

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

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

Студент гр. 0382

\_\_\_\_\_

Бочаров Г.С.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2022

## **Цель работы.**

Для исследования организации управления памятью необходимо ориентироваться на тип основной памяти, реализованный в компьютере и способ организации, принятый в ОС. В лабораторной работе рассматривается не страничная память и способ управления динамическими разделами. Для реализации управления памятью в этом случае строится список занятых и свободных участков памяти. Функции ядра, обеспечивающие управление основной памятью, предусматривают и преобразуют этот список.

В лабораторной работе исследуются структуры данных и работа функций управления памятью ядра операционной системы.

## **Постановка задачи.**

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .COM, который выбирает и распечатывает следующую информацию:

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

Адреса при выводе представляются шестнадцатеричными числами. Объем памяти функциями управления памятью выводится в параграфах. Необходимо преобразовать его в байты и выводить в виде десятичных чисел. Последние восемь байт МСВ выводятся как символы, не следует преобразовывать их в шестнадцатеричные числа.

Запустите программу и внимательно оцените результаты. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

**Шаг 2.** Измените программу таким образом, чтобы она освобождала память, которую она не занимает. Для этого используйте функцию 4Ah прерывания 21h (пример в разделе «Использование функции 4AH»). Повторите эксперимент, запустив модифицированную программу.

Сравните выходные данные с результатами, полученными на предыдущем шаге. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

**Шаг 3.** Измените программу еще раз таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти функцией 48H прерывания 21H. Повторите эксперимент, запустив модифицированную программу. Сравните выходные данные с результатами, полученными на предыдущих шагах. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

**Шаг 4.** Измените первоначальный вариант программы, запросив 64Кб памяти функцией 48H прерывания 21H до освобождения памяти. Обязательно обрабатывайте завершение функций ядра, проверяя флаг CF. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

**Шаг 5.** Оцените результаты, полученные на предыдущих шагах. Ответьте на контрольные вопросы и оформите отчет

### **Используемые функции**

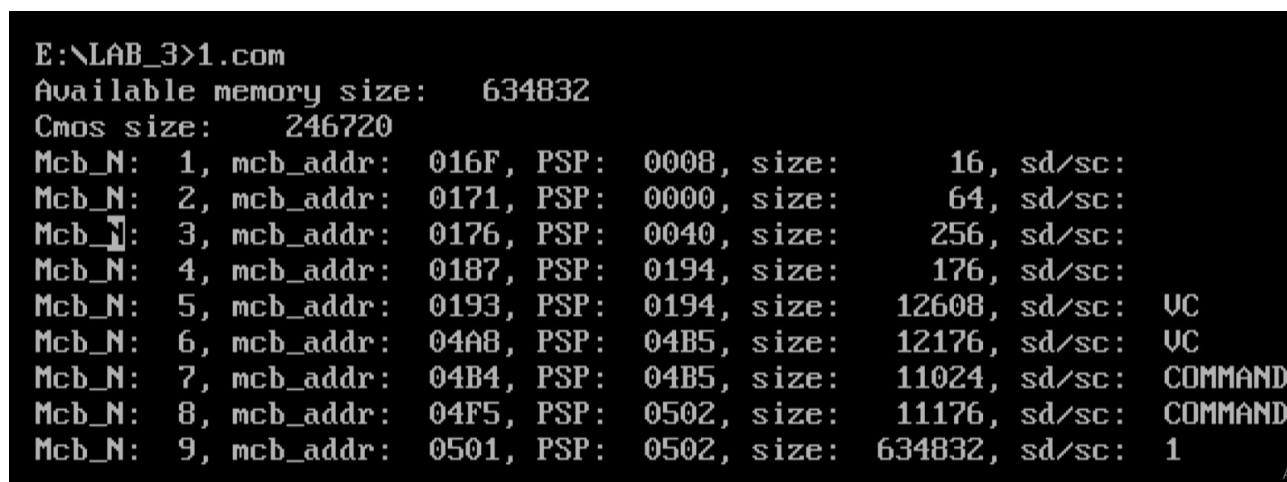
1. `byte_to_dec_2` – модификация функции из методических указаний, переводит шестнадцатичное число в десятичное.
2. `par_to_dec` – переводит кол-во параграфов, в байты.
3. `print_word` – данная функция выводит строку.
4. `print_mcb_chain` – получает адреа mcb из списка и выводит информацию о них.
5. `print_cmos_size` – выводит на экран информацию о размере расширенной памяти
6. `print_mcb` –выводит на экран информацию в данном mcb.

7. mem\_free – освобождает память, которая не занята программой.
8. get\_mem – запрашивает дополнительно 64КВ памяти и выводит сообщение об удачном или неудачном выделении памяти

### **Выполнение работы.**

**Шаг 1.** На первом шаге был написан модуль .COM который выводит на экран количество доступной памяти, размер расширенной памяти и цепочку блоков управления памятью.

Результат работы программы, написанной на первом шаге представлен на рисунке 1.



```
E:\LAB_3>1.com
Available memory size: 634832
Cmos size: 246720
Mcb_N: 1, mcb_addr: 016F, PSP: 0008, size: 16, sd/sc:
Mcb_N: 2, mcb_addr: 0171, PSP: 0000, size: 64, sd/sc:
Mcb_N: 3, mcb_addr: 0176, PSP: 0040, size: 256, sd/sc:
Mcb_N: 4, mcb_addr: 0187, PSP: 0194, size: 176, sd/sc:
Mcb_N: 5, mcb_addr: 0193, PSP: 0194, size: 12608, sd/sc: UC
Mcb_N: 6, mcb_addr: 04A8, PSP: 04B5, size: 12176, sd/sc: UC
Mcb_N: 7, mcb_addr: 04B4, PSP: 04B5, size: 11024, sd/sc: COMMAND
Mcb_N: 8, mcb_addr: 04F5, PSP: 0502, size: 11176, sd/sc: COMMAND
Mcb_N: 9, mcb_addr: 0501, PSP: 0502, size: 634832, sd/sc: 1
```

Рисунок 1 – Результат работы модуля 1.COM

**Шаг 2.** На втором шаге был написан модуль .COM который помимо вывода информации, освобождает неиспользуемую программой память.

```

E:\LAB_3>2.com
Available memory size: 634832
Cmos size: 246720
Mcb_N: 1, mcb_addr: 016F, PSP: 0008, size: 16, sd/sc:
Mcb_N: 2, mcb_addr: 0171, PSP: 0000, size: 64, sd/sc:
Mcb_N: 3, mcb_addr: 0176, PSP: 0040, size: 256, sd/sc:
Mcb_N: 4, mcb_addr: 0187, PSP: 0194, size: 176, sd/sc:
Mcb_N: 5, mcb_addr: 0193, PSP: 0194, size: 12608, sd/sc: UC
Mcb_N: 6, mcb_addr: 04A8, PSP: 04B5, size: 12176, sd/sc: UC
Mcb_N: 7, mcb_addr: 04B4, PSP: 04B5, size: 11024, sd/sc: COMMAND
Mcb_N: 8, mcb_addr: 04F5, PSP: 0502, size: 11176, sd/sc: COMMAND
Mcb_N: 9, mcb_addr: 0501, PSP: 0502, size: 11784, sd/sc: 2
Mcb_N: 10, mcb_addr: 0533, PSP: 0000, size: 634032, sd/sc: ыЛБ||@<

```

Рисунок 2 – Результат работы модуля 2.COM

По результатам работы предыдущих 2-х программ видно, что в первом случае программа занимала всю свободную память, однако на втором шаге выделился еще 1 блок свободной памяти, а программа заняла только то что необходимо для ее хранения.

**Шаг 3.** На третьем шаге был написан модуль .COM в котором после освобождения памяти, запрашивается дополнительный кусок памяти размером 64KB.

```

E:\LAB_3>3.com
Available memory size: 634832
Cmos size: 246720
memory request success
Mcb_N: 1, mcb_addr: 016F, PSP: 0008, size: 16, sd/sc:
Mcb_N: 2, mcb_addr: 0171, PSP: 0000, size: 64, sd/sc:
Mcb_N: 3, mcb_addr: 0176, PSP: 0040, size: 256, sd/sc:
Mcb_N: 4, mcb_addr: 0187, PSP: 0194, size: 176, sd/sc:
Mcb_N: 5, mcb_addr: 0193, PSP: 0194, size: 12608, sd/sc: UC
Mcb_N: 6, mcb_addr: 04A8, PSP: 04B5, size: 12176, sd/sc: UC
Mcb_N: 7, mcb_addr: 04B4, PSP: 04B5, size: 11024, sd/sc: COMMAND
Mcb_N: 8, mcb_addr: 04F5, PSP: 0502, size: 11176, sd/sc: COMMAND
Mcb_N: 9, mcb_addr: 0501, PSP: 0502, size: 11864, sd/sc: 3
Mcb_N: 10, mcb_addr: 0538, PSP: 0502, size: 65536, sd/sc: 3
Mcb_N: 11, mcb_addr: 1539, PSP: 0000, size: 568400, sd/sc: lay Link

```

Рисунок 3 – Результат работы модуля 3.COM

В тот раз выделился еще один блок памяти, который мы запрашивали.

**Шаг 4.** На четвертом шаге программа запрашивает дополнительную память до освобождения неиспользуемой.

Рисунок 4 – Результат выполнения модуля MAIN4.COM

```

E:\LAB_3>4.com
Available memory size: 634832
Cmos size: 246720
memory request failed
Mcb_N: 1, mcb_addr: 016F, PSP: 0008, size: 16, sd/sc:
Mcb_N: 2, mcb_addr: 0171, PSP: 0000, size: 64, sd/sc:
Mcb_N: 3, mcb_addr: 0176, PSP: 0040, size: 256, sd/sc:
Mcb_N: 4, mcb_addr: 0187, PSP: 0194, size: 176, sd/sc:
Mcb_N: 5, mcb_addr: 0193, PSP: 0194, size: 12608, sd/sc: UC
Mcb_N: 6, mcb_addr: 04A8, PSP: 04B5, size: 12176, sd/sc: UC
Mcb_N: 7, mcb_addr: 04B4, PSP: 04B5, size: 11024, sd/sc: COMMAND
Mcb_N: 8, mcb_addr: 04F5, PSP: 0502, size: 11176, sd/sc: COMMAND
Mcb_N: 9, mcb_addr: 0501, PSP: 0502, size: 11864, sd/sc: 4
Mcb_N: 10, mcb_addr: 0538, PSP: 0000, size: 633952, sd/sc: A>

```

В этом случае выделение памяти провалилось, т. к. программа итак занимала всю доступную память

### Контрольные вопросы.

Сегментный адрес недоступной памяти:

1. Что означает «доступный объём памяти»?

Это количество памяти, которое может использовать программа в процессе своего выполнения.

2. Где MCB блок Вашей программы в списке?

Программе принадлежат MCB блоки, у которых в поле SD/SC написано имя исполняемого файла. В 1-м случае 9-й, во втором 9-й и 10-й.

3. Какой размер памяти занимает программа в каждом случае?

Размер памяти, занимаемой программой, это сумма размеров всех блоков, принадлежащих программе. Шаг 1 – вся доступная память, шаг 2 – 11864 байт, шаг 3 – 11864 + 65536 байт, шаг 4 – 11864 байта.

### Выводы.

В ходе работы были изучены основные принципы структур данных и работы функций управления памятью ядра операционной системы.

# ПРИЛОЖЕНИЕ А.

## Исходный код модулей

### 1.asm:

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100h

    start:
        jmp begin
        available_memory_size db          "Available      memory      size:
", 0dh, 0ah, "$"

        cmos_size db                      "Cmos size:              ",
0dh, 0ah, "$"

        mcb_info db                      "Mcb_N:      , mcb_addr:      ,
PSP:      , size:      , sd/sc:      ", 0dh, 0ah, "$"

    begin:
        call main
        xor al, al
        mov ah, 4ch
        int 21h

    print_byte proc near
        push ax
        mov ah, 02h
        mov ah, 02h
        int 21h
        pop ax
        ret
    print_byte endp

    print_word proc near
        mov ah, 09h
        int 21h
        ret
    print_word 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
    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 al,00h
    je end_l
    or al,30h
    mov [si],al
end_l:
    pop ax
    pop dx
    pop cx
    ret
byte_to_dec endp

```

;-----Start-----

```

byte_to_dec_2 proc near
    push cx
    push dx
    push ax
    mov cx,10
loop_bd_2:

```



```

        div cx
        add dx,30h
        mov [si],dl
        dec si
        xor dx,dx
        cmp ax,10
        jae loop_bd_2
        cmp al,00h
        je end_l
        or al,30h
        mov [si],al
end_l_2:
        pop ax
        pop dx
        pop cx
        ret
byte_to_dec_2 endp

```

```

par_to_dec proc near
        push bx
        push ax
        push dx
        push si

        mov bx, 16
        mul bx                ; par too byte
        call byte_to_dec_2    ; byte to dec

        pop si
        pop dx
        pop ax
        pop bx
        ret
par_to_dec endp

```

```

print_available_memory_size proc near
        mov ah, 4ah
        mov bx, 0ffffh
        int 21h
        mov ax, bx
        mov si, offset available_memory_size
        add si, 30
        call par_to_dec
        mov dx, offset available_memory_size
        call print_word
        ret
print_available_memory_size endp

```

```

print_cmos_size proc near
        push ax
        push dx

```

```

    mov al, 30h
    out 70h, al
    in al, 71h
    mov al, 31h
    out 70h, al
    in al, 71h
    mov ah, al

    mov si, offset cmos_size
    add si, 19
    call par_to_dec
    mov dx, offset cmos_size
    call print_word

    pop dx
    pop ax
    ret
print_cmos_size endp

```

```

print_mcb proc near
    push ax
    push si
    push di
    push cx
    push dx
    push bx

    mov al, cl
    mov si, offset mcb_info
    add si, 8
    call byte_to_dec

    mov ax, es
    mov di, offset mcb_info
    add di, 25
    call wrd_to_hex

    mov ax, es:[1]
    mov di, offset mcb_info
    add di, 37
    call wrd_to_hex

    mov ax, es:[3]
    mov si, offset mcb_info
    add si, 52

    call par_to_dec

    mov bx, 8
    mov cx, 7
    mov si, offset mcb_info
    add si, 63
scsd_print_lp:
    mov dx, es:[bx]
    mov ds:[si], dx

```

```

    inc bx
    inc si
    loop scsd_print_lp

    mov dx, offset mcb_info
    call print_word

    pop bx
    pop dx
    pop cx
    pop di
    pop si
    pop ax
    ret
print_mcb endp

print_mcb_chain proc near
    push ax
    push es
    push cx
    push bx

    mov ah, 52h
    int 21h
    mov ax, es:[bx-2] ; first mcb
    mov es, ax
    mov cl, 1          ; number

get_mcb:
    call print_mcb

    mov al, es:[0]
    cmp al, 5ah        ; if last mcb
    je mcb_end

    mov ax, es          ; curent address
    add ax, es:[3]      ; get next address
    inc ax
    mov es, ax
    inc cl              ; inc number
    jmp get_mcb

mcb_end:
    pop bx
    pop cx
    pop es
    pop ax
    ret
print_mcb_chain endp

main proc near
    call print_available_memory_size
    call print_cmos_size
    call print_mcb_chain

```

```

        ret
main endp
TESTPC ends
end start

```

## 2.asm:

```

TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100h

start:
    jmp begin
    available_memory_size db          "Available      memory      size:
", 0dh, 0ah, "$"

    cmos_size db                      "Cmos size:          ",
0dh, 0ah, "$"

    mcb_info db                      "Mcb_N:      , mcb_addr:      ,
PSP:      , size:      , sd/sc:      ", 0dh, 0ah, "$"

    mem_message_S db                  "memory          request
success", 0dh, 0ah, "$"
    mem_message_F db                  "memory request failed",
0dh, 0ah, "$"

begin:
    call main
    xor al, al
    mov ah, 4ch
    int 21h

print_byte proc near
    push ax
    mov ah, 02h
    mov ah, 02h
    int 21h
    pop ax
    ret
print_byte endp

print_word proc near
    mov ah, 09h
    int 21h
    ret
print_word 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
    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 al,00h
    je end_l
    or al,30h
    mov [si],al
end_l:
    pop ax
    pop dx
    pop cx
    ret
byte_to_dec endp

```

;-----Start-----

```

byte_to_dec_2 proc near
    push cx
    push dx
    push ax
    mov cx,10
loop_bd_2:
    div cx
    add dx,30h
    mov [si],dl
    dec si
    xor dx,dx
    cmp ax,10
    jae loop_bd_2
    cmp al,00h
    je end_l_1
    or al,30h
    mov [si],al
end_l_1_2:
    pop ax
    pop dx
    pop cx
    ret
byte_to_dec_2 endp


par_to_dec proc near
    push bx
    push ax
    push dx
    push si

    mov bx, 16
    mul bx                ; par too byte
    call byte_to_dec_2    ; byte to dec

    pop si
    pop dx
    pop ax
    pop bx
    ret
par_to_dec endp


print_available_memory_size proc near
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    mov si, offset available_memory_size
    add si, 30
    call par_to_dec
    mov dx, offset available_memory_size
    call print_word
    ret
print_available_memory_size endp

```

```

print_cmos_size proc near
    push ax
    push dx

    mov al, 30h
    out 70h, al
    in al, 71h
    mov al, 31h
    out 70h, al
    in al, 71h
    mov ah, al

    mov si, offset cmos_size
    add si, 19
    call par_to_dec
    mov dx, offset cmos_size
    call print_word

    pop dx
    pop ax
    ret
print_cmos_size endp

```

```

print_mcb proc near
    push ax
    push si
    push di
    push cx
    push dx
    push bx

    mov al, cl
    mov si, offset mcb_info
    add si, 8
    call byte_to_dec                ; numer

    mov ax, es
    mov di, offset mcb_info
    add di, 25
    call wrd_to_hex                ; type

    mov ax, es:[1]
    mov di, offset mcb_info        ; psp
    add di, 37
    call wrd_to_hex

    mov ax, es:[3]                ; size
    mov si, offset mcb_info
    add si, 52

    call par_to_dec

    mov bx, 8

```

```

        mov cx, 7
        mov si, offset mcb_info                ;scsd
        add si, 63
scsd_print_lp:
        mov dx, es:[bx]
        mov ds:[si], dx
        inc bx
        inc si
        loop scsd_print_lp

        mov dx, offset mcb_info
        call print_word

        pop bx
        pop dx
        pop cx
        pop di
        pop si
        pop ax
        ret
print_mcb endp

print_mcb_chain proc near
        push ax
        push es
        push cx
        push bx

        mov ah, 52h
        int 21h
        mov ax, es:[bx-2] ; first mcb
        mov es, ax
        mov cl, 1          ; number

get_mcb:
        call print_mcb

        mov al, es:[0]
        cmp al, 5ah        ; if last mcb
        je mcb_end

        mov ax, es          ; curent address
        add ax, es:[3]      ; get next address
        inc ax
        mov es, ax
        inc cl              ; inc number
        jmp get_mcb

mcb_end:
        pop bx
        pop cx
        pop es
        pop ax
        ret
print_mcb_chain endp

```



```
;-----free-----
```

```
mem_free proc near
    push ax
    push bx
    push dx

    lea ax, testpc_end
    mov bx, 16
    xor dx, dx
    div bx
    inc ax
    mov bx, ax
    xor ax, ax
    mov ah, 4ah
    int 21h

    pop dx
    pop bx
    pop ax
    ret
mem_free endp

main proc near
    call print_available_memory_size
    call print_cmos_size
    call mem_free
    call print_mcb_chain
    ret
main endp

TESTPC_END:

TESTPC ends
end start
```

### 3.asm:

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100h

start:
    jmp begin
    available_memory_size db          "Available      memory      size:
", 0dh, 0ah, "$"

    cmos_size db                      "Cmos size:          ",
0dh, 0ah, "$"

    mcb_info db                      "Mcb_N:      , mcb_addr:      ,
PSP:      , size:      , sd/sc:      ", 0dh, 0ah, "$"

    mem_message_S db                  "memory          request
success", 0dh, 0ah, "$"
```

```
mem_message_F db
0dh, 0ah, "$"
```

```
"memory request failed",
```

```
begin:
```

```
    call main
    xor al, al
    mov ah, 4ch
    int 21h
```

```
print_byte proc near
```

```
    push ax
    mov ah, 02h
    mov ah, 02h
    int 21h
    pop ax
    ret
```

```
print_byte endp
```

```
print_word proc near
```

```
    mov ah, 09h
    int 21h
    ret
```

```
print_word 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
wrд_to_hex endp

byte_to_dec proc near
    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 al,00h
    je end_l
    or al,30h
    mov [si],al
end_l:
    pop ax
    pop dx
    pop cx
    ret
byte_to_dec endp

```

;-----Start-----

```

byte_to_dec_2 proc near
    push cx
    push dx
    push ax
    mov cx,10
loop_bd_2:
    div cx
    add dx,30h
    mov [si],dl
    dec si
    xor dx,dx
    cmp ax,10
    jae loop_bd_2
    cmp al,00h
    je end_l_2
    or al,30h
    mov [si],al
end_l_2:
    pop ax
    pop dx
    pop cx
    ret
byte_to_dec_2 endp

```

```

par_to_dec proc near
    push bx
    push ax
    push dx
    push si

    mov bx, 16
    mul bx                ; par too byte
    call byte_to_dec_2    ; byte to dec

    pop si
    pop dx
    pop ax
    pop bx
    ret
par_to_dec endp

```

```

print_available_memory_size proc near
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    mov si, offset available_memory_size
    add si, 30
    call par_to_dec
    mov dx, offset available_memory_size
    call print_word
    ret
print_available_memory_size endp

```

```

print_cmos_size proc near
    push ax
    push dx

    mov al, 30h
    out 70h, al
    in al, 71h
    mov al, 31h
    out 70h, al
    in al, 71h
    mov ah, al

    mov si, offset cmos_size
    add si, 19
    call par_to_dec
    mov dx, offset cmos_size
    call print_word

    pop dx
    pop ax
    ret
print_cmos_size endp

```

```

print_mcb proc near
    push ax
    push si
    push di
    push cx
    push dx
    push bx

    mov al, cl
    mov si, offset mcb_info
    add si, 8
    call byte_to_dec                ; numer

    mov ax, es
    mov di, offset mcb_info
    add di, 25
    call wrd_to_hex                ; type

    mov ax, es:[1]
    mov di, offset mcb_info        ; psp
    add di, 37
    call wrd_to_hex

    mov ax, es:[3]                ; size
    mov si, offset mcb_info
    add si, 52

    call par_to_dec

    mov bx, 8
    mov cx, 7
    mov si, offset mcb_info        ;scsd
    add si, 63
scsd_print_lp:
    mov dx, es:[bx]
    mov ds:[si], dx
    inc bx
    inc si
    loop scsd_print_lp

    mov dx, offset mcb_info
    call print_word

    pop bx
    pop dx
    pop cx
    pop di
    pop si
    pop ax
    ret
print_mcb endp

print_mcb_chain proc near
    push ax

```

```

    push es
    push cx
    push bx

    mov ah, 52h
    int 21h
    mov ax, es:[bx-2] ; first mcb
    mov es, ax
    mov cl, 1          ; number

get_mcb:
    call print_mcb

    mov al, es:[0]
    cmp al, 5ah        ; if last mcb
    je mcb_end

    mov ax, es          ; curent address
    add ax, es:[3]      ; get next address
    inc ax
    mov es, ax
    inc cl              ; inc number
    jmp get_mcb

mcb_end:
    pop bx
    pop cx
    pop es
    pop ax
    ret
print_mcb_chain endp

;-----free-----

mem_free proc near
    push ax
    push bx
    push dx

    lea ax, testpc_end
    mov bx, 16
    xor dx, dx
    div bx
    inc ax
    mov bx, ax
    xor ax, ax
    mov ah, 4ah
    int 21h

    pop dx
    pop bx
    pop ax
    ret
mem_free endp

```

```

get_mem PROC near
    mov     bx, 1000h ; 1000h ==64KB
    mov     ah, 48h
    int     21h
    jnc     success   ; CF = 1 ax = error code

fail:
    mov     dx, offset mem_message_F
    call    print_word
    jmp     get_mem_end

success:
    mov     dx, offset mem_message_S
    call    print_word

get_mem_end:
    ret
get_mem endp

```

```

main proc near
    call print_available_memory_size
    call print_cmos_size
    call mem_free
    call get_mem
    call print_mcb_chain
    ret
main endp

```

TESTPC\_END:

TESTPC ends  
end start

#### 4.asm:

```

TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100h

```

```

start:
    jmp begin
    available_memory_size db           "Available      memory      size:
", 0dh, 0ah, "$"

    cmos_size db                       "Cmos size:           ",
0dh, 0ah, "$"

    mcb_info db                        "Mcb_N:      , mcb_addr:      ,
PSP:      , size:      , sd/sc:      ", 0dh, 0ah, "$"

    mem_message_S db                  "memory      request
success", 0dh, 0ah, "$"
    mem_message_F db                  "memory request failed",
0dh, 0ah, "$"

```

```

begin:
    call main
    xor al, al
    mov ah, 4ch
    int 21h

print_byte proc near
    push ax
    mov ah, 02h
    mov ah, 02h
    int 21h
    pop ax
    ret
print_byte endp

print_word proc near
    mov ah, 09h
    int 21h
    ret
print_word 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
    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 al,00h
    je end_l
    or al,30h
    mov [si],al
end_l:
    pop ax
    pop dx
    pop cx
    ret
byte_to_dec endp

```

;-----Start-----

```

byte_to_dec_2 proc near
    push cx
    push dx
    push ax
    mov cx,10
loop_bd_2:
    div cx
    add dx,30h
    mov [si],dl
    dec si
    xor dx,dx
    cmp ax,10
    jae loop_bd_2
    cmp al,00h
    je end_l
    or al,30h
    mov [si],al
end_l_2:
    pop ax
    pop dx
    pop cx
    ret
byte_to_dec_2 endp

```

```

par_to_dec proc near
    push bx
    push ax

```

```

    push dx
    push si

    mov bx, 16
    mul bx                ; par too byte
    call byte_to_dec_2    ; byte to dec

    pop si
    pop dx
    pop ax
    pop bx
    ret
par_to_dec endp

```

```

print_available_memory_size proc near
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    mov si, offset available_memory_size
    add si, 30
    call par_to_dec
    mov dx, offset available_memory_size
    call print_word
    ret
print_available_memory_size endp

```

```

print_cmos_size proc near
    push ax
    push dx

    mov al, 30h
    out 70h, al
    in al, 71h
    mov al, 31h
    out 70h, al
    in al, 71h
    mov ah, al

    mov si, offset cmos_size
    add si, 19
    call par_to_dec
    mov dx, offset cmos_size
    call print_word

    pop dx
    pop ax
    ret
print_cmos_size endp

```

```

print_mcb proc near

```

```

push ax
push si
push di
push cx
push dx
push bx

mov al, cl
mov si, offset mcb_info
add si, 8
call byte_to_dec                ; numer

mov ax, es
mov di, offset mcb_info
add di, 25
call wrd_to_hex                ; type

mov ax, es:[1]
mov di, offset mcb_info        ; psp
add di, 37
call wrd_to_hex

mov ax, es:[3]                ; size
mov si, offset mcb_info
add si, 52

call par_to_dec

mov bx, 8
mov cx, 7
mov si, offset mcb_info        ;scsd
add si, 63
scsd_print_lp:
mov dx, es:[bx]
mov ds:[si], dx
inc bx
inc si
loop scsd_print_lp

mov dx, offset mcb_info
call print_word

pop bx
pop dx
pop cx
pop di
pop si
pop ax
ret
print_mcb endp

print_mcb_chain proc near
push ax
push es
push cx
push bx

```

```

    mov ah, 52h
    int 21h
    mov ax, es:[bx-2] ; first mcb
    mov es, ax
    mov cl, 1          ; number

get_mcb:
    call print_mcb

    mov al, es:[0]
    cmp al, 5ah        ; if last mcb
    je mcb_end

    mov ax, es          ; curent address
    add ax, es:[3]      ; get next address
    inc ax
    mov es, ax
    inc cl              ; inc number
    jmp get_mcb

mcb_end:
    pop bx
    pop cx
    pop es
    pop ax
    ret
print_mcb_chain endp

;-----free-----

mem_free proc near
    push ax
    push bx
    push dx

    lea ax, testpc_end
    mov bx, 16
    xor dx, dx
    div bx
    inc ax
    mov bx, ax
    xor ax, ax
    mov ah, 4ah
    int 21h

    pop dx
    pop bx
    pop ax
    ret
mem_free endp

get_mem PROC near
    mov     bx, 1000h ; 1000h ==64KB

```

```

        mov     ah, 48h
        int     21h
        jnc     success      ; CF = 1 ax = error code

fail:
        mov     dx, offset mem_message_F
        call    print_word
        jmp     get_mem_end

success:
        mov     dx, offset mem_message_S
        call    print_word

get_mem_end:
        ret
get_mem endp

main proc near
        call print_available_memory_size
        call print_cmos_size
        call get_mem
        call mem_free
        call print_mcb_chain
        ret
main endp

TESTPC_END:

TESTPC ends
end start

```