

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

ОТЧЕТ
по практической работе № 3
по дисциплине «Операционные системы»
Тема: Исследование организации управления основной памятью

Студент гр. 8383

Степанов В.Д.

Преподаватель

Губкин А.Ф.

Санкт-Петербург

2020

Цель работы.

Исследовать структуры данных и работа функций управления памятью ядра операционной системы.

Выполнение работы.

1. Был написан и отлажен программный модуль типа .COM, который выбирает и распечатывает следующую информацию:

- 1) Количество доступной памяти.
- 2) Размер расширенной памяти.
- 3) Выходит цепочку блоков управления памятью.

Адреса при выводе переставляются шестнадцатичными числами. Объем памяти функциями управления памятью выводится в параграфах. Пример вывода программы 1 приведен на рисунке 1.

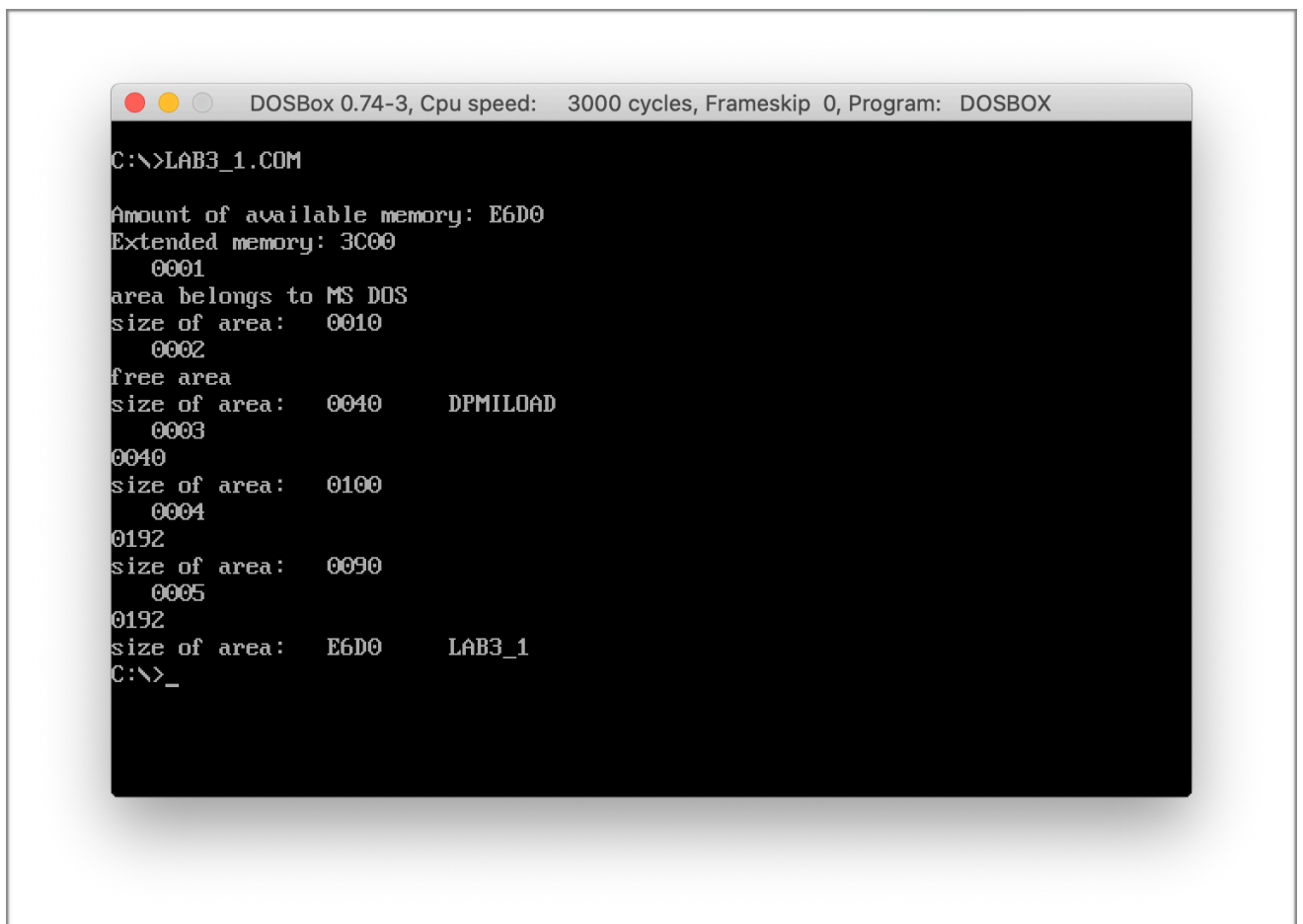


Рисунок 1 – Вывод программы 1

2. Программа была изменена так, чтобы она освобождала память, которую не занимает. Вывод программы 2 представлен на рисунке 2. Из рисунков 1 и 2 видно, что освобожденная память относится к шестому блоку управления памятью.

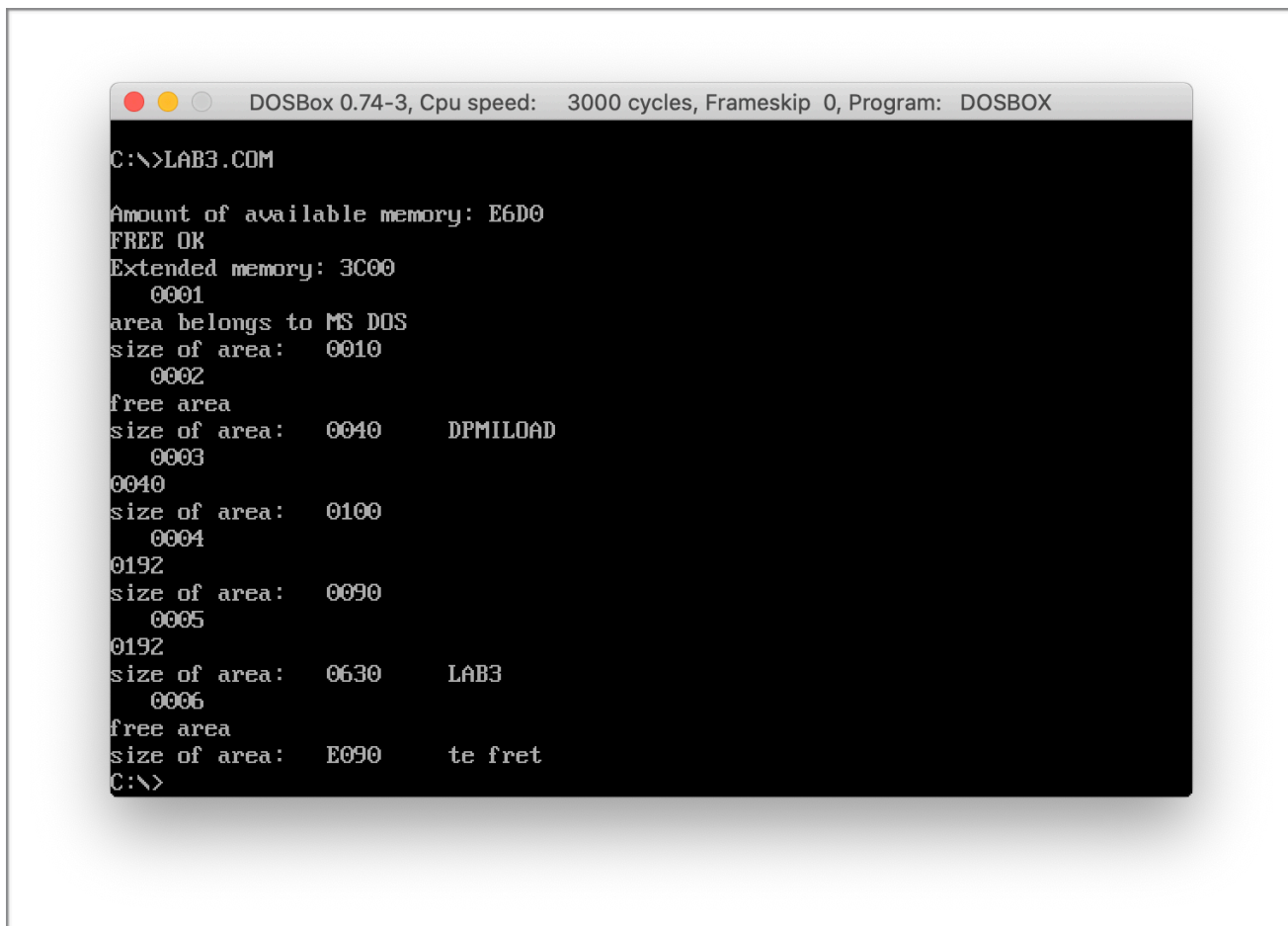


Рисунок 2 – Вывод программы 2

3. Программа была изменена так, чтобы после освобождения памяти, она запрашивала 64Кб памяти функцией 48H прерывания 21h. Вывод программы 3 приведен на рисунках 3 и 4. Из рисунков 1 – 4 видно, что выделенная память относится к шестому блоку, а освобожденная к седьмому блоку управления памятью.

4. Программа была изменена так, чтобы она запрашивала выделение памяти до освобождения. Вывод программы 4 представлен на рисунках 5. Из рисунков 1 – 5 видно, память не была выделена.

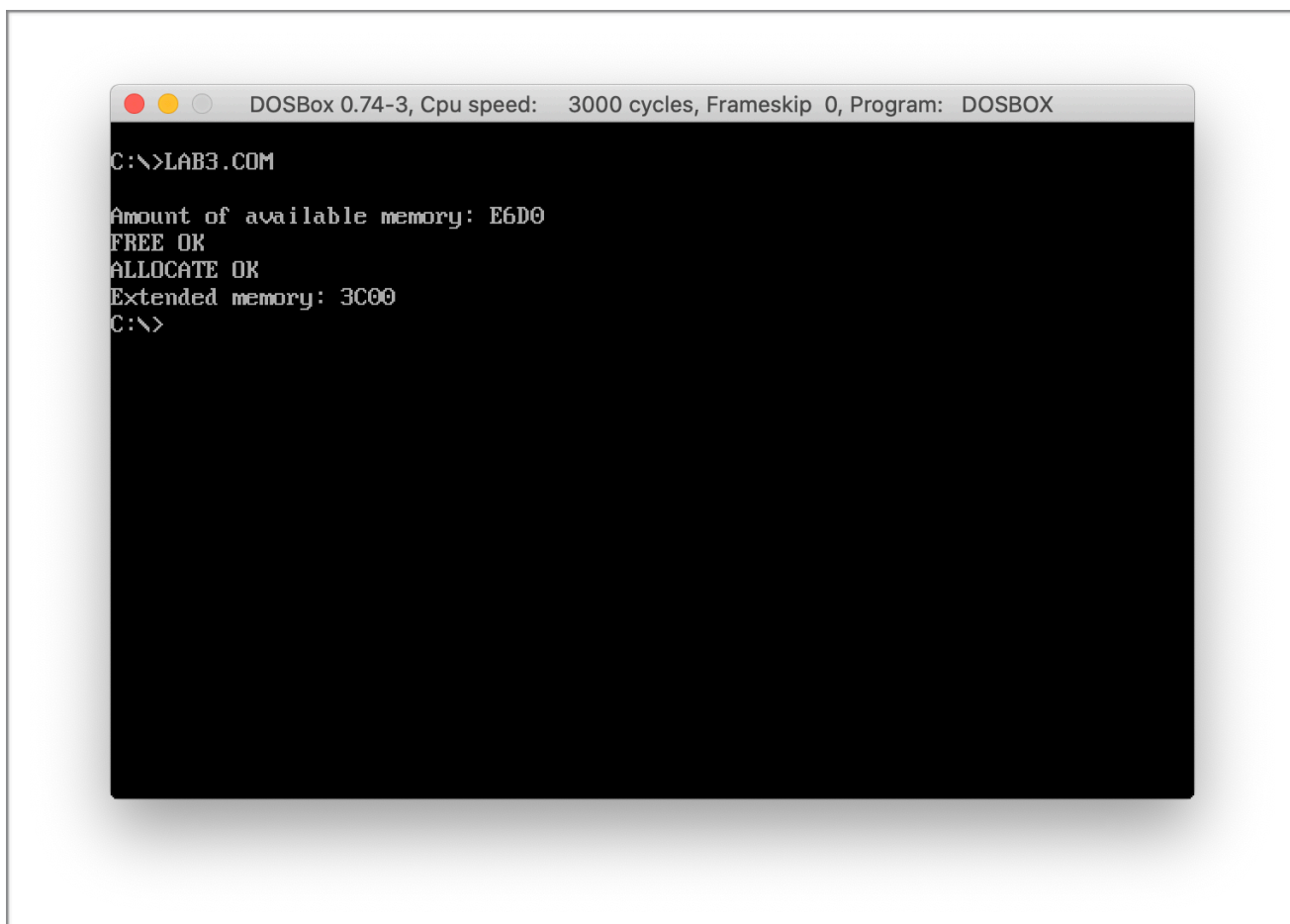


Рисунок 3 – Вывод программы 3 без блоков памяти

Контрольные вопросы.

- 1) Что означает “доступный объем памяти”?

Память доступная для выполнения программы.

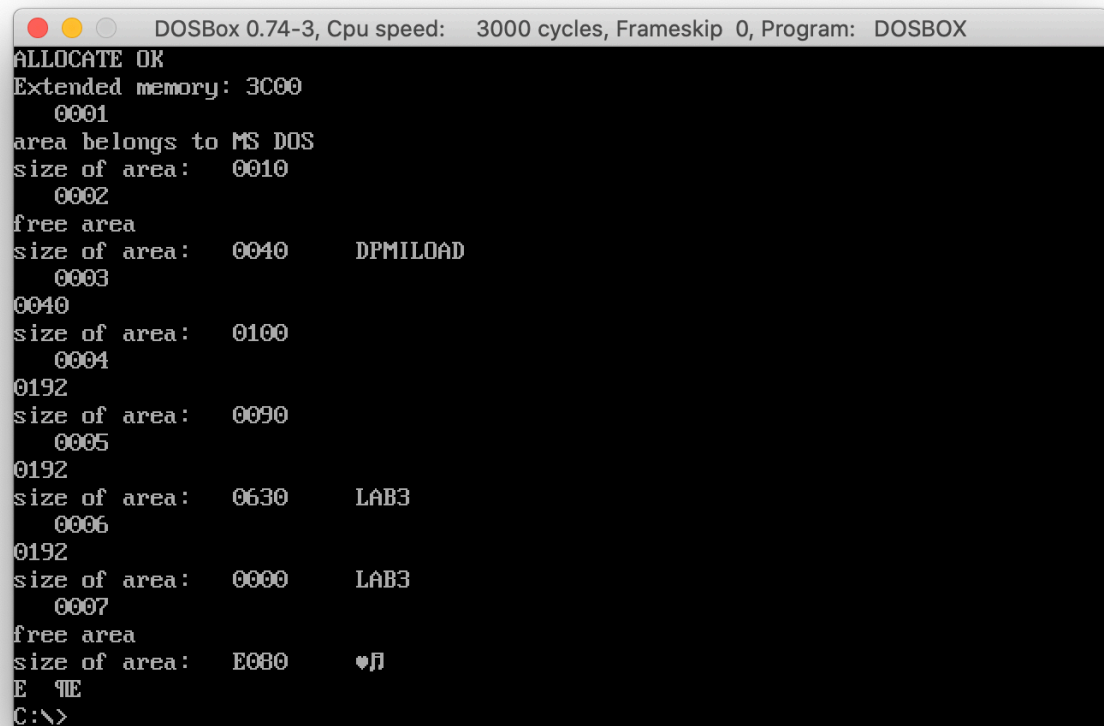
- 2) Где MCB блок Вашей программы в списке?

Слово после первого байта mcb указывает на того, кому принадлежит участок памяти, mcb блоки программы указывают на один адрес PSP владельца. В первой программе видно, что блоки 4 и 5 имеют одинаковый адрес владельца, а так же в 5 блоке имеется часть названия программы. Из этого можно сделать вывод, что они принадлежат программе. Аналогично в программах 2 и 4. В 3 программе еще блок 6.

- 3) Какой размер памяти она занимает в каждом случае?

Программа 1 – E760h (59232) байт.

Программа 2 – 06C0h (1728) байт.

A screenshot of a DOSBox window titled "DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX". The window displays the output of a memory allocation program. The text is as follows:

```
ALLOCATE OK
Extended memory: 3C00
0001
area belongs to MS DOS
size of area: 0010
0002
free area
size of area: 0040      DPMILOAD
0003
0040
size of area: 0100
0004
0192
size of area: 0090
0005
0192
size of area: 0630      LAB3
0006
0192
size of area: 0000      LAB3
0007
free area
size of area: E080      ♥♣
E 7E
C:\>_
```

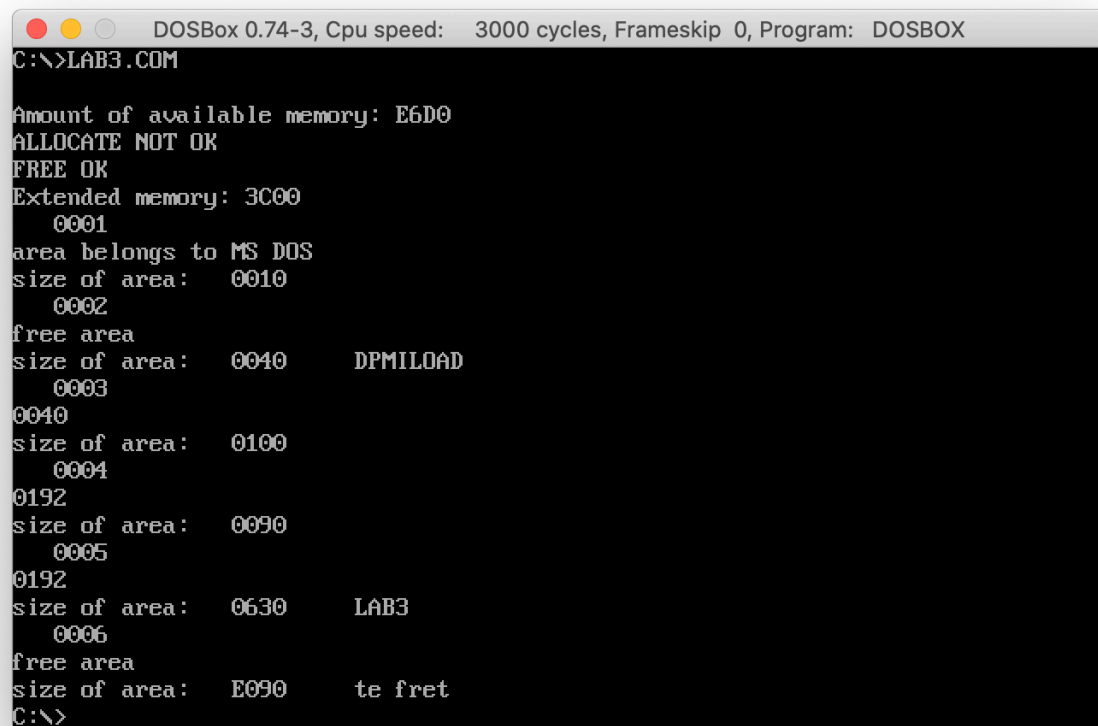
Рисунок 4 – Вывод программы 3 с выводом блоков памяти

Программа 3 – 06C0h (1728)байт + 64кб память выделения для программы.

Программа 4 – 06C0h (1728) байт.

Выводы.

В ходе лабораторной работы были исследованы структуры данных и работа функций управления памятью ядра операционной системы.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>LAB3.COM

Amount of available memory: E6D0
ALLOCATE NOT OK
FREE OK
Extended memory: 3C00
  0001
area belongs to MS DOS
size of area:  0010
  0002
free area
size of area:  0040    DPMILOAD
  0003
0040
size of area:  0100
  0004
0192
size of area:  0090
  0005
0192
size of area:  0630    LAB3
  0006
free area
size of area:  E090    te fret
C:\>_
```

Рисунок 5 – Вывод программы 4

ПРИЛОЖЕНИЕ А

КОД ДЛЯ .COM МОДУЛЯ

```
LAB1  SEGMENT
      ASSUME  CS:LAB1, DS:LAB1, ES:NOTHING, SS:NOTHING
      ORG 100H

      START: JMP BEGIN

;-----
TYPE_OF_PC db "Type of PC: $"
PC db "PC$"
PC_XT db "PC/XT$"
AT db "AT$"
PS2_30 db "PS2 model 30$"
PS2_50 db "PS2 model 50 or 60$"
PS2_80 db "PS2 model 80$"
PCjr db "PCjr$"
PC_Con db "PC convertible$"
UNKNOWN_TYPE db " Unknown$"
OUT_UNKNOWN db "    $"
VER_NUM db 13, 10, "Version number: $"

OUT_VER_NUM db "    .  $"

OEM_NUM db 13, 10, "OEM: $"
OUT_OEM db "        $"
SER_NUM db 13, 10, "Serial number: $"
OUT_SER_NUM DB "                $"
;-----

TYPE_PC PROC near

    push ax
    push dx

    mov dx, offset TYPE_OF_PC
    call PRINT

    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEH]

    mov al, 7h
```

```

    cmp al, 0FFh
        je W_PC
    cmp al, 0FEh
        je W_PC_XT
    cmp al, 0FBh
        je W_PC_XT
    cmp al, 0FCh
        je W_AT
    cmp al, 0FAh
        je W_PS2_30
    cmp al, 0FCh
        je W_PS2_50
    cmp al, 0F8h
        je W_PS2_80
    cmp al, 0FDh
        je W_PCjr
    cmp al, 0F9h
        je W_PC_Con

    mov di, offset OUT_UNKNOWN
    call BYTE_TO_HEX
    mov [di], ax
    mov dx, offset OUT_UNKNOWN
    CALL PRINT

    mov DX, offset UNKNOWN_TYPE
    jmp ESC_PROC

W_PC:
    mov DX, offset PC
    jmp ESC_PROC

W_PC_XT:
    mov DX, offset PC_XT
    jmp ESC_PROC

W_AT:
    mov DX, offset AT
    jmp ESC_PROC

W_PS2_30:

```



```

        mov DX, offset PS2_30
        jmp ESC_PROC

W_PS2_50:
        mov DX, offset PS2_50
        jmp ESC_PROC

W_PS2_80:
        mov DX, offset PS2_80
        jmp ESC_PROC

W_PCjr:
        mov DX, offset PCjr
        jmp ESC_PROC

W_PC_Con:
        mov DX, offset PC_Con

ESC_PROC:
        call PRINT

pop dx
pop ax

ret
TYPE_PC ENDP

;-----

PRINT PROC near

push ax
sub ax, ax
mov ah, 9h
int 21h
pop ax

ret
PRINT 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

```

;-----

```

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 ;â AL ñòàðøàÿ öèôðà
        pop     CX          ;â AH ìèääøàÿ
        ret
BYTE_TO_HEX    ENDP

```

;-----

```

BYTE_TO_DEC    PROC    near

                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_1
                or       AL,30h
                mov      [SI],AL

end_1:          pop      DX
                pop      CX
                ret

BYTE_TO_DEC    ENDP

```

;-----

```

SYSTEM_VERSION PROC

    push ax
    push bx
    push cx
    push dx

    sub ax, ax
    mov ah, 30h
    int 21h

    mov dx, offset VER_NUM
    call PRINT

    push ax
    mov si, offset OUT_VER_NUM
    inc si
    call BYTE_TO_DEC

```

```

pop ax
mov al, ah
add si, 3
call BYTE_TO_DEC

mov dx, offset OUT_VER_NUM
call print

mov DX, offset OEM_NUM
call PRINT

mov si, offset OUT_OEM
add si, 2
mov al, bh
call BYTE_TO_DEC
mov dx, OFFSET OUT_OEM
call PRINT

mov dx, offset SER_NUM
call PRINT

mov di, offset OUT_SER_NUM
add di, 6
mov ax, cx

call WRD_TO_HEX
mov al, bl
call BYTE_TO_HEX
sub di, 2
mov [di], ax
mov dx, offset OUT_SER_NUM
CALL PRINT

pop DX
pop CX
pop BX
pop AX
ret
SYSTEM_VERSION ENDP

```

```

;-----

```

```
PRINT_NUMBER_DEC PROC
```

```
    push AX
```

```
    push BX
```

```
    sub BX, BX
```

```
    mov Bl, 10
```

```
    mov AH, 0
```

```
    div Bl
```

```
    mov DX, AX
```

```
    add DL, '0'
```

```
    sub AX, AX
```

```
    mov AH, 02h
```

```
    int 21h
```

```
    mov DL, DH
```

```
    add DL, '0'
```

```
    sub AX, AX
```

```
    mov AH, 02h
```

```
    int 21h
```

```
    pop BX
```

```
    pop AX
```

```
    ret
```

```
PRINT_NUMBER_DEC ENDP
```

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

```
BEGIN:
```

```
    call TYPE_PC
```

```
    call SYSTEM_VERSION
```

```
    xor AX, AX
```

```
    mov AH, 4Ch
```

```
    int 21h
```

```
LAB1 ENDS
```

```
END START
```

ПРИЛОЖЕНИЕ Б

КОД ДЛЯ «ХОРОШЕГО» .EXE МОДУЛЯ

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
    AMOUNT_MEM db 13, 10, "Amount of available memory:      $"
    EX_ME db 13, 10, "Extended memory:                      $"
    OUT_STR db 13, 10, "                                     $"
    NUM db 13, 10, "                                     $"
    ENDL db 13, 10, '$'
    ST_1 db 13, 10, "free area$"
    ST_2 db "OS XMS UMB$"
    ST_3 db "driver's top memory$"
    ST_4 db "MS DOS$"
    ST_5 db "control block 386MAX UMB$"
    ST_6 db "blocked 386MAX$"
    ST_7 db "386MAX UMB$"

    AREA_BELONGS db 13, 10, "area belongs to $"
    SIZE_OF_AREA db 13, 10, "size of area: $"
    SIZE DB "                                     $"

    FREE_OK db 13, 10, "FREE OK$"
    FREE_NOT_OK db 13, 10, "FREE NOT OK$"

    ALLOC_OK db 13, 10, "ALLOCATE OK$"
    ALLOC_NOT_OK db 13, 10, "ALLOCATE NOT OK"

;-----

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 ; т AL өГрӨİр IшЄӨр
pop CX ;т AH - ьырфİр
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_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_1
    or AL,30h
    mov [SI],AL
end_1: pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
;-----

PRINT PROC near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT ENDP
;-----

AVAIL_MEMORY PROC near
    push ax
    push bx
    push dx

    mov di, offset AMOUNT_MEM
    add di, 33
    mov ah, 4Ah
    mov bx, 0FFFFh
    int 21h
    mov ax, bx
    mov bx, 10h
    mul bx
    call WRD_TO_HEX
    mov dx, offset AMOUNT_MEM
    call PRINT

    pop dx
    pop bx
    pop ax
    ret
AVAIL_MEMORY ENDP

```



```

;-----

LENGTH_OF_EXTENDED_MEMPRY PROC near

PUSH AX
PUSH BX
PUSH DX

    xor ax, ax
    mov AL, 30h

    out 70h, AL
    in AL, 71h
    mov BL, AL
    mov AL, 31h
    out 70h, AL
    in AL, 71h

    mov bh, ah
    mov ah, al
    mov al, bh

    mov di, offset EX_ME
    add di, 22
    call WRD_TO_HEX
    mov dx, offset EX_ME
    call PRINT

POP DX
POP BX
POP AX

    RET
LENGTH_OF_EXTENDED_MEMPRY ENDP

;-----

```

```

OUT_MEMORY_BLOCKS PROC near

```

```

    push ax

```

```

push bx
push dx

    mov ah, 52h
    int 21h
    mov ax, es:[bx-2]
    mov es, ax
    xor cx, cx
NEXT_1:

    inc cx
    mov ax, cx

    mov di, offset NUM
    add di, 8
    call WRD_TO_HEX
    mov dx, offset NUM
    call PRINT

    push cx
    xor ax, ax
    mov al, es:[0h]
    push ax
    mov ax, es:[1h]

    mov dx, offset AREA_BELONGS

    cmp ax, 0h
    je M_ST_1
    cmp ax, 6h
    je M_ST_2
    cmp ax, 7h
    je M_ST_3
    cmp ax, 8h
    je M_ST_4
    cmp ax, 0FFFAh
    je M_ST_5
    cmp ax, 0FFFDh
    je M_ST_6
    cmp ax, 0FFFEh
    je M_ST_7

    mov di, offset OUT_STR

```

```

    add di, 5
    call WRD_TO_HEX
    mov dx, offset OUT_STR
    call PRINT
    jmp M_END

M_ST_1:
    mov dx, offset ST_1
    jmp M_PR;
M_ST_2:
    call PRINT
    mov dx, offset ST_2
    jmp M_PR;
M_ST_3:
    call PRINT
    mov dx, offset ST_3
    jmp M_PR;
M_ST_4:
    call PRINT
    mov dx, offset ST_4
    jmp M_PR;
M_ST_5:
    call PRINT
    mov dx, offset ST_5
    jmp M_PR;
M_ST_6:
    call PRINT
    mov dx, offset ST_6
    jmp M_PR;
M_ST_7:
    call PRINT
    mov dx, offset ST_7
    jmp M_PR;

M_PR:
    call PRINT

M_END:
    mov ax, es:[3h]
    mov bx, 10h
    mul bx

```

```

mov dx, offset SIZE_OF_AREA
call PRINT

mov di, offset SIZE
add di, 5
call WRD_TO_HEX
mov dx, offset SIZE
call PRINT

mov cx, 8
xor si, si

M_LOOP:
mov dl, es:[si+8h]
mov ah, 02h
int 21h
inc si
loop M_LOOP

mov ax, es:[3h]
mov bx, es
add bx, ax
inc bx
mov es, bx
pop ax
pop cx
cmp al, 5Ah
je END_P
jmp NEXT_1

END_P:

pop dx
pop bx
pop ax

ret
OUT_MEMORY_BLOCKS ENDP

;-----

```

```

FREE_MEM PROC near

push ax
push bx
push cx

mov bx, offset END_PROG
add bx, 10Fh
shr BX, 4
mov ah, 4Ah
int 21h
jnc OK
mov dx, offset FREE_NOT_OK
jmp END_F

OK:
    mov dx, offset FREE_OK

END_F:
    call PRINT

pop cx
pop bx
pop ax

ret
FREE_MEM ENDP
;-----

ALLOCATE_MEM PROC near
push ax
push bx
push dx

mov bx, 1000h
mov ah, 48h
int 21h
jnc AL_OK
mov dx, offset ALLOC_NOT_OK
jmp END_A

```

```

AL_OK:
    mov dx, offset ALLOC_OK
END_A:

    call PRINT

pop dx
pop bx
pop ax
ret
ALLOCATE_MEM ENDP
;-----

BEGIN:

    call AVAIL_MEMORY
        call ALLOCATE_MEM
        call FREE_MEM
    call LENGTH_OF_EXTENDED_MEMPRY
    call OUT_MEMORY_BLOCKS

    xor AL, AL
    mov AH, 4Ch
    int 21h

    PROG_FREE:
        DW 128 dup(0)
    END_PROG:
TESTPC ENDS
    END START

```