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

# ОТЧЕТ

**по лабораторной работе №6**

**по дисциплине «Операционные системы»**

**Тема: Построение модуля динамической структуры**

Студент гр. 8383 _____ Федоров И.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

**Цель работы.**

Исследование возможности построения загрузочного модуля динамической структуры, интерфейса между программой "родителем" и загружаемой программой "потомком" по управлению и данными.

**Ход работы.**

Был написан и отлажен программный модуль типа **.EXE**, который выполняет следующие функции:

1. Подготавливает параметры для запуска "потомка" из того же каталога.
2. Вызываемый модуль запускается с использованием загрузчика.
3. После запуска проверяется выполнение загрузчика и результат выполнения вызываемой программы.

Исходный код **.EXE** модуля приведен в приложении А. В качестве вызываемой программы была взята программа ЛР 2, которая была модифицирована: перед выходом она запрашивает ввод символа с клавиатуры, введенное значение записывается в регистр AL. Исходный код вызываемой программы приведен в приложении В.

Отлаженная программа была запущена, когда текущим каталогов является каталог с разработанными модулями. Вызываемый модуль ожидает ввода произвольного символа из числа A-Z. Результат работы программы представлен на рис. 1.

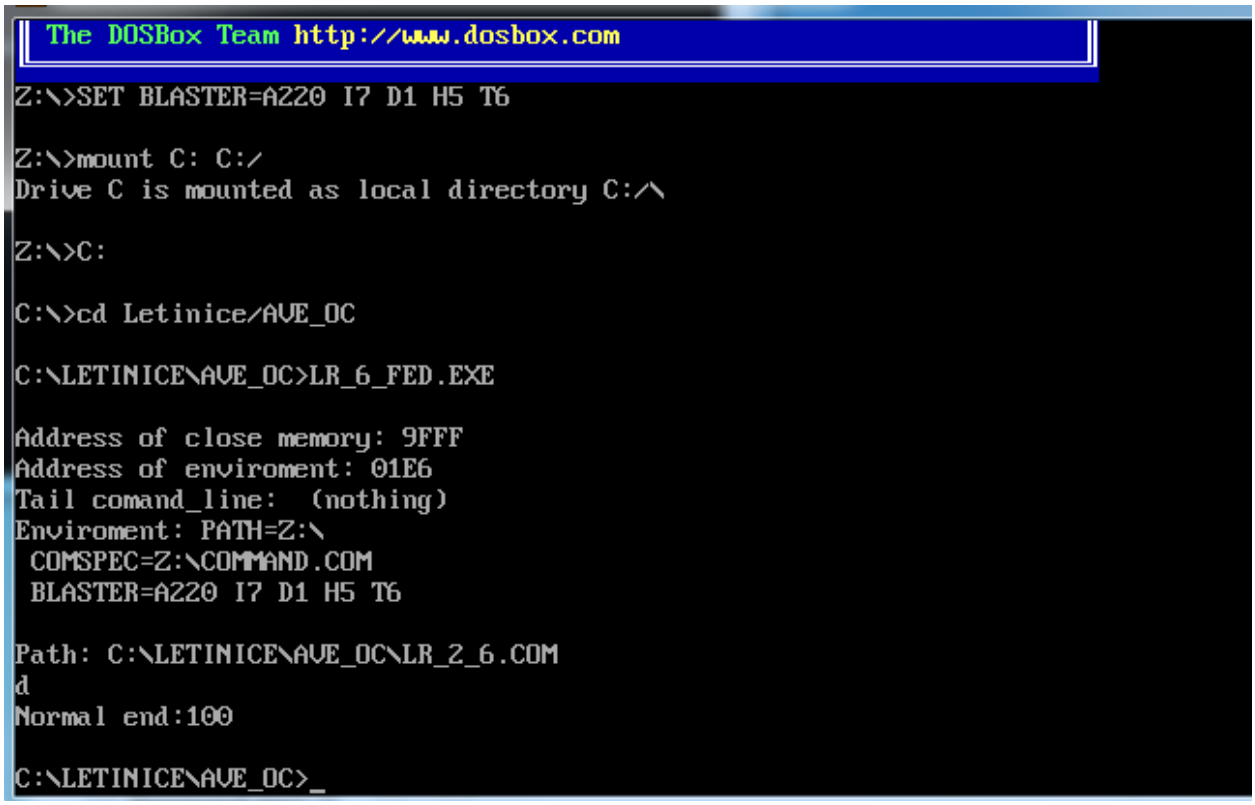


Рисунок 1 – Работа программы при вводе символа

Программа была запущена при таких же условиях (текущим является каталог с разработанными модулями), но теперь вместо символа была введена комбинация символов Ctrl-C. Результат работы программы представлен на рис. 2.


Рисунок 2 – Результат при вводе Ctrl-C

Программа была запущена при условии, что текущим является другой каталог, отличный от того, в котором содержатся разработанные модули. Результаты работы приведены на рис. 3.

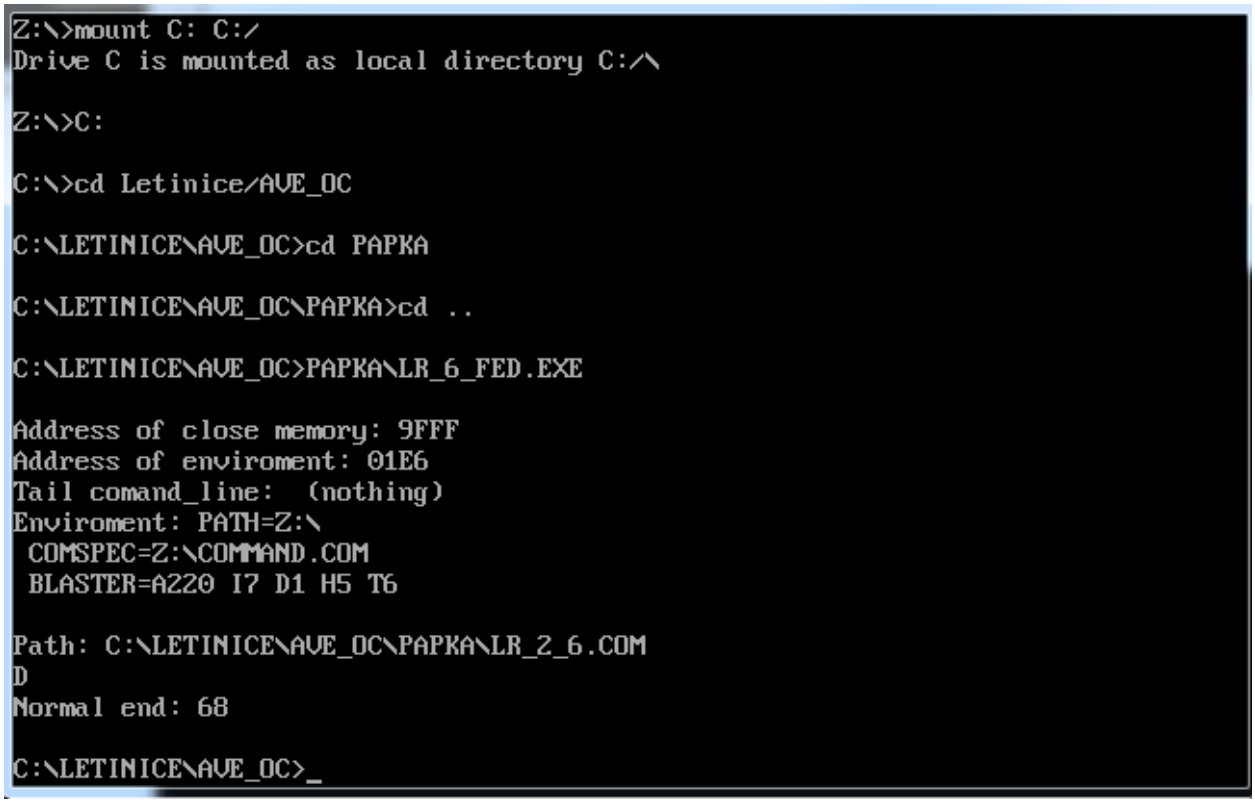


Рисунок 3 – Работа при вызове из другого каталога

Программа была запущена при условии, что разработанные модули находятся теперь в разных каталогах. На рис. 4. приведено расположение программы "родителя" и программы "потомка". Результат работы приведен на рис. 5.

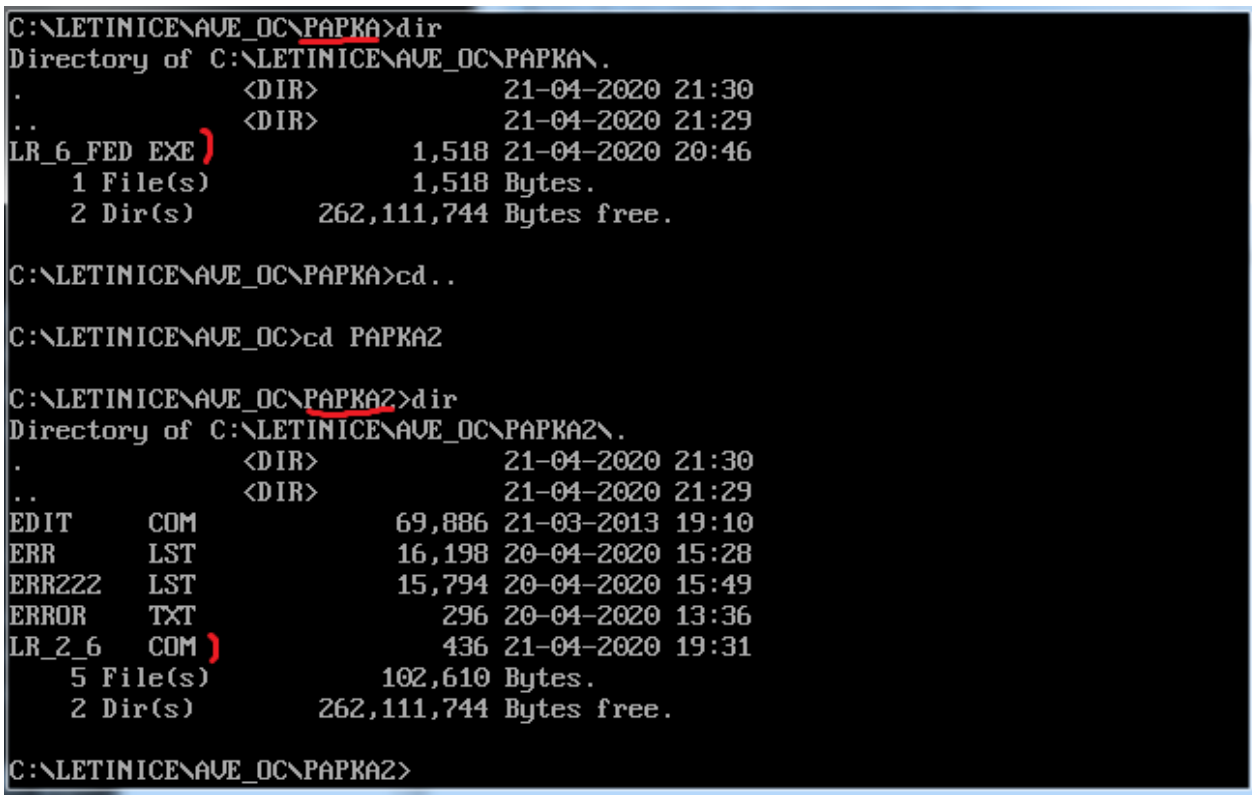


Рисунок 4 – Расположение файлов

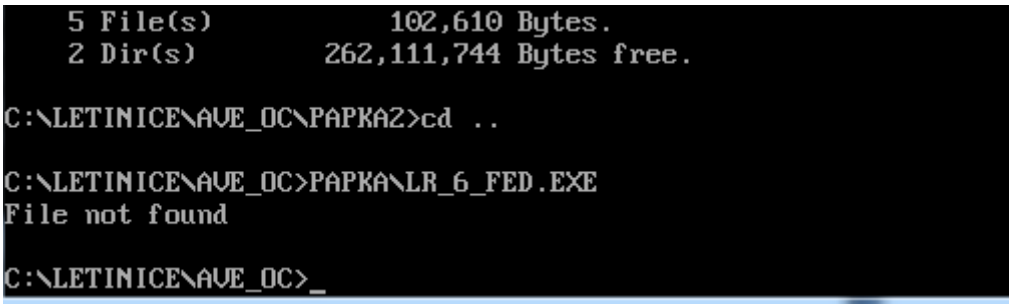


Рисунок 5 - Попытка вызова "дочерней" программы

**Ответы на контрольные вопросы.**

1) Как реализовано прерывание Ctrl-C?

Обработчиком является int 23H. При вводе Ctrl-C в кольцевой буфер клавиатуры поступает код 2E03h (в случае Ctrl-Break буфер очищается и в буферный байт драйвера клавиатуры кладется 03h). Функции DOS перед выполнением действия анализируют наличие кода 03h как в кольцевом буфере клавиатуры, так и в буфере драйвера. При обнаружении функция

DOS выполняет команду int 23h, обработчик которой завершает программу вызовом функции 4Ch.

2) В какой точке заканчивается вызываемая программа, если код причины завершения 0?

Т.к. это код нормального завершения, то в точке вызова функции завершения процесса с кодом возврата 4Ch прерывания 21h.

3) В какой точке заканчивается вызываемая программа по прерыванию Ctrl-C?

В точке выполнения функции считывания символа 01h прерывания int 21h, т.к. это при обнаружении Ctrl-C вызывается int 23h.

**Выводы.**

В ходе выполнения работы была исследована возможность построения модуля динамической структуры и интерфейс между программой "родителем" и программой "потомком".

# ПРИЛОЖЕНИЕ А
# КОД ПРОГРАММЫ

```
;ZSEG SEGMENT
;фиктивный сегмент
;ZSEG ENDS

; Программа "родителя"

DATA SEGMENT
    CODE_ db '    ', 13,10,'$'
       KEEP_SS DW 0  ;переменная для SS
    KEEP_SP DW 0  ;переменная для SP
       KEEP_PSP DW 0 ;переменная для PSP
       FILENAME_ DB 'LR_2_6.COM', 0
       FILEPATH DB 128 DUP(0)
       PARAMETERS DW 7 DUP(0)
       MEM_FLAG DB 1


  ; Load programm errors (CF = 1)
    LOAD_ERR_1 db 'Invalid function numver',13,10,'$'
       LOAD_ERR_2 db 'File not found',13,10,'$'
       LOAD_ERR_5 db 'Disk error',13,10,'$'
       LOAD_ERR_8 db 'Out of memory',13,10,'$'
       LOAD_ERR_10 db 'Invalid enviroment str',13,10,'$'
       LOAD_ERR_11 db 'Invalid format',13,10,'$'

  ; Succes load (CF = 0)
       NORMAL_CODE_0 db 13,10,'Normal end:$'
       NORMAL_CODE_1 db 'Ctrl-Break end',13,10,'$'
       NORMAL_CODE_2 db 'Device error',13,10,'$'
       NORMAL_CODE_3 db '  31H End',13,10,'$'

  ; Memory error
    MEMORY_ERR_7 db 'Memory block was destroyed',13,10,'$'
       MEMORY_ERR_8 db 'Out of memory to function',13,10,'$'
       MEMORY_ERR_9 db 'Invalid memory block`s address',13,10,'$'

       END_STR  EQU '$'

       dsize=$-CODE_        ;размер сегмента данных
DATA ENDS


       astack segment stack
       dw 64 dup(?)
       astack ends

       code segment
       assume CS:CODE, DS:DATA, ss:astack

CODES:

WriteMsg  PROC  NEAR
   push ax
   mov ah,09h
```

```
   int 21h
   pop ax
   ret
WriteMsg  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
;--------------------------------


;--------------------------------
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
```

```
;-------------------------------------------------------------------
PREPARE_MEMORY_SPACE PROC NEAR
       PUSH AX
       PUSH BX
       PUSH DX
    mov BX,((csize/16)+1)+256/16+((dsize/16)+1)+200/16  ; перевод в
параграфы /16
       MOV AH,4Ah                                          ; psp +
64*2 stack
    int 21h
    JC MEMORY_ERROR                                     ;проверяем на
ошибку   CF = 1
    MOV MEM_FLAG, 0
       JMP END_MEM_SPACE
MEMORY_ERROR:
       cmp ax, 7
       Jne else_if_1
       mov dx, offset MEMORY_ERR_7
       jmp print_err
else_if_1:
       cmp ax, 8
       jne else_if_2
       mov dx, offset MEMORY_ERR_8
       jmp print_err
else_if_2:
       mov dx, offset MEMORY_ERR_9
       jmp print_err
print_err:
    call WriteMsg
       mov MEM_FLAG, -1

END_MEM_SPACE:
       POP DX
       POP BX
       POP AX
       ret
PREPARE_MEMORY_SPACE ENDP
;-------------------------------------------------------------------



;-------------------------------------------------------------------
GET_FILEPATH PROC NEAR
       PUSH DX
       PUSH AX
       PUSH DI
       PUSH SI
       PUSH ES

       mov KEEP_PSP, es
       mov es, es:[2CH]
       xor si, si
while_not_path:
       mov ax, es:[si]
```

```
        inc si
        cmp ax, 0
        jne while_not_path
        inc si
        inc si
        inc si
        xor di, di

while_path:
        mov dl, es:[si]
        cmp dl, 0
        je rewrite_name
        mov FILEPATH[di], dl
        inc SI
        inc di
        jmp while_path

rewrite_name:
    dec di
    cmp FILEPATH[di], '\'
    je filename__
    jmp rewrite_name

filename__:
        inc di
        xor si, si

while_filename:
        mov dl,FILENAME_[si]
        mov FILEPATH[di], dl
        cmp dl, 0
        je end_filepath
        inc si
        inc di
        jmp while_filename

end_filepath:
        POP ES
        POP SI
        POP DI
        POP AX
        POP DX
        ret
GET_FILEPATH ENDP
;-------------------------------------------------------------------------



Main PROC FAR
        mov BX, DS
        mov AX, DATA
        mov DS, AX
        mov KEEP_PSP, BX
        call PREPARE_MEMORY_SPACE
        cmp MEM_FLAG, -1
        je END_MAIN1
        call GET_FILEPATH
```

```
       PUSH ES
       MOV KEEP_SP, SP
       MOV KEEP_SS, SS
    MOV BX, OFFSET PARAMETERS
    MOV DX, OFFSET FILEPATH     ;смещение - в DX
    MOV AX, SEG FILEPATH        ;сегмент - в DS
       MOV DS, AX
    MOV AH, 4BH                 ;функция EXEC
    MOV AL, 0                   ;выбираем "загрузку и запуск"
    INT 21H                     ;запускаем задачу
       MOV BX,AX
    MOV AX,DATA                 ;восстанавливаем DS
    MOV       DS,AX
       MOV AX,BX
    MOV SS,KEEP_SS              ;восстанавливаем SS
    MOV SP,KEEP_SP              ;восстанавливаем SP
       POP ES
       JC ERROR_LOAD
NORMAL_LOAD:
    mov AH, 4Dh
       int 21h
       cmp AH, 0
       jne normal_1
       mov DX, offset NORMAL_CODE_0
    call WriteMsg
    ;CBW
       mov dx, offset CODE_
    mov si, dx
    add si, 2
       call BYTE_TO_DEC
       call WriteMsg
       jmp END_MAIN1
  normal_1:
       cmp AH, 1
       jne normal_2
       mov DX, offset NORMAL_CODE_1
       call WriteMsg
       jmp END_MAIN1
  normal_2:
       cmp AH, 2
       jne normal_3
       mov DX, offset NORMAL_CODE_2
       call WriteMsg
       jmp END_MAIN1
  normal_3:
       cmp AH, 3
       mov DX, offset NORMAL_CODE_3
       call WriteMsg
       jmp END_MAIN1
END_MAIN1:
       jmp END_MAIN


ERROR_LOAD:
       cmp AX, 1
       jne else_2
       mov DX, offset LOAD_ERR_1
```

```
      jmp print_error
  else_2:
      cmp AX, 2
      jne else_5
      mov DX, offset LOAD_ERR_2
      jmp print_error
  else_5:
      cmp AX, 5
      jne else_8
      mov DX, offset LOAD_ERR_5
      jmp print_error
  else_8:
      cmp AX, 8
      jne else_10
      mov DX, offset LOAD_ERR_8
      jmp print_error
  else_10:
      cmp AX, 10
      jne else_11
      mov DX, offset LOAD_ERR_10
      jmp print_error
  else_11:
      cmp AX, 11
      mov DX, offset LOAD_ERR_11
      jmp print_error
  print_error:
    call WriteMsg
END_MAIN:
    mov AH, 4Ch
       int 21h
csize=$-CODES
Main ENDP
CODE ENDS
;ZSEG SEGMENT ;фиктивный сегмент
;ZSEG ENDS
END MAIN
```

## ПРИЛОЖЕНИЕ В

## КОД ВЫЗЫВАЕМОЙ ПРОГРАММЫ

```
TESTPC      SEGMENT
            ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
                 ORG 100H    ;обязательно!
START:      JMP MAIN

LEN_CLOSE_MEM  EQU 30
LEN_ENV_SEG EQU 28
;TAIL db 83 DUP(?)
STR_CLOSE_MEM db 13,10, "Address of close memory:             $"
STR_ENV_SEG db 13,10, "Address of enviroment:             $"
STR_TAIL db 13,10, "Tail comand_line: $"
STR_EMPTY_TAIL db " (nothing) $"
STR_ENVIROMENT_AREA db 13,10, "Enviroment: $"
STR_ENTER db 13,10, " $"
STR_PATH db 13,10, "Path: $"
```

```
ENTER_STR db 13,10, '$'
;ПРОЦЕУРЫ
;------------------------------

WRITE_STR PROC near
      push ax
      mov ah, 09h
      int 21h
      pop ax
      ret
WRITE_STR 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
; 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
;--------------------------------------------------------------------
--------
;--------------------------------------------------------------------
--------
; Funct lab_2

PRINT_ADDRESS_CLOSE_MEM PROC near
   push ax
   push dx
   mov ax, ds:[02h]                    ;в PSP
   mov di, offset STR_CLOSE_MEM
   add di, LEN_CLOSE_MEM
   call WRD_TO_HEX
   mov dx, offset STR_CLOSE_MEM
   call WRITE_STR
   pop dx
   pop ax
   ret
PRINT_ADDRESS_CLOSE_MEM ENDP


PRINT_ADDRESS_ENVIROMENT  PROC near
   push ax
   push dx
   mov ax, ds:[2Ch]               ;(44)
   mov di, offset STR_ENV_SEG
   add di, LEN_ENV_SEG
   call WRD_TO_HEX
   mov dx, offset STR_ENV_SEG
   call WRITE_STR
```

```
   pop dx
   pop ax
   ret
PRINT_ADDRESS_ENVIROMENT ENDP



PRINT_TAIL PROC near
   push ax
   push dx
   mov dx, offset STR_TAIL
   call WRITE_STR
   mov cx, 0
   mov cl, ds:[80h]        ;число символов в хвосте
   cmp cl, 0
   je tail_empty
   mov di, 0
   xor dx,dx
 print_tail_cycle:
   mov dl, ds:[81h+di]   ;ds+81h+di сделать потом
   mov ah,02H
   int 21h
   inc di
   loop print_tail_cycle
   jmp end_print
tail_empty:
   mov dx, offset STR_EMPTY_TAIL
   call WRITE_STR
end_print:
   pop dx
   pop ax
   ret
PRINT_TAIL ENDP



PRINT_PATH_ENVIROMENT PROC near
   push dx
   push ax
   push ds
   mov dx, offset STR_ENVIROMENT_AREA
   call WRITE_STR
   mov di, 0
   mov es, ds:[2Ch]
cycle_env:
   cmp byte ptr es:[di], 00h            ;mov dl, ds:[di] и
сранивать dl  с 0
   je enter_                        ;==
   mov dl, es:[di]
   mov ah, 02h
   int 21h
   inc di
   jmp cycle_env
```

```
enter_:
   inc di
   cmp word ptr es:[di], 0001h
   je path_
   mov dx, offset STR_ENTER
   call WRITE_STR
   jmp cycle_env
path_:
   inc di
   inc di
   mov DX, offset STR_PATH
   call WRITE_STR
cycle_p:
   cmp byte ptr es:[di], 00h
   je end_print_p
   mov dl, es:[di]
   mov ah, 02h
   int 21h
   inc di
   jmp cycle_p
end_print_p:
   mov dx, offset ENTER_STR
   call WRITE_STR
   pop dx
   pop ax
   pop ds
   ret
PRINT_PATH_ENVIROMENT ENDP



MAIN:
   call PRINT_ADDRESS_CLOSE_MEM
   call PRINT_ADDRESS_ENVIROMENT
   call PRINT_TAIL
   call PRINT_PATH_ENVIROMENT

   xor al, al
   mov ah, 01h
   int 21h
   mov AH,4Ch
   int 21H
TESTPC ENDS
END START
```