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 _____ Кондратов Ю.А.

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 щестнадцатеричными числами. Объем памяти функциями управления памятью выводится в параграфах. Необходимо преобразовать его в байты и выводить в виде десятичных чисел. Последние восемь байт MCB 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. print_mem_size –выполняет вывод на экран строки вида «Available mem size: {количество памяти в байтах}».
2. par_to_byte –переводит число параграфов, записанное в AX в байты и записывает результат справа налево начиная с адреса ES:[SI].
3. print – данная функция выводит строку, смещение которой находится в DX.
4. print_cmos_size –выполняет вывод на экран строки вида «Extended mem size: {количество расширенной памяти в байтах}».
5. print_mcb_list –выполняет вывод на экран таблицы, содержащей информация о списке MCB.
6. print_one_mcb –выполняет вывод на экран информации об одном MCB в виде строки «MCB: {номер MCB}, addr: {адрес блока}, owner PSP: {адрес PSP владельца}, size: {размер блока}, SD/SC: {SC/SD}».

7. byte_to_dec – преобразует число в регистре AL в ASCII коды символов его десятичного представления.
8. wrd_to_hex – преобразует число в регистре AX в ASCII коды символов его десятичного представления.
9. free_mem – освобождает память, не занимаемую программой.
10. ask_mem – запрашивает дополнительные 64КБ памяти. В случае ошибки выводит сообщение «Memory allocation failed!».

Выполнение работы.

Исходные коды модулей представлены в приложении А.

Шаг 1. На первом шаге были написаны функции print_mem_size, par_to_byte, print_cmos_size, print, print_mcb_list, print_one_mcb. Эти функции необходимо для выполнения задач вывода количества доступной памяти, размера расширенной памяти, цепочки блоков управления памятью.

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

```
C:\>MAIN1.com
Available mem size:   648912
Extended mem size:    246720
MCB:  1, addr:  016F, owner PSP:  0008, size:      16, SD/SC:
MCB:  2, addr:  0171, owner PSP:  0000, size:      64, SD/SC: DPMILOAD
MCB:  3, addr:  0176, owner PSP:  0040, size:     256, SD/SC:
MCB:  4, addr:  0187, owner PSP:  0192, size:     144, SD/SC:
MCB:  5, addr:  0191, owner PSP:  0192, size:  648912, SD/SC: MAIN1
```

Рисунок 1 – Результат запуска модуля MAIN1.COM

Шаг 2. На втором шаге была написана функция free_mem, необходимая для выполнения задачи освобождения памяти, не занимаемой программой. Результат работы программы, написанной на втором шаге представлен на рисунке 2.

```
C:\>MAIN2.com
Available mem size:   648912
Extended mem size:    246720
MCB:  1, addr:  016F, owner PSP:  0008, size:      16, SD/SC:
MCB:  2, addr:  0171, owner PSP:  0000, size:      64, SD/SC: DPMILOAD
MCB:  3, addr:  0176, owner PSP:  0040, size:     256, SD/SC:
MCB:  4, addr:  0187, owner PSP:  0192, size:     144, SD/SC:
MCB:  5, addr:  0191, owner PSP:  0192, size:     816, SD/SC: MAIN2
MCB:  6, addr:  01C5, owner PSP:  0000, size:  648080, SD/SC:
```

Рисунок 2 – Результат выполнения модуля MAIN2.COM

По выводу программы видно, что на первом шаге программа занимала практически всю свободную память, однако на втором шаге свободная память выделилась в отдельный (шестой) блок, а блок, в который загружена программа стал значительно меньше, ужавшись точно до размеров, необходимых для хранения программы.

Шаг 3. На третьем шаге была написана функция ask_mem, необходимая для выполнения задача запрашивание дополнительных 64КБ памяти. Данная функция вызывается после освобождения памяти. Результат выполнения программы, написанной на третьем шаге представлен на рисунке 3.

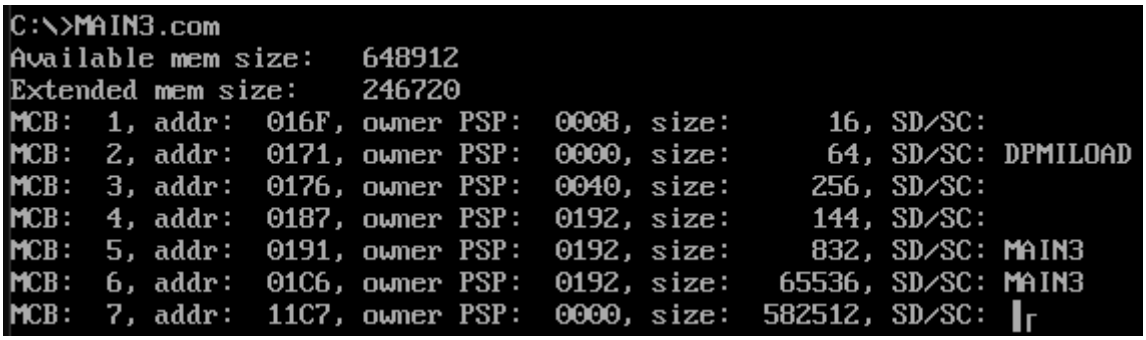

Рисунок 3 – Результат выполнения модуля MAIN3

В данном случае программе принадлежат два блока памяти – тот, который остался после освобождения неиспользуемой памяти и выделенный при помощи функции ask_mem (под номерами 5 и 6) соответственно.

Шаг 4. Четвертый шаг аналогичен третьему за тем лишь исключением, что функция ask_mem вызывается перед free_mem. Результат выполнения программы, написанной на четвёртом шаге представлен на рисунке 4.

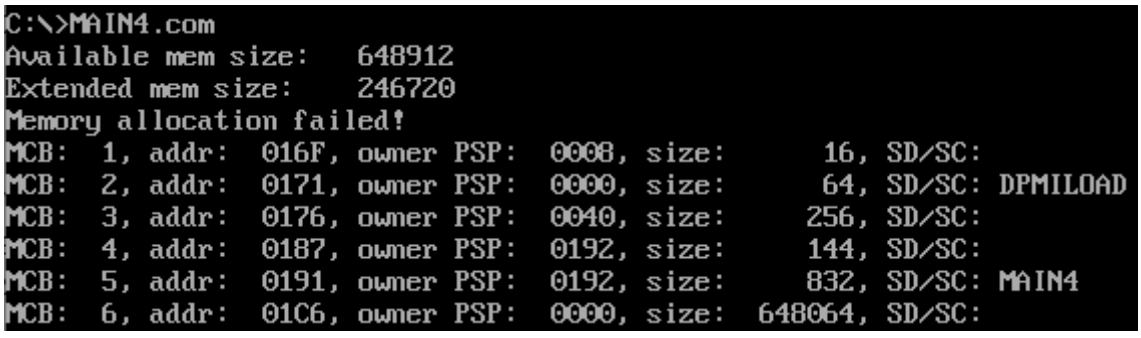

Рисунок 4 – Результат выполнения модуля MAIN4.COM

В 3 строке вывода можно заметить сообщение об ошибке во время аллокации памяти. Действительно, программе уже принадлежит вся свободная память, поэтому запрос о выделении дополнительных 64КБ вызывает ошибку.

Контрольные вопросы.

Сегментный адрес недоступной памяти:

1. Что означает «доступный объём памяти»?

Это количество оперативной памяти, которое может быть использовано программой в процессе выполнения (для использования может потребоваться запрос к операционной системе для выделения памяти).

2. Где MCB блок Вашей программы в списке?

На каждом шаге программе принадлежат MCB блоки, в графе SD/SC которых написано MAIN{номер шага} (название исполняемого модуля).

3. Какой размер памяти занимает программа в каждом случае?

Размер памяти, занимаемой программой, в каждом случае рассчитывается как сумма значение в графе size у всех блоков, принадлежащих программе. Шаг 1 – 648912 байт, шаг 2 – 816 байт, шаг 3 – 66368 байт, шаг 4 – 832 байта.

Выводы.

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

ПРИЛОЖЕНИЕ А.

Исходный код модулей

main1.asm:

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

start:
    jmp begin

data:
    MEM_SIZE db "Available mem size:         ", 0dh, 0ah, "$"
    CMOS_SIZE db "Extended mem size:         ", 0dh, 0ah, "$"
    MCB_ROW db "MCB:   , addr:      , owner PSP:      , size:       ,
SD/SC:          ", 0dh, 0ah, "$"

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

print PROC NEAR
    mov ah, 09h
    int 21h
    ret
print 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

print_symbol PROC NEAR
    push ax
    mov ah, 02h
    int 21h
    pop ax
    ret
print_symbol ENDP

par_to_byte PROC NEAR
    ; ES - dest seg, SI - dest offset (end), AX - paragraphs number
    push bx
    push ax
    push dx
    push si

    mov bx, 10h
    mul bx
    mov bx, 0ah
```

```
div_loop:
    div bx
    add dx, 30h
    mov es:[si], dl
    xor dx, dx
    dec si
    cmp ax, 0000h
    jne div_loop

    pop si
    pop dx
    pop ax
    pop bx
    ret
par_to_byte ENDP

print_mem_size PROC NEAR
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    mov si, offset MEM_SIZE
    add si, 27
    call par_to_byte
    mov dx, offset MEM_SIZE
    call print
    ret
print_mem_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, 27
    call par_to_byte
    mov dx, offset CMOS_SIZE
    call print

    pop dx
    pop ax
    ret
print_cmos_size ENDP

print_one_mcb PROC NEAR
    push ax
    push si
    push di
    push cx
    push dx
```

```
    push bx

    mov al, cl
    mov si, offset MCB_ROW
    add si, 6
    call byte_to_dec

    mov ax, es
    mov di, offset MCB_ROW
    add di, 19
    call wrd_to_hex

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

    mov ax, es:[3]
    mov si, offset MCB_ROW
    add si, 52
    push es
    mov dx, ds
    mov es, dx
    call par_to_byte
    pop es

    mov bx, 8
    mov cx, 7
    mov si, offset MCB_ROW
    add si, 62
scsd_loop:
    mov dx, es:[bx]
    mov ds:[si], dx
    inc bx
    inc si
    loop scsd_loop

    mov dx, offset MCB_ROW
    call print

    pop bx
    pop dx
    pop cx
    pop di
    pop si
    pop ax
    ret
print_one_mcb ENDP

print_mcb_list PROC NEAR
    push ax
    push es
    push cx
    push bx

    mov ah, 52h
    int 21h
    mov ax, es:[bx-2]
```

```
        mov es, ax
        mov cl, 1

    mcb_loop:
        call print_one_mcb

        mov al, es:[0]
        cmp al, 5ah
        je mcb_end

        mov bx, es:[3]
        mov ax, es
        add ax, bx
        inc ax
        mov es, ax
        inc cl
        jmp mcb_loop

    mcb_end:
        pop bx
        pop cx
        pop es
        pop ax
        ret
    print_mcb_list ENDP

    main PROC NEAR
        call print_mem_size
        call print_cmos_size
        call print_mcb_list
        ret
    main ENDP

    main_seg ENDS
    END start
```

main2.asm:

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

    start:
        jmp begin

    data:
        MEM_SIZE db "Available mem size:         ", 0dh, 0ah, "$"
        CMOS_SIZE db "Extended mem size:         ", 0dh, 0ah, "$"
        MCB_ROW db "MCB:   , addr:      , owner PSP:      , size:       ,
SD/SC:          ", 0dh, 0ah, "$"
        MEM_FAIL db "Memory allocation failed!", 0dh, 0ah, "$"

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

```
print PROC NEAR
    mov ah, 09h
    int 21h
    ret
print 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

print_symbol PROC NEAR
    push ax
    mov ah, 02h
    int 21h
    pop ax
    ret
print_symbol ENDP

par_to_byte PROC NEAR
    ; ES - dest seg, SI - dest offset (end), AX - paragraphs number
    push bx
    push ax
    push dx
    push si

    mov bx, 10h
    mul bx
    mov bx, 0ah
div_loop:
    div bx
    add dx, 30h
    mov es:[si], dl
    xor dx, dx
    dec si
    cmp ax, 0000h
    jne div_loop

    pop si
    pop dx
    pop ax
    pop bx
    ret
par_to_byte ENDP

print_mem_size PROC NEAR
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    mov si, offset MEM_SIZE
    add si, 27
    call par_to_byte
    mov dx, offset MEM_SIZE
    call print
```

```
    ret
print_mem_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, 27
    call par_to_byte
    mov dx, offset CMOS_SIZE
    call print

    pop dx
    pop ax
    ret
print_cmos_size ENDP

print_one_mcb PROC NEAR
    push ax
    push si
    push di
    push cx
    push dx
    push bx

    mov al, cl
    mov si, offset MCB_ROW
    add si, 6
    call byte_to_dec

    mov ax, es
    mov di, offset MCB_ROW
    add di, 19
    call wrd_to_hex

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

    mov ax, es:[3]
    mov si, offset MCB_ROW
    add si, 52
    push es
    mov dx, ds
    mov es, dx
    call par_to_byte
    pop es
```

```
    mov bx, 8
    mov cx, 7
    mov si, offset MCB_ROW
    add si, 62
scsd_loop:
    mov dx, es:[bx]
    mov ds:[si], dx
    inc bx
    inc si
    loop scsd_loop

    mov dx, offset MCB_ROW
    call print

    pop bx
    pop dx
    pop cx
    pop di
    pop si
    pop ax
    ret
print_one_mcb ENDP

print_mcb_list PROC NEAR
    push ax
    push es

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

mcb_loop:
    call print_one_mcb

    mov al, es:[0]
    cmp al, 5ah
    je mcb_end

    mov bx, es:[3]
    mov ax, es
    add ax, bx
    inc ax
    mov es, ax
    inc cl
    jmp mcb_loop

mcb_end:
    pop es
    pop ax
    ret
print_mcb_list ENDP

free_mem PROC NEAR
    push ax
    push bx
    push dx
```

```
        lea ax, end_addr
        mov bx, 10h
        xor dx, dx
        div bx
        inc ax
        mov bx, ax
        mov al, 0
        mov ah, 4ah
        int 21h

        pop dx
        pop bx
        pop ax
        ret
    free_mem ENDP

    main PROC NEAR
        call print_mem_size
        call print_cmos_size
        call free_mem
        call print_mcb_list
        ret
    main ENDP

    end_addr:
    main_seg ENDS
    END start
```

main3.asm:

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

    start:
        jmp begin

    data:
        MEM_SIZE db "Available mem size:         ", 0dh, 0ah, "$"
        CMOS_SIZE db "Extended mem size:         ", 0dh, 0ah, "$"
        MCB_ROW db "MCB:   , addr:      , owner PSP:      , size:       ,
SD/SC:          ", 0dh, 0ah, "$"
        MEM_FAIL db "Memory allocation failed!", 0dh, 0ah, "$"

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

    print PROC NEAR
        mov ah, 09h
        int 21h
        ret
    print 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

print_symbol PROC NEAR
    push ax
    mov ah, 02h
    int 21h
    pop ax
    ret
print_symbol ENDP

par_to_byte PROC NEAR
    ; ES - dest seg, SI - dest offset (end), AX - paragraphs number
    push bx
    push ax
    push dx
    push si

    mov bx, 10h
    mul bx
    mov bx, 0ah
div_loop:
    div bx
    add dx, 30h
    mov es:[si], dl
    xor dx, dx
    dec si
    cmp ax, 0000h
    jne div_loop

    pop si
    pop dx
    pop ax
    pop bx
    ret
par_to_byte ENDP

print_mem_size PROC NEAR
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    mov si, offset MEM_SIZE
    add si, 27
    call par_to_byte
    mov dx, offset MEM_SIZE
    call print
    ret
print_mem_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, 27
    call par_to_byte
    mov dx, offset CMOS_SIZE
    call print

    pop dx
    pop ax
    ret
print_cmos_size ENDP

print_one_mcb PROC NEAR
    push ax
    push si
    push di
    push cx
    push dx
    push bx

    mov al, cl
    mov si, offset MCB_ROW
    add si, 6
    call byte_to_dec

    mov ax, es
    mov di, offset MCB_ROW
    add di, 19
    call wrd_to_hex

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

    mov ax, es:[3]
    mov si, offset MCB_ROW
    add si, 52
    push es
    mov dx, ds
    mov es, dx
    call par_to_byte
    pop es

    mov bx, 8
    mov cx, 7
    mov si, offset MCB_ROW
    add si, 62
scsd_loop:
    mov dx, es:[bx]
    mov ds:[si], dx
```

```
    inc bx
    inc si
    loop scsd_loop

    mov dx, offset MCB_ROW
    call print

    pop bx
    pop dx
    pop cx
    pop di
    pop si
    pop ax
    ret
print_one_mcb ENDP

print_mcb_list PROC NEAR
    push ax
    push es

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

mcb_loop:
    call print_one_mcb

    mov al, es:[0]
    cmp al, 5ah
    je mcb_end

    mov bx, es:[3]
    mov ax, es
    add ax, bx
    inc ax
    mov es, ax
    inc cl
    jmp mcb_loop

mcb_end:
    pop es
    pop ax
    ret
print_mcb_list ENDP

free_mem PROC NEAR
    push ax
    push bx
    push dx

    lea ax, end_addr
    mov bx, 10h
    xor dx, dx
    div bx
    inc ax
    mov bx, ax
```

```
        mov al, 0
        mov ah, 4ah
        int 21h

        pop dx
        pop bx
        pop ax
        ret
    free_mem ENDP

    ask_mem PROC NEAR
        push ax
        push bx
        push dx

        mov bx, 1000h
        mov ah, 48h
        int 21h

        jnc end_ask
        mov dx, offset MEM_FAIL
        call print

    end_ask:
        pop dx
        pop bx
        pop cx
        ret
    ask_mem ENDP

    main PROC NEAR
        call print_mem_size
        call print_cmos_size
        call free_mem
        call ask_mem
        call print_mcb_list
        ret
    main ENDP

    end_addr:
    main_seg ENDS
    END start
```

main4.asm:

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

    start:
        jmp begin

    data:
        MEM_SIZE db "Available mem size:         ", 0dh, 0ah, "$"
        CMOS_SIZE db "Extended mem size:         ", 0dh, 0ah, "$"
        MCB_ROW db "MCB:   , addr:      , owner PSP:      , size:       ,
SD/SC:         ", 0dh, 0ah, "$"
```

```
    MEM_FAIL db "Memory allocation failed!", 0dh, 0ah, "$"

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

print PROC NEAR
    mov ah, 09h
    int 21h
    ret
print 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

print_symbol PROC NEAR
    push ax
    mov ah, 02h
    int 21h
    pop ax
    ret
print_symbol ENDP

par_to_byte PROC NEAR
    ; ES - dest seg, SI - dest offset (end), AX - paragraphs number
    push bx
    push ax
    push dx
    push si

    mov bx, 10h
    mul bx
    mov bx, 0ah
div_loop:
    div bx
    add dx, 30h
    mov es:[si], dl
    xor dx, dx
    dec si
    cmp ax, 0000h
    jne div_loop

    pop si
    pop dx
    pop ax
    pop bx
    ret
par_to_byte ENDP

print_mem_size PROC NEAR
    mov ah, 4ah
    mov bx, 0ffffh
```

```
    int 21h
    mov ax, bx
    mov si, offset MEM_SIZE
    add si, 27
    call par_to_byte
    mov dx, offset MEM_SIZE
    call print
    ret
print_mem_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, 27
    call par_to_byte
    mov dx, offset CMOS_SIZE
    call print

    pop dx
    pop ax
    ret
print_cmos_size ENDP

print_one_mcb PROC NEAR
    push ax
    push si
    push di
    push cx
    push dx
    push bx

    mov al, cl
    mov si, offset MCB_ROW
    add si, 6
    call byte_to_dec

    mov ax, es
    mov di, offset MCB_ROW
    add di, 19
    call wrd_to_hex

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

    mov ax, es:[3]
    mov si, offset MCB_ROW
```

```
    add si, 52
    push es
    mov dx, ds
    mov es, dx
    call par_to_byte
    pop es

    mov bx, 8
    mov cx, 7
    mov si, offset MCB_ROW
    add si, 62
scsd_loop:
    mov dx, es:[bx]
    mov ds:[si], dx
    inc bx
    inc si
    loop scsd_loop

    mov dx, offset MCB_ROW
    call print

    pop bx
    pop dx
    pop cx
    pop di
    pop si
    pop ax
    ret
print_one_mcb ENDP

print_mcb_list PROC NEAR
    push ax
    push es

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

mcb_loop:
    call print_one_mcb

    mov al, es:[0]
    cmp al, 5ah
    je mcb_end

    mov bx, es:[3]
    mov ax, es
    add ax, bx
    inc ax
    mov es, ax
    inc cl
    jmp mcb_loop

mcb_end:
    pop es
    pop ax
```

```
    ret
print_mcb_list ENDP

free_mem PROC NEAR
    push ax
    push bx
    push dx

    lea ax, end_addr
    mov bx, 10h
    xor dx, dx
    div bx
    inc ax
    mov bx, ax
    mov al, 0
    mov ah, 4ah
    int 21h

    pop dx
    pop bx
    pop ax
    ret
free_mem ENDP

ask_mem PROC NEAR
    push ax
    push bx
    push dx

    mov bx, 1000h
    mov ah, 48h
    int 21h

    jnc end_ask
    mov dx, offset MEM_FAIL
    call print

end_ask:
    pop dx
    pop bx
    pop cx
    ret
ask_mem ENDP

main PROC NEAR
    call print_mem_size
    call print_cmos_size
    call ask_mem
    call free_mem
    call print_mcb_list
    ret
main ENDP

end_addr:
main_seg ENDS
END start
```