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 памятью.

Студент гр. 0382 _____ Санников В.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.

Задание.

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 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

Ход работы.

Для выполнение лабораторной работы были написаны 4 .COM модуля.

Для выполнения задания были использованы процедуры:

1) memory_func — выводит на консоль размер доступной памти в байтах.

2) cmos_func — выводит на консоль размер расширенной памяти в килобайтах.

3) mcb_func — выводит на консоль цепочку блоков управления памяти.

4) free_memory — процедура освобождения памяти.

5) memory_request — процедура, которая запрашивает 64 Кб памяти и проверяет флаг CF.

Результаты выполнения пунктов задания приведены на следующих рисунках:

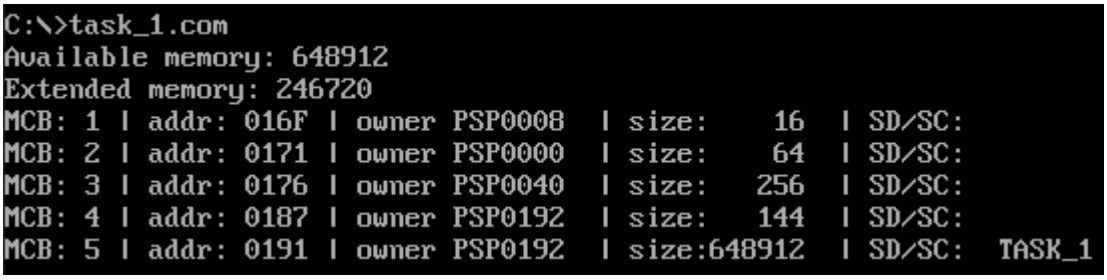

Рисунок 1 — Результат выполнения файла task_1.com

Как видно из данного рисунка, программа занимает всю доступную память.

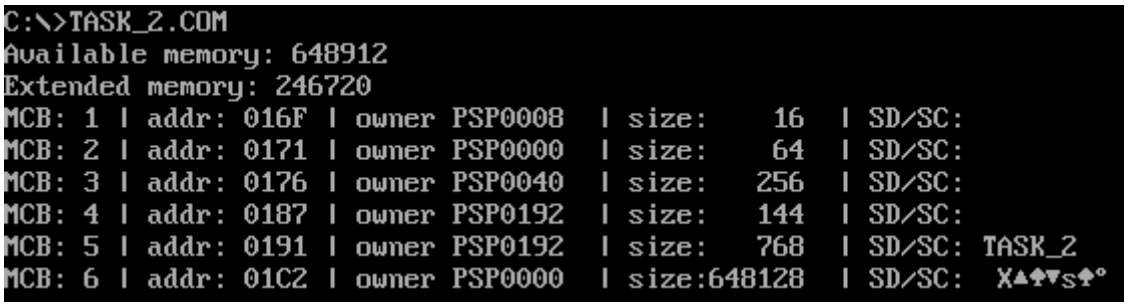

Рисунок 2 — Результат выполнения файла task_2.com

На данном шаге программа использует только необходимую ей память.

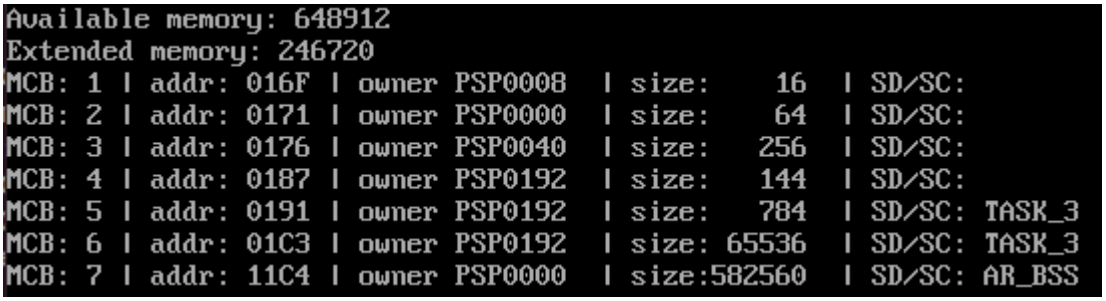

Рисунок 3 — Результат выполнения файла task_3.com

В данном случае на рисунке видно, что программа занимает не только необходимую ей память, а еще и запрошенные 64 Кб.

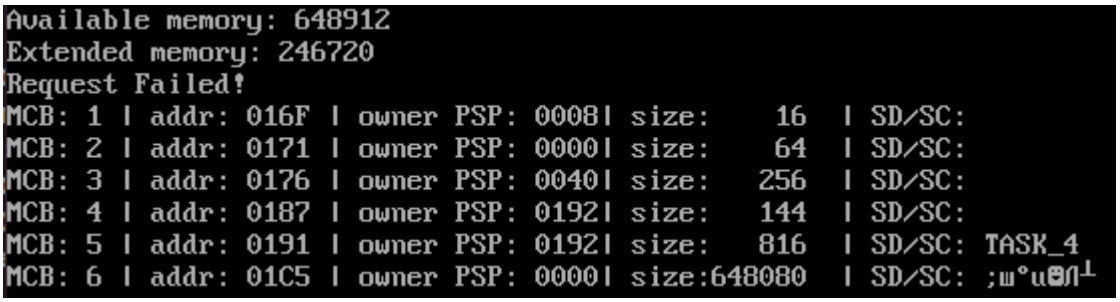

Рисунок 4 — Результат выполнения файла task_4.com

Из рисунка 4 мы можем наблюдать, что дополнительная память не была выделена, ведь запрос приходил тогда, когда не было доступной памяти.

Исходный код программы см в приложении А.

Ответы на контрольные вопросы.

1. Что означает «доступный объём памяти»?

Это объем памяти, который выделяет программа для использования нашим модулем.

2. Где MCB блок Вашей программы в списке?

На первом шаге 5 место в списке, на втором аналогично, на третьем шаге 5 и 6 место, на четвертом 5 место в списке.

3. Какой размер памяти занимает программа в каждом случае?

На первом шаге программа занимает всю доступную память — 648912 байт, на втором шаге - 768 байт, на третьем - 66320 байт и на четвертом - 816 байт.

Вывод.

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

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл task_1.asm

```
TESTPC SEGMENT
     ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
     ORG  100H
START:    jmp BEGIN
;------------------------------
MEMORY_SIZE db 'Available memory:          ', 0DH, 0AH, '$'
CMOS_SIZE db 'Extended memory:          ', 0DH, 0AH, '$'
MCB db 'MCB:    | addr:       | owner PSP:      | size:          |
SD/SC:             ', 0DH, 0AH, '$'
;------------------------------
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-го 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, в 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 10 с/с, в 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_1
      or AL, 30h
      mov [SI], AL
end_1:  pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP
;----------------------
BYTE_FUNC PROC near
      push ax
      push bx
      push dx
      push si

      mov bx, 10h
      mul bx
      mov bx, 0ah
      byte_loop:
          div bx
          add dx, 30h
          mov es:[si], dl
          xor dx, dx
          dec si
          cmp ax, 0000h
             jne byte_loop
         pop si
         pop dx
         pop bx
         pop ax
      ret
BYTE_FUNC ENDP
;----------------------
```

```
PRINT PROC near
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
PRINT ENDP
;---------------------
memory_func PROC near
      mov ah, 4ah
      mov bx, 0FFFFh
      int 21h
      mov ax, bx
      mov si, offset MEMORY_SIZE
      add si, 23
      call BYTE_FUNC
      mov dx, offset MEMORY_SIZE
      call PRINT
      ret
memory_func ENDP
;-----------------------------------
cmos_func 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, 22
      call BYTE_FUNC
      mov dx, offset CMOS_SIZE
      call PRINT

      pop dx
      pop ax
      ret
cmos_func ENDP
;-----------------------------------
mcb_func PROC near
      push ax
      push bx
      push cx
      push es

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

```
mov es, ax
mov cl, 1

check_mcb:
     push ax
     push bx
     push dx
     push cx
     push si
     push di

     mov al, cl
     mov si, offset MCB
     add si, 5
     call BYTE_TO_DEC

     mov ax, es
     mov di, offset MCB
     add di, 18
     call WRD_TO_HEX

     mov ax, es:[1]
     mov di, offset MCB
     add di, 34
     call WRD_TO_HEX

     mov ax, es:[3]
     mov si, offset MCB
     add si, 49
     push es
     mov dx, ds
     mov es, dx
     call BYTE_FUNC
     pop es

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

     mov dx, offset MCB
     call PRINT

     pop di
     pop si
     pop cx
```

```
            pop dx
            pop bx
            pop ax

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

            mov bx, es:[3]
            mov ax, es
            add ax, bx
            inc ax
            mov es, ax
            inc cl
            jmp check_mcb
      final:
            push es
            push cx
            push bx
            push ax
            ret
mcb_func ENDP
;---------------------------------
BEGIN:
      call memory_func
      call cmos_func
      call mcb_func

      xor AL, AL
      mov AH, 4Ch
      int 21h
TESTPC ENDS
      END START
```

Файл task_2.asm

```
TESTPC SEGMENT
      ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
      ORG  100H
START:    jmp BEGIN
;-------------------------------
MEMORY_SIZE db 'Available memory:          ', 0DH, 0AH, '$'
CMOS_SIZE db 'Extended memory:          ', 0DH, 0AH, '$'
MCB db 'MCB:    | addr:       | owner PSP:      | size:          |
SD/SC:                ', 0DH, 0AH, '$'
;-------------------------------
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-го 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, в 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 10 с/с, в 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_1
      or AL, 30h
      mov [SI], AL
```

```
end_1:  pop DX
     pop CX
     ret
BYTE_TO_DEC ENDP
;----------------------
BYTE_FUNC PROC near
     push ax
     push bx
     push dx
     push si

     mov bx, 10h
     mul bx
     mov bx, 0ah
     byte_loop:
         div bx
         add dx, 30h
         mov es:[si], dl
         xor dx, dx
         dec si
         cmp ax, 0000h
            jne byte_loop
        pop si
        pop dx
        pop bx
        pop ax
     ret
BYTE_FUNC ENDP
;----------------------
PRINT PROC near
     push AX
     mov AH, 09h
     int 21h
     pop AX
     ret
PRINT ENDP
;----------------------
memory_func PROC near
     mov ah, 4ah
     mov bx, 0FFFFh
     int 21h
     mov ax, bx
     mov si, offset MEMORY_SIZE
     add si, 23
     call BYTE_FUNC
     mov dx, offset MEMORY_SIZE
     call PRINT
     ret
memory_func ENDP
;-----------------------------------
cmos_func 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, 22
      call BYTE_FUNC
      mov dx, offset CMOS_SIZE
      call PRINT

      pop dx
      pop ax
      ret
cmos_func ENDP
;-----------------------------------
mcb_func PROC near
      push ax
      push bx
      push cx
      push es

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

      check_mcb:
            push ax
            push bx
            push dx
            push cx
            push si
            push di

            mov al, cl
            mov si, offset MCB
            add si, 5
            call BYTE_TO_DEC

            mov ax, es
            mov di, offset MCB
            add di, 18
            call WRD_TO_HEX

            mov ax, es:[1]
            mov di, offset MCB
```

```
            add di, 34
            call WRD_TO_HEX

            mov ax, es:[3]
            mov si, offset MCB
            add si, 49
            push es
            mov dx, ds
            mov es, dx
            call BYTE_FUNC
            pop es

            mov bx, 8
            mov cx, 7
            mov si, offset MCB
            add si, 61
            scsd_loop:
                mov dx, es:[bx]
                mov ds:[si], dx
                inc bx
                inc si
                loop scsd_loop

            mov dx, offset MCB
            call PRINT

            pop di
            pop si
            pop cx
            pop dx
            pop bx
            pop ax

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

            mov bx, es:[3]
            mov ax, es
            add ax, bx
            inc ax
            mov es, ax
            inc cl
            jmp check_mcb
      final:
            push es
            push cx
            push bx
            push ax
            ret
mcb_func ENDP
;---------------------------------
```

```
free_memory PROC NEAR
    push ax
    push bx
    push dx

    lea ax, quit_prog
    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_memory ENDP
;-----------------------------------
BEGIN:
     call memory_func
     call cmos_func
     call free_memory
     call mcb_func

     xor AL, AL
     mov AH, 4Ch
     int 21h
quit_prog:
TESTPC ENDS
     END START
```

Файл task_3.asm

```
TESTPC SEGMENT
     ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
     ORG  100H
START:    jmp BEGIN
;-------------------------------
MEMORY_SIZE db 'Available memory:          ', 0DH, 0AH, '$'
CMOS_SIZE db 'Extended memory:          ', 0DH, 0AH, '$'
MCB db 'MCB:    | addr:       | owner PSP:      | size:          |
SD/SC:                ', 0DH, 0AH, '$'
;-------------------------------
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-го 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, в 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 10 с/с, в 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_1
      or AL, 30h
```

```
     mov [SI], AL
end_1:  pop DX
     pop CX
     ret
BYTE_TO_DEC ENDP
;----------------------
BYTE_FUNC PROC near
     push ax
     push bx
     push dx
     push si

     mov bx, 10h
     mul bx
     mov bx, 0ah
     byte_loop:
         div bx
         add dx, 30h
         mov es:[si], dl
         xor dx, dx
         dec si
         cmp ax, 0000h
            jne byte_loop
        pop si
        pop dx
        pop bx
        pop ax
     ret
BYTE_FUNC ENDP
;----------------------
PRINT PROC near
     push AX
     mov AH, 09h
     int 21h
     pop AX
     ret
PRINT ENDP
;----------------------
memory_func PROC near
     mov ah, 4ah
     mov bx, 0FFFFh
     int 21h
     mov ax, bx
     mov si, offset MEMORY_SIZE
     add si, 23
     call BYTE_FUNC
     mov dx, offset MEMORY_SIZE
     call PRINT
     ret
memory_func ENDP
;-----------------------------------
cmos_func 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, 22
      call BYTE_FUNC
      mov dx, offset CMOS_SIZE
      call PRINT

      pop dx
      pop ax
      ret
cmos_func ENDP
;-----------------------------------
mcb_func PROC near
      push ax
      push bx
      push cx
      push es

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

      check_mcb:
            push ax
            push bx
            push dx
            push cx
            push si
            push di

            mov al, cl
            mov si, offset MCB
            add si, 5
            call BYTE_TO_DEC

            mov ax, es
            mov di, offset MCB
            add di, 18
            call WRD_TO_HEX

            mov ax, es:[1]
```

```
            mov di, offset MCB
            add di, 34
            call WRD_TO_HEX

            mov ax, es:[3]
            mov si, offset MCB
            add si, 49
            push es
            mov dx, ds
            mov es, dx
            call BYTE_FUNC
            pop es

            mov bx, 8
            mov cx, 7
            mov si, offset MCB
            add si, 61
            scsd_loop:
                mov dx, es:[bx]
                mov ds:[si], dx
                inc bx
                inc si
                loop scsd_loop

            mov dx, offset MCB
            call PRINT

            pop di
            pop si
            pop cx
            pop dx
            pop bx
            pop ax

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

            mov bx, es:[3]
            mov ax, es
            add ax, bx
            inc ax
            mov es, ax
            inc cl
            jmp check_mcb
      final:
            push es
            push cx
            push bx
            push ax
            ret
mcb_func ENDP
```

```
;----------------------------------
free_memory PROC NEAR
    push ax
    push bx
    push dx

    lea ax, quit_prog
    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_memory ENDP
;----------------------------------
memory_request PROC NEAR
    push ax
    push bx
    push dx

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

    pop dx
    pop bx
    pop ax
    ret
memory_request ENDP
;----------------------------------
BEGIN:
     call memory_func
     call cmos_func
     call free_memory
     call memory_request
     call mcb_func

     xor AL, AL
     mov AH, 4Ch
     int 21h
quit_prog:
TESTPC ENDS
     END START
```

Файл task_4.asm

```
TESTPC SEGMENT
     ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
     ORG  100H
START:    jmp BEGIN
;-------------------------------
MEMORY_SIZE db 'Available memory:          ', 0DH, 0AH, '$'
CMOS_SIZE db 'Extended memory:          ', 0DH, 0AH, '$'
MCB db 'MCB:    | addr:       | owner PSP:      | size:          |
SD/SC:                ', 0DH, 0AH, '$'
MESSAGE db 'Request Failed!', 0DH, 0AH, '$'
;-------------------------------
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-го 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, в 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 10 с/с, в 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_1
      or AL, 30h
      mov [SI], AL
end_1:  pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP
;----------------------
BYTE_FUNC PROC near
      push ax
      push bx
      push dx
      push si

      mov bx, 10h
      mul bx
      mov bx, 0ah
      byte_loop:
          div bx
          add dx, 30h
          mov es:[si], dl
          xor dx, dx
          dec si
          cmp ax, 0000h
             jne byte_loop
        pop si
        pop dx
        pop bx
        pop ax
      ret
BYTE_FUNC ENDP
;----------------------
PRINT PROC near
      push AX
      mov AH, 09h
      int 21h
```

```
      pop AX
      ret
PRINT ENDP
;----------------------
memory_func PROC near
      mov ah, 4ah
      mov bx, 0FFFFh
      int 21h
      mov ax, bx
      mov si, offset MEMORY_SIZE
      add si, 23
      call BYTE_FUNC
      mov dx, offset MEMORY_SIZE
      call PRINT
      ret
memory_func ENDP
;----------------------------------
cmos_func 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, 22
      call BYTE_FUNC
      mov dx, offset CMOS_SIZE
      call PRINT

      pop dx
      pop ax
      ret
cmos_func ENDP
;----------------------------------
mcb_func PROC near
      push ax
      push bx
      push cx
      push es

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

      check_mcb:
```

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

mov al, cl
mov si, offset MCB
add si, 5
call BYTE_TO_DEC

mov ax, es
mov di, offset MCB
add di, 18
call WRD_TO_HEX

mov ax, es:[1]
mov di, offset MCB
add di, 36
call WRD_TO_HEX

mov ax, es:[3]
mov si, offset MCB
add si, 49
push es
mov dx, ds
mov es, dx
call BYTE_FUNC
pop es

mov bx, 8
mov cx, 7
mov si, offset MCB
add si, 61
scsd_loop:
    mov dx, es:[bx]
    mov ds:[si], dx
    inc bx
    inc si
    loop scsd_loop

mov dx, offset MCB
call PRINT

pop di
pop si
pop cx
pop dx
pop bx
pop ax
```

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

            mov bx, es:[3]
            mov ax, es
            add ax, bx
            inc ax
            mov es, ax
            inc cl
            jmp check_mcb
      final:
            push es
            push cx
            push bx
            push ax
            ret
mcb_func ENDP
;-----------------------------------
free_memory PROC NEAR
    push ax
    push bx
    push dx

    lea ax, quit_prog
    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_memory ENDP
;-----------------------------------
memory_request PROC NEAR
    push ax
    push bx
    push dx

    mov bx, 1000h
    mov ah, 48h
    int 21h
    jnc success
    mov dx, offset MESSAGE
    call PRINT
```

```
    success:
         pop dx
         pop bx
         pop ax
         ret
memory_request ENDP
;-----------------------------------
BEGIN:
     call memory_func
     call cmos_func
     call memory_request
     call free_memory
     call mcb_func

     xor AL, AL
     mov AH, 4Ch
     int 21h
quit_prog:
TESTPC ENDS
     END START
```