FFFEh]

irpc case, FECA8D9

cmp al, 0F&case&h

je type\_&case&

endm

irpc met, FECA8D9

type\_&met&:

lea dx, pc\_&met&

call print

jmp OS

endm

OS:

mov ah, 30h

int 21h ;al - осноная версия, ah - модификация, bh - OEM, bl:cx - номер пользователя

lea si, os\_arg

add si, 15

call byte\_to\_dec ;пишется основная версия

add si, 3

xchg al, ah

call byte\_to\_dec ;пишется модификация

lea dx, os\_arg

call print

OEM:

mov al, bh

lea si, oem\_arg

add si, 19

call byte\_to\_dec

lea dx, oem\_arg

call print

serial\_number:

mov al, bl

lea si, user\_arg

call byte\_to\_hex

mov [si+20], ax

add si, 25

mov di, si ;ax - число, di - адрес последнего символа для word\_to\_hex

mov ax, cx

call word\_to\_hex

lea dx, user\_arg

call print

pop di

pop si

pop dx

pop ax

mov ah, 4ch

int 21h

ret

pc\_F db 'PC', 0dh, 0ah, '$'

pc\_E db 'PC/XT', 0dh, 0ah, '$'

pc\_C db 'AT or PS2 .50/60', 0dh, 0ah, '$'

pc\_A db 'PS2 .30', 0dh, 0ah, '$'

pc\_8 db 'PS2 .80', 0dh, 0ah, '$'

pc\_D db 'PCjr', 0dh, 0ah, '$'

pc\_9 db 'PC Convertible', 0dh, 0ah, '$'

pc\_arg db 'PC type: ', '$'

os\_arg db 'MSDOS version: . ', 0dh, 0ah, '$'

oem\_arg db 'OEM serial number: ', 0dh, 0ah, '$'

user\_arg db 'User serial number: ', 0dh, 0ah, '$'

lab ends

end main

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. LAB\_EXE.EXE**

dosseg

.model small

.stack 100h

.data

pc\_F db 'PC', 0dh, 0ah, '$'

pc\_E db 'PC/XT', 0dh, 0ah, '$'

pc\_C db 'AT or PS2 .50/60', 0dh, 0ah, '$'

pc\_A db 'PS2 .30', 0dh, 0ah, '$'

pc\_8 db 'PS2 .80', 0dh, 0ah, '$'

pc\_D db 'PCjr', 0dh, 0ah, '$'

pc\_9 db 'PC Convertible', 0dh, 0ah, '$'

pc\_arg db 'PC type: ', '$'

os\_arg db 'MSDOS version: . ', 0dh, 0ah, '$'

oem\_arg db 'OEM serial number: ', 0dh, 0ah, '$'

user\_arg db 'User serial number: ', 0dh, 0ah, '$'

.code

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 16 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 цифра

ret

byte\_to\_hex endp

word\_to\_hex proc near

;перевод в 16 сс 16 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

xor ah, ah

call byte\_to\_hex

mov [di], ah

dec di

mov [di], al

pop bx

ret

word\_to\_hex endp

byte\_to\_dec proc near

;перевод в 10 сс, si - адрес поля младшей цифры

push cx

push dx

push ax

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 ax, 00h

jbe end\_l

or al, 30h

mov [si], al

end\_l:

pop ax

pop dx

pop cx

ret

byte\_to\_dec endp

print proc near

push ax

mov ah, 09h

int 21h

pop ax

ret

print endp

processing proc far

push ds

xor ax, ax

push ax

mov ax, @data

mov ds, ax

lea dx, pc\_arg

call print

mov ax, 0F000h ;указывает ES на ПЗУ

mov es, ax

mov al, es:[0FFFEh]

irpc case, FECA8D9

cmp al, 0F&case&h

je type\_&case&

endm

irpc met, FECA8D9

type\_&met&:

lea dx, pc\_&met&

call print

jmp OS

endm

OS:

mov ah, 30h

int 21h ;al - осноная версия, ah - модификация, bh - OEM, bl:cx - номер пользователя

lea si, os\_arg

add si, 15

call byte\_to\_dec ;пишется основная версия

add si, 3

xchg al, ah

call byte\_to\_dec ;пишется модификация

lea dx, os\_arg

call print

OEM:

mov al, bh

lea si, oem\_arg

add si, 19

call byte\_to\_dec

lea dx, oem\_arg

call print

serial\_number:

mov al, bl

lea si, user\_arg

call byte\_to\_hex

mov [si+20], ax

add si, 25

mov di, si ;ax - число, di - адрес последнего символа для word\_to\_hex

mov ax, cx

call word\_to\_hex

lea dx, user\_arg

call print

mov ah, 4ch

int 21h

ret

processing endp

end processing