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 _____ Сергеев Д.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Для исследования организации управления памятью необходимо ориентироваться на тип основной памяти, реализованной в компьютере и способ организации, принятой в ОС. В лабораторной работе рассматривается неограниченная 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, просматривают и преобразуют этот список.

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

Задание.

Шаг 1. Написать и отладить программный модуль .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 3 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.

Выполнение работы.

В ходе работы был взят шаблон из методических материалов, также были написаны процедуры:

- AV_MEM – выводит на консоль размер доступной памяти в байтах
- EX_MEM – выводит на консоль размер расширенной памяти в килобайтах
- REQUEST_MEMORY – запрашивает 64 Кбайта памяти, проверяет флаг CF
- FREE_MEMORY – освобождает память, которая не используется программой
- LIST_MCB – выводит цепочку блоков управления памяти

В ходе выполнения первого шага задания была написана программа:

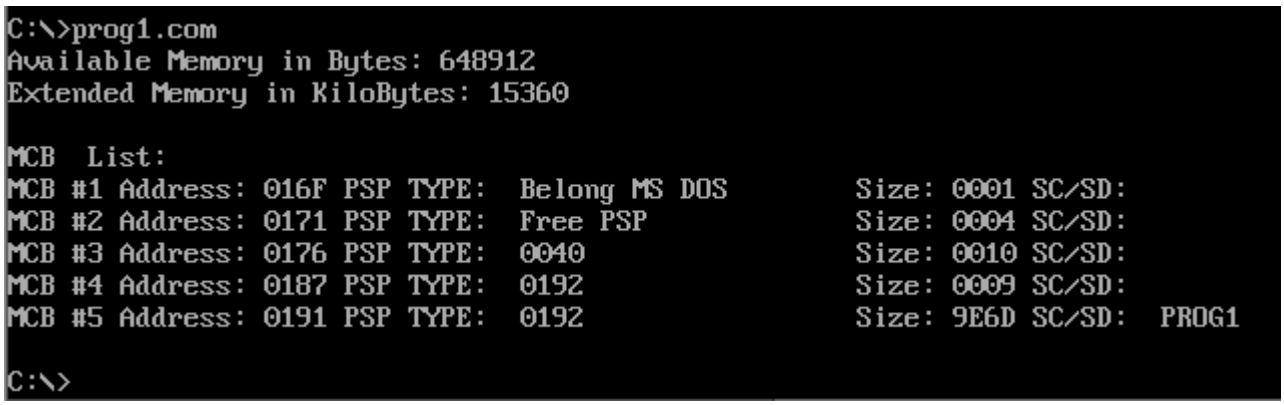

Рисунок 1 – Результат выполнения .COM модуля после первого шага

Как видно на Рисунке 1, написанная программа занимает всю доступную память.

На втором шаге программа была изменена таким образом, чтобы она использовала только необходимую ей память:

Рисунок 2 – Результат выполнения .COM модуля после второго шага

В данном же случае можем заметить, что программа занимает уже ту память, которая ей необходима для работы.

На третьем шаге программа после освобождения также запрашивает 64 Кбайта памяти:

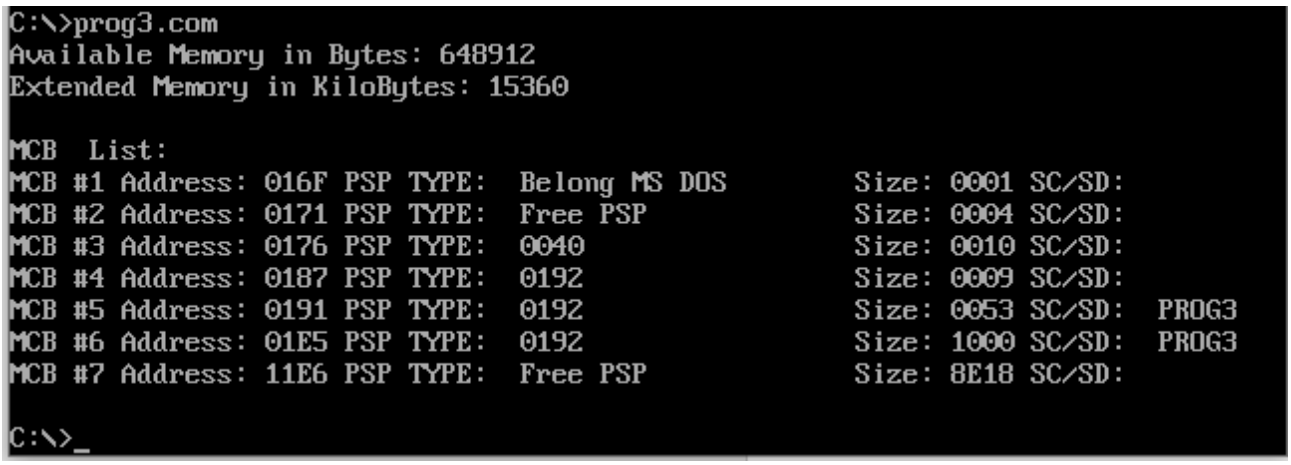

Рисунок 3 - Результат выполнения .COM модуля после третьего шага

На Рисунке 3 можно увидеть, что кроме памяти под собственные нужны, программа также занимает 64 Кбайта памяти, которые были запрошены.

На четвертом шаге, программа запрашивает 64 Кбайта памяти до освобождения памяти:

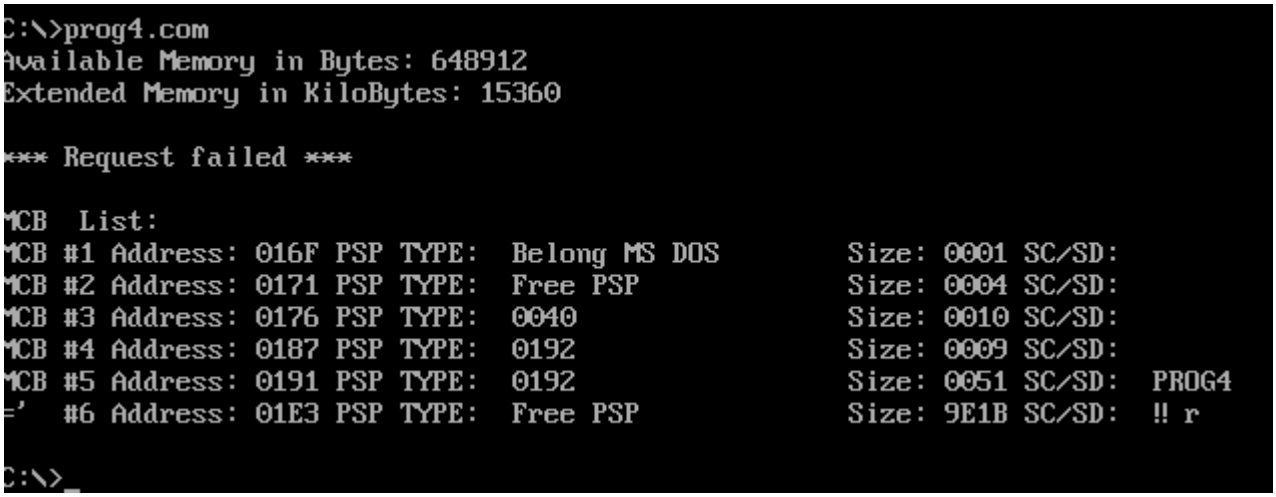

Рисунок 4 – Результат выполнения .COM модуля после четвертого шага

Как видно на Рисунке 4, дополнительная память не была выделена, ведь запрос происходил в тот момент, когда не было доступной памяти.

Исходный программный код смотрите в приложении А.

Контрольные вопросы.

1. Что означает “доступный объём памяти”?

Ответ: объём памяти, который выделяет управляющая программа для использования нашем модулем.

2. Где MCB блок Вашей программы в списке?

Ответ: На первом шаге программа находится на пятом-последнем месте в списке, т.к. занимает всю свободную память. На втором шаге она также находится на пятом месте, но при этом после неё есть ещё один блок MCB, ведь на данном шаге мы освобождаем память. На третьем шаге программа находится 5 и 6 местах, на пятом непосредственно программа, а на шестом – дополнительно выделенная память. На четвертом шаге программа занимает пятое место, т.к. выделить дополнительную память не удалось.

3. Какой размер памяти занимает программа в каждом случае?

Ответ:

1) На первом шаге программа занимает всю доступную память, а именно 648912 байт.
2) На втором шаге программа занимает только необходимую ей память, а именно 1312 байт
3) На третьем шаге программа занимает необходимую ей память + 64 Кбайта, а именно 66864 байта.
4) На четвертом шаге программа занимает 1296 байт, ведь выделить доп. память не удалось.

Выводы.

В результате работы была исследована организация управления памятью. Было рассмотрено устройство нестраничной памяти и способы управления динамическими разделами. Были исследованы структуры данных и работы функций управления памятью ядра операционной системы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: prog1.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN

;---------------------------
AVAILABLE_MEMORY db 'Available Memory in Bytes: $'
EXTENDED_MEMORY db 'Extended Memory in KiloBytes: $'
MCB_LIST db 'MCB  List:', 0DH, 0AH, '$'
MCB_LIST_NUMBER db 'MCB #  $'
MCB_LIST_SIZE db 'Size:      $'
MCB_LIST_ADDRESS db 'Address:      $'
MCB_LIST_SC_SD db 'SC/SD:  $'

PSP_TYPE db 'PSP TYPE:  $'
PSP_FREE db 'Free PSP             $'
PSP_EXCLUDED_HIGH_DRIVER_MEM db'Excluded high driver $'
PSP_OS_XMS_UMB db 'Belongs to OS XMS UMB$'
PSP_MS_DOS db 'Belong MS DOS        $'
PSP_OCCUPIED_386MAX_UMB db 'Occupied mem by 386MAX UMB$'
PSP_BLOCKED_386MAX db 'Blocked 386MAX        $'
PSP_BELONGS_386MAX db 'Belongs 386MAX        $'

DEFAULT_TYPE db '                    $'

;---------------------------
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
    xor AH, AH
    xor DX, DX
    mov CX, 10
loop_bd:
    div CX
    or DL, 30h
    mov [SI], DL
    dec SI
    xor DX, DX
    cmp AX, 10
    jae loop_bd
    cmp AL, 00h
    je end_l
    or AL, 30h
    mov [SI], AL
end_l:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

;-----------------------
PRINT_MESSAGE  PROC  NEAR
    push AX
    mov AH, 9
    int 21h
    pop AX
    ret
PRINT_MESSAGE  ENDP

PRINT_MESSAGE_BYTE  PROC  NEAR
    push AX
    mov AH, 02h
    int 21h
    pop AX
    ret
PRINT_MESSAGE_BYTE  ENDP

PRINT_EOF PROC NEAR
    push AX
    push DX
    mov DL, 0dh ; возврат каретки
```

```
    call PRINT_MESSAGE_BYTE
    mov DL, 0ah ; перед строки
    call PRINT_MESSAGE_BYTE
    pop DX
    pop AX
    ret
PRINT_EOF ENDP

PRINT_AV_MEMORY PROC NEAR
    push AX
    push BX
    push CX
    push DX
    push DI
    xor CX, CX
    mov BX, 010h
    mul BX
    mov DI, DX
    mov BX, 0ah

loop_for_division_1:
    div BX
    push DX
    xor DX, DX
    inc CX
    cmp AX, 0h
    jne loop_for_division_1

loop_for_print_symbol_1:
    pop DX
    add DL, 30h
    call PRINT_MESSAGE_BYTE
    loop loop_for_print_symbol_1

    pop DI
    pop DX
    pop CX
    pop BX
    pop AX
    ret
PRINT_AV_MEMORY ENDP

PRINT_EX_MEMORY PROC NEAR
    push AX
    push BX
    push CX
    push DX
    push DI
    xor CX, CX
    xor DX, DX
    mov BX, 0ah

loop_for_division_2:
    div BX
    push DX
    xor DX, DX
    inc CX
    cmp AX, 0h
```

```
    jne loop_for_division_2

loop_for_print_symbol_2:
    pop DX
    add DL, 30h
    call PRINT_MESSAGE_BYTE
    loop loop_for_print_symbol_2

    pop DI
    pop DX
    pop CX
    pop BX
    pop AX
    ret
PRINT_EX_MEMORY ENDP

;-----------------------
AV_MEM PROC NEAR
    push ax
    push bx
    push dx
    mov dx, offset AVAILABLE_MEMORY
    call PRINT_MESSAGE
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    call PRINT_AV_MEMORY
    call PRINT_EOF
    pop dx
    pop bx
    pop ax
    ret
AV_MEM ENDP

EX_MEM PROC NEAR
    push AX
    push BX
    push DX
    mov AL, 30h
    out 70h, AL
    in AL, 71h
    mov bl, AL
    mov AL, 31h
    out 70h, AL
    in AL, 71h
    mov BH, AL
    mov DX, offset EXTENDED_MEMORY
    call PRINT_MESSAGE
    mov AX, BX
    call PRINT_EX_MEMORY
    call PRINT_EOF
    pop DX
    pop BX
    pop AX
    ret
EX_MEM ENDP
```

```
PRINT_MCB PROC near
    push AX
    push DX
    push DI
    mov DX, offset PSP_TYPE
    call PRINT_MESSAGE
    cmp AX, 0000h
    je print_free
    cmp AX, 0006h
    je print_OS_XMS_UMB
    cmp AX, 0007h
    je print_excluded_high_driver_mem
    cmp AX, 0008h
    je print_MS_DOS
    cmp AX, 0FFFAh
    je print_occupied_386MAX_UMB
    cmp AX, 0FFFDh
    je print_blocked_386MAX
    cmp AX, 0FFFEh
    je print_belongs_386MAX_UMB
    jmp print_default

print_free:
    mov DX, offset PSP_FREE
    call PRINT_MESSAGE
    jmp end_print

print_OS_XMS_UMB:
    mov DX, offset PSP_OS_XMS_UMB
    call PRINT_MESSAGE
    jmp end_print

print_excluded_high_driver_mem:
    mov DX, offset PSP_EXCLUDED_HIGH_DRIVER_MEM
    call PRINT_MESSAGE
    jmp end_print

print_MS_DOS:
    mov DX, offset PSP_MS_DOS
    call PRINT_MESSAGE
    jmp end_print

print_occupied_386MAX_UMB:
    mov DX, offset PSP_OCCUPIED_386MAX_UMB
    call PRINT_MESSAGE
    jmp end_print

print_blocked_386MAX:
    mov DX, offset PSP_BLOCKED_386MAX
    call PRINT_MESSAGE
    jmp end_print

print_belongs_386MAX_UMB:
    mov DX, offset PSP_BELONGS_386MAX
    call PRINT_MESSAGE
    jmp end_print

print_default:
```

```
    mov DI, offset DEFAULT_TYPE
    add DI, 3
    call WRD_TO_HEX
    mov DX, offset DEFAULT_TYPE
    call PRINT_MESSAGE

end_print:
    pop DI
    pop DX
    pop AX
    ret
PRINT_MCB ENDP

LIST_MCB PROC NEAR
    push AX
    push BX
    push DX
    push CX
    push SI
    push DI
    call PRINT_EOF
    mov DX, offset MCB_LIST
    call PRINT_MESSAGE
    mov AH, 52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES, AX
    xor CX, CX
    mov CL, 01h

loop_for_mcb:
    mov AL, CL
    mov SI, offset MCB_LIST_NUMBER
    add SI, 5
    call BYTE_TO_DEC
    mov DX, offset MCB_LIST_NUMBER
    call PRINT_MESSAGE
    mov AX, ES
    mov DI, offset MCB_LIST_ADDRESS
    add DI, 12
    call WRD_TO_HEX
    mov DX, offset MCB_LIST_ADDRESS
    call PRINT_MESSAGE
    mov AX, ES:[1]
    call PRINT_MCB
    mov AX, ES:[3]
    mov DI, offset MCB_LIST_SIZE
    add DI, 9
    call WRD_TO_HEX
    mov DX, offset MCB_LIST_SIZE
    call PRINT_MESSAGE
    mov BX, 8
    mov DX, offset MCB_LIST_SC_SD
    call PRINT_MESSAGE
    push CX
    mov CX, 7

    loop_for_print_sc_sd:
```

```
        mov DL, ES:[BX]
        call PRINT_MESSAGE_BYTE
        inc BX
        loop loop_for_print_sc_sd

    call PRINT_EOF
    pop CX
    mov AH, ES:[0]
    cmp AH, 5ah
    je end_LIST_MCB
    mov BX, ES:[3]
    inc BX
    mov AX, ES
    add AX, BX
    mov ES, AX
    inc CL
    jmp loop_for_mcb

end_LIST_MCB:
    pop DI
    pop SI
    pop CX
    pop DX
    pop BX
    pop AX
    ret
LIST_MCB ENDP

BEGIN:
    call AV_MEM
    call EX_MEM
    call LIST_MCB
    xor AL, AL
    mov AH, 4ch
    int 21h

TESTPC  ENDS
        END START
```

Название файла: prog2.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN

;---------------------------
AVAILABLE_MEMORY db 'Available Memory in Bytes: $'
EXTENDED_MEMORY db 'Extended Memory in KiloBytes: $'
MCB_LIST db 'MCB  List:', 0DH, 0AH, '$'
MCB_LIST_NUMBER db 'MCB #  $'
MCB_LIST_SIZE db 'Size:      $'
MCB_LIST_ADDRESS db 'Address:      $'
MCB_LIST_SC_SD db 'SC/SD:  $'

PSP_TYPE db 'PSP TYPE:  $'
PSP_FREE db 'Free PSP             $'
PSP_EXCLUDED_HIGH_DRIVER_MEM db'Excluded high driver $'
```

```
PSP_OS_XMS_UMB db 'Belongs to OS XMS UMB$'
PSP_MS_DOS db 'Belong MS DOS        $'
PSP_OCCUPIED_386MAX_UMB db 'Occupied mem by 386MAX UMB$'
PSP_BLOCKED_386MAX db 'Blocked 386MAX        $'
PSP_BELONGS_386MAX db 'Belongs 386MAX        $'

DEFAULT_TYPE db '                    $'

;----------------------------
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
    xor AH, AH
    xor DX, DX
    mov CX, 10
loop_bd:
    div CX
    or DL, 30h
    mov [SI], DL
```

```
    dec SI
    xor DX, DX
    cmp AX, 10
    jae loop_bd
    cmp AL, 00h
    je end_l
    or AL, 30h
    mov [SI], AL
end_l:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

;-----------------------
PRINT_MESSAGE  PROC  NEAR
    push AX
    mov AH, 9
    int 21h
    pop AX
    ret
PRINT_MESSAGE  ENDP

PRINT_MESSAGE_BYTE  PROC  NEAR
    push AX
    mov AH, 02h
    int 21h
    pop AX
    ret
PRINT_MESSAGE_BYTE  ENDP

PRINT_EOF PROC NEAR
    push AX
    push DX
    mov DL, 0dh
    call PRINT_MESSAGE_BYTE
    mov DL, 0ah
    call PRINT_MESSAGE_BYTE
    pop DX
    pop AX
    ret
PRINT_EOF ENDP

PRINT_AV_MEMORY PROC NEAR
    push AX
    push BX
    push CX
    push DX
    push DI
    xor CX, CX
    mov BX, 010h
    mul BX
    mov DI, DX
    mov BX, 0ah

loop_for_division_1:
    div BX
    push DX
```

```
    xor DX, DX
    inc CX
    cmp AX, 0h
    jne loop_for_division_1

loop_for_print_symbol_1:
    pop DX
    add DL, 30h
    call PRINT_MESSAGE_BYTE
    loop loop_for_print_symbol_1

    pop DI
    pop DX
    pop CX
    pop BX
    pop AX
    ret
PRINT_AV_MEMORY ENDP

PRINT_EX_MEMORY PROC NEAR
    push AX
    push BX
    push CX
    push DX
    push DI
    xor CX, CX
    xor DX, DX
    mov BX, 0ah

loop_for_division_2:
    div BX
    push DX
    xor DX, DX
    inc CX
    cmp AX, 0h
    jne loop_for_division_2

loop_for_print_symbol_2:
    pop DX
    add DL, 30h
    call PRINT_MESSAGE_BYTE
    loop loop_for_print_symbol_2

    pop DI
    pop DX
    pop CX
    pop BX
    pop AX
    ret
PRINT_EX_MEMORY ENDP

;-----------------------
AV_MEM PROC NEAR
    push ax
    push bx
    push dx
    mov dx, offset AVAILABLE_MEMORY
    call PRINT_MESSAGE
```

```
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    call PRINT_AV_MEMORY
    call PRINT_EOF
    pop dx
    pop bx
    pop ax
    ret
AV_MEM ENDP

EX_MEM PROC NEAR
    push AX
    push BX
    push DX
    mov AL, 30h
    out 70h, AL
    in AL, 71h
    mov bl, AL
    mov AL, 31h
    out 70h, AL
    in AL, 71h
    mov BH, AL
    mov DX, offset EXTENDED_MEMORY
    call PRINT_MESSAGE
    mov AX, BX
    call PRINT_EX_MEMORY
    call PRINT_EOF
    pop DX
    pop BX
    pop AX
    ret
EX_MEM ENDP

PRINT_MCB PROC near
    push AX
    push DX
    push DI
    mov DX, offset PSP_TYPE
    call PRINT_MESSAGE
    cmp AX, 0000h
    je print_free
    cmp AX, 0006h
    je print_OS_XMS_UMB
    cmp AX, 0007h
    je print_excluded_high_driver_mem
    cmp AX, 0008h
    je print_MS_DOS
    cmp AX, 0FFFAh
    je print_occupied_386MAX_UMB
    cmp AX, 0FFFDh
    je print_blocked_386MAX
    cmp AX, 0FFFEh
    je print_belongs_386MAX_UMB
    jmp print_default

print_free:
```

```
    mov DX, offset PSP_FREE
    call PRINT_MESSAGE
    jmp end_print

print_OS_XMS_UMB:
    mov DX, offset PSP_OS_XMS_UMB
    call PRINT_MESSAGE
    jmp end_print

print_excluded_high_driver_mem:
    mov DX, offset PSP_EXCLUDED_HIGH_DRIVER_MEM
    call PRINT_MESSAGE
    jmp end_print

print_MS_DOS:
    mov DX, offset PSP_MS_DOS
    call PRINT_MESSAGE
    jmp end_print

print_occupied_386MAX_UMB:
    mov DX, offset PSP_OCCUPIED_386MAX_UMB
    call PRINT_MESSAGE
    jmp end_print

print_blocked_386MAX:
    mov DX, offset PSP_BLOCKED_386MAX
    call PRINT_MESSAGE
    jmp end_print

print_belongs_386MAX_UMB:
    mov DX, offset PSP_BELONGS_386MAX
    call PRINT_MESSAGE
    jmp end_print

print_default:
    mov DI, offset DEFAULT_TYPE
    add DI, 3
    call WRD_TO_HEX
    mov DX, offset DEFAULT_TYPE
    call PRINT_MESSAGE

end_print:
    pop DI
    pop DX
    pop AX
    ret
PRINT_MCB ENDP

LIST_MCB PROC NEAR
    push AX
    push BX
    push DX
    push CX
    push SI
    push DI
    call PRINT_EOF
    mov DX, offset MCB_LIST
    call PRINT_MESSAGE
```

```
    mov AH, 52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES, AX
    xor CX, CX
    mov CL, 01h

loop_for_mcb:
    mov AL, CL
    mov SI, offset MCB_LIST_NUMBER
    add SI, 5
    call BYTE_TO_DEC
    mov DX, offset MCB_LIST_NUMBER
    call PRINT_MESSAGE
    mov AX, ES
    mov DI, offset MCB_LIST_ADDRESS
    add DI, 12
    call WRD_TO_HEX
    mov DX, offset MCB_LIST_ADDRESS
    call PRINT_MESSAGE
    mov AX, ES:[1]
    call PRINT_MCB
    mov AX, ES:[3]
    mov DI, offset MCB_LIST_SIZE
    add DI, 9
    call WRD_TO_HEX
    mov DX, offset MCB_LIST_SIZE
    call PRINT_MESSAGE
    mov BX, 8
    mov DX, offset MCB_LIST_SC_SD
    call PRINT_MESSAGE
    push CX
    mov CX, 7

    loop_for_print_sc_sd:
        mov DL, ES:[BX]
        call PRINT_MESSAGE_BYTE
        inc BX
        loop loop_for_print_sc_sd

    call PRINT_EOF
    pop CX
    mov AH, ES:[0]
    cmp AH, 5ah
    je end_LIST_MCB
    mov BX, ES:[3]
    inc BX
    mov AX, ES
    add AX, BX
    mov ES, AX
    inc CL
    jmp loop_for_mcb

end_LIST_MCB:
    pop DI
    pop SI
    pop CX
    pop DX
```

```
    pop BX
    pop AX
    ret
LIST_MCB ENDP

FREE_MEMORY PROC NEAR
    push AX
    push BX
    push DX
    xor DX, DX
    lea AX, end_programm
    mov BX, 10h
    div BX
    add AX, DX
    mov BX,AX
    xor AX, AX
    mov AH,4Ah
    int 21h
    pop DX
    pop BX
    pop AX
    ret
FREE_MEMORY ENDP

BEGIN:
    mov AH, 4ah
    mov BX, 0ffffh
    int 21h
    call AV_MEM
    call EX_MEM
    call FREE_MEMORY
    call LIST_MCB
    xor AL, AL
    mov AH, 4ch
    int 21h
end_programm:
TESTPC  ENDS
        END START
```

Название файла: prog3.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN

;---------------------------
AVAILABLE_MEMORY db 'Available Memory in Bytes: $'
EXTENDED_MEMORY db 'Extended Memory in KiloBytes: $'
MCB_LIST db 'MCB  List:', 0DH, 0AH, '$'
MCB_LIST_NUMBER db 'MCB #  $'
MCB_LIST_SIZE db 'Size:      $'
MCB_LIST_ADDRESS db 'Address:      $'
MCB_LIST_SC_SD db 'SC/SD:  $'

PSP_TYPE db 'PSP TYPE:  $'
PSP_FREE db 'Free PSP             $'
PSP_EXCLUDED_HIGH_DRIVER_MEM db'Excluded high driver $'
```

```
PSP_OS_XMS_UMB db 'Belongs to OS XMS UMB$'
PSP_MS_DOS db 'Belong MS DOS        $'
PSP_OCCUPIED_386MAX_UMB db 'Occupied mem by 386MAX UMB$'
PSP_BLOCKED_386MAX db 'Blocked 386MAX        $'
PSP_BELONGS_386MAX db 'Belongs 386MAX        $'

DEFAULT_TYPE db '                    $'

;----------------------------
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
    xor AH, AH
    xor DX, DX
    mov CX, 10
loop_bd:
    div CX
    or DL, 30h
    mov [SI], DL
```

```
    dec SI
    xor DX, DX
    cmp AX, 10
    jae loop_bd
    cmp AL, 00h
    je end_l
    or AL, 30h
    mov [SI], AL
end_l:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

;-----------------------
PRINT_MESSAGE  PROC  NEAR
    push AX
    mov AH, 9
    int 21h
    pop AX
    ret
PRINT_MESSAGE  ENDP

PRINT_MESSAGE_BYTE  PROC  NEAR
    push AX
    mov AH, 02h
    int 21h
    pop AX
    ret
PRINT_MESSAGE_BYTE  ENDP

PRINT_EOF PROC NEAR
    push AX
    push DX
    mov DL, 0dh
    call PRINT_MESSAGE_BYTE
    mov DL, 0ah
    call PRINT_MESSAGE_BYTE
    pop DX
    pop AX
    ret
PRINT_EOF ENDP

PRINT_AV_MEMORY PROC NEAR
    push AX
    push BX
    push CX
    push DX
    push DI
    xor CX, CX
    mov BX, 010h
    mul BX
    mov DI, DX
    mov BX, 0ah

loop_for_division_1:
    div BX
    push DX
```

```
    xor DX, DX
    inc CX
    cmp AX, 0h
    jne loop_for_division_1

loop_for_print_symbol_1:
    pop DX
    add DL, 30h
    call PRINT_MESSAGE_BYTE
    loop loop_for_print_symbol_1

    pop DI
    pop DX
    pop CX
    pop BX
    pop AX
    ret
PRINT_AV_MEMORY ENDP

PRINT_EX_MEMORY PROC NEAR
    push AX
    push BX
    push CX
    push DX
    push DI
    xor CX, CX
    xor DX, DX
    mov BX, 0ah

loop_for_division_2:
    div BX
    push DX
    xor DX, DX
    inc CX
    cmp AX, 0h
    jne loop_for_division_2

loop_for_print_symbol_2:
    pop DX
    add DL, 30h
    call PRINT_MESSAGE_BYTE
    loop loop_for_print_symbol_2

    pop DI
    pop DX
    pop CX
    pop BX
    pop AX
    ret
PRINT_EX_MEMORY ENDP

;-----------------------
AV_MEM PROC NEAR
    push ax
    push bx
    push dx
    mov dx, offset AVAILABLE_MEMORY
    call PRINT_MESSAGE
```

```
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    call PRINT_AV_MEMORY
    call PRINT_EOF
    pop dx
    pop bx
    pop ax
    ret
AV_MEM ENDP

 EX_MEM PROC NEAR
    push AX
    push BX
    push DX
    mov AL, 30h
    out 70h, AL
    in AL, 71h
    mov bl, AL
    mov AL, 31h
    out 70h, AL
    in AL, 71h
    mov BH, AL
    mov DX, offset EXTENDED_MEMORY
    call PRINT_MESSAGE
    mov AX, BX
    call PRINT_EX_MEMORY
    call PRINT_EOF
    pop DX
    pop BX
    pop AX
    ret
EX_MEM ENDP

PRINT_MCB PROC near
    push AX
    push DX
    push DI
    mov DX, offset PSP_TYPE
    call PRINT_MESSAGE
    cmp AX, 0000h
    je print_free
    cmp AX, 0006h
    je print_OS_XMS_UMB
    cmp AX, 0007h
    je print_excluded_high_driver_mem
    cmp AX, 0008h
    je print_MS_DOS
    cmp AX, 0FFFAh
    je print_occupied_386MAX_UMB
    cmp AX, 0FFFDh
    je print_blocked_386MAX
    cmp AX, 0FFFEh
    je print_belongs_386MAX_UMB
    jmp print_default

print_free:
```

```
    mov DX, offset PSP_FREE
    call PRINT_MESSAGE
    jmp end_print

print_OS_XMS_UMB:
    mov DX, offset PSP_OS_XMS_UMB
    call PRINT_MESSAGE
    jmp end_print

print_excluded_high_driver_mem:
    mov DX, offset PSP_EXCLUDED_HIGH_DRIVER_MEM
    call PRINT_MESSAGE
    jmp end_print

print_MS_DOS:
    mov DX, offset PSP_MS_DOS
    call PRINT_MESSAGE
    jmp end_print

print_occupied_386MAX_UMB:
    mov DX, offset PSP_OCCUPIED_386MAX_UMB
    call PRINT_MESSAGE
    jmp end_print

print_blocked_386MAX:
    mov DX, offset PSP_BLOCKED_386MAX
    call PRINT_MESSAGE
    jmp end_print

print_belongs_386MAX_UMB:
    mov DX, offset PSP_BELONGS_386MAX
    call PRINT_MESSAGE
    jmp end_print

print_default:
    mov DI, offset DEFAULT_TYPE
    add DI, 3
    call WRD_TO_HEX
    mov DX, offset DEFAULT_TYPE
    call PRINT_MESSAGE

end_print:
    pop DI
    pop DX
    pop AX
    ret
PRINT_MCB ENDP

LIST_MCB PROC NEAR
    push AX
    push BX
    push DX
    push CX
    push SI
    push DI
    call PRINT_EOF
    mov DX, offset MCB_LIST
    call PRINT_MESSAGE
```

```
    mov AH, 52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES, AX
    xor CX, CX
    mov CL, 01h

loop_for_mcb:
    mov AL, CL
    mov SI, offset MCB_LIST_NUMBER
    add SI, 5
    call BYTE_TO_DEC
    mov DX, offset MCB_LIST_NUMBER
    call PRINT_MESSAGE
    mov AX, ES
    mov DI, offset MCB_LIST_ADDRESS
    add DI, 12
    call WRD_TO_HEX
    mov DX, offset MCB_LIST_ADDRESS
    call PRINT_MESSAGE
    mov AX, ES:[1]
    call PRINT_MCB
    mov AX, ES:[3]
    mov DI, offset MCB_LIST_SIZE
    add DI, 9
    call WRD_TO_HEX
    mov DX, offset MCB_LIST_SIZE
    call PRINT_MESSAGE
    mov BX, 8
    mov DX, offset MCB_LIST_SC_SD
    call PRINT_MESSAGE
    push CX
    mov CX, 7

    loop_for_print_sc_sd:
        mov DL, ES:[BX]
        call PRINT_MESSAGE_BYTE
        inc BX
        loop loop_for_print_sc_sd

    call PRINT_EOF
    pop CX
    mov AH, ES:[0]
    cmp AH, 5ah
    je end_LIST_MCB
    mov BX, ES:[3]
    inc BX
    mov AX, ES
    add AX, BX
    mov ES, AX
    inc CL
    jmp loop_for_mcb

end_LIST_MCB:
    pop DI
    pop SI
    pop CX
    pop DX
```

```
    pop BX
    pop AX
    ret
LIST_MCB ENDP

REQUEST_MEMORY PROC NEAR
    push AX
    push BX
    push DX
    mov BX, 1000h
    mov AH, 48h
    int 21h
    pop DX
    pop BX
    pop AX
REQUEST_MEMORY ENDP

FREE_MEMORY PROC NEAR
    push AX
    push BX
    push DX
    xor DX, DX
    lea AX, end_programm
    mov BX, 10h
    div BX
    add AX, DX
    mov BX,AX
    xor AX, AX
    mov AH,4Ah
    int 21h
    pop DX
    pop BX
    pop AX
    ret
FREE_MEMORY ENDP

BEGIN:
    mov AH, 4ah
    mov BX, 0ffffh
    int 21h
    call AV_MEM
    call EX_MEM
    call FREE_MEMORY
    call REQUEST_MEMORY
    call LIST_MCB
    xor AL, AL
    mov AH, 4ch
    int 21h
end_programm:
TESTPC  ENDS
        END START
```

Название файла: prog4.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
```

```
;---------------------------
AVAILABLE_MEMORY db 'Available Memory in Bytes: $'
EXTENDED_MEMORY db 'Extended Memory in KiloBytes: $'
MCB_LIST db 'MCB  List:', 0DH, 0AH, '$'
MCB_LIST_NUMBER db 'MCB #  $'
MCB_LIST_SIZE db 'Size:      $'
MCB_LIST_ADDRESS db 'Address:      $'
MCB_LIST_SC_SD db 'SC/SD:  $'

PSP_TYPE db 'PSP TYPE:  $'
PSP_FREE db 'Free PSP             $'
PSP_EXCLUDED_HIGH_DRIVER_MEM db'Excluded high driver $'
PSP_OS_XMS_UMB db 'Belongs to OS XMS UMB$'
PSP_MS_DOS db 'Belong MS DOS        $'
PSP_OCCUPIED_386MAX_UMB db 'Occupied mem by 386MAX UMB$'
PSP_BLOCKED_386MAX db 'Blocked 386MAX        $'
PSP_BELONGS_386MAX db 'Belongs 386MAX        $'

DEFAULT_TYPE db '                     $'

REQUEST_FAILED db '*** Request failed ***$'
REQUEST_SUCCESS db '*** Request success ***$'

;---------------------------
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
    xor AH, AH
    xor DX, DX
    mov CX, 10
loop_bd:
    div CX
    or DL, 30h
    mov [SI], DL
    dec SI
    xor DX, DX
    cmp AX, 10
    jae loop_bd
    cmp AL, 00h
    je end_l
    or AL, 30h
    mov [SI], AL
end_l:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

;-----------------------
PRINT_MESSAGE  PROC  NEAR
    push AX
    mov AH, 9
    int 21h
    pop AX
    ret
PRINT_MESSAGE  ENDP

PRINT_MESSAGE_BYTE  PROC  NEAR
    push AX
    mov AH, 02h
    int 21h
    pop AX
    ret
PRINT_MESSAGE_BYTE  ENDP

PRINT_EOF PROC NEAR
    push AX
    push DX
    mov DL, 0dh
    call PRINT_MESSAGE_BYTE
    mov DL, 0ah
    call PRINT_MESSAGE_BYTE
    pop DX
    pop AX
    ret
PRINT_EOF ENDP
```

```
PRINT_AV_MEMORY PROC NEAR
    push AX
    push BX
    push CX
    push DX
    push DI
    xor CX, CX
    mov BX, 010h
    mul BX
    mov DI, DX
    mov BX, 0ah

loop_for_division_1:
    div BX
    push DX
    xor DX, DX
    inc CX
    cmp AX, 0h
    jne loop_for_division_1

loop_for_print_symbol_1:
    pop DX
    add DL, 30h
    call PRINT_MESSAGE_BYTE
    loop loop_for_print_symbol_1

    pop DI
    pop DX
    pop CX
    pop BX
    pop AX
    ret
PRINT_AV_MEMORY ENDP

PRINT_EX_MEMORY PROC NEAR
    push AX
    push BX
    push CX
    push DX
    push DI
    xor CX, CX
    xor DX, DX
    mov BX, 0ah

loop_for_division_2:
    div BX
    push DX
    xor DX, DX
    inc CX
    cmp AX, 0h
    jne loop_for_division_2

loop_for_print_symbol_2:
    pop DX
    add DL, 30h
    call PRINT_MESSAGE_BYTE
    loop loop_for_print_symbol_2
```

```
    pop DI
    pop DX
    pop CX
    pop BX
    pop AX
    ret
PRINT_EX_MEMORY ENDP

;-----------------------
AV_MEM PROC NEAR
    push ax
    push bx
    push dx
    mov dx, offset AVAILABLE_MEMORY
    call PRINT_MESSAGE
    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx
    call PRINT_AV_MEMORY
    call PRINT_EOF
    pop dx
    pop bx
    pop ax
    ret
AV_MEM ENDP

EX_MEM PROC NEAR
    push AX
    push BX
    push DX
    mov AL, 30h
    out 70h, AL
    in AL, 71h
    mov bl, AL
    mov AL, 31h
    out 70h, AL
    in AL, 71h
    mov BH, AL
    mov DX, offset EXTENDED_MEMORY
    call PRINT_MESSAGE
    mov AX, BX
    call PRINT_EX_MEMORY
    call PRINT_EOF
    pop DX
    pop BX
    pop AX
    ret
EX_MEM ENDP

PRINT_MCB PROC near
    push AX
    push DX
    push DI
    mov DX, offset PSP_TYPE
    call PRINT_MESSAGE
    cmp AX, 0000h
```

```
    je print_free
    cmp AX, 0006h
    je print_OS_XMS_UMB
    cmp AX, 0007h
    je print_excluded_high_driver_mem
    cmp AX, 0008h
    je print_MS_DOS
    cmp AX, 0FFFAh
    je print_occupied_386MAX_UMB
    cmp AX, 0FFFDh
    je print_blocked_386MAX
    cmp AX, 0FFFEh
    je print_belongs_386MAX_UMB
    jmp print_default

print_free:
    mov DX, offset PSP_FREE
    call PRINT_MESSAGE
    jmp end_print

print_OS_XMS_UMB:
    mov DX, offset PSP_OS_XMS_UMB
    call PRINT_MESSAGE
    jmp end_print

print_excluded_high_driver_mem:
    mov DX, offset PSP_EXCLUDED_HIGH_DRIVER_MEM
    call PRINT_MESSAGE
    jmp end_print

print_MS_DOS:
    mov DX, offset PSP_MS_DOS
    call PRINT_MESSAGE
    jmp end_print

print_occupied_386MAX_UMB:
    mov DX, offset PSP_OCCUPIED_386MAX_UMB
    call PRINT_MESSAGE
    jmp end_print

print_blocked_386MAX:
    mov DX, offset PSP_BLOCKED_386MAX
    call PRINT_MESSAGE
    jmp end_print

print_belongs_386MAX_UMB:
    mov DX, offset PSP_BELONGS_386MAX
    call PRINT_MESSAGE
    jmp end_print

print_default:
    mov DI, offset DEFAULT_TYPE
    add DI, 3
    call WRD_TO_HEX
    mov DX, offset DEFAULT_TYPE
    call PRINT_MESSAGE

end_print:
```

```
    pop DI
    pop DX
    pop AX
    ret
PRINT_MCB ENDP

LIST_MCB PROC NEAR
    push AX
    push BX
    push DX
    push CX
    push SI
    push DI
    call PRINT_EOF
    mov DX, offset MCB_LIST
    call PRINT_MESSAGE
    mov AH, 52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES, AX
    xor CX, CX
    mov CL, 01h

loop_for_mcb:
    mov AL, CL
    mov SI, offset MCB_LIST_NUMBER
    add SI, 5
    call BYTE_TO_DEC
    mov DX, offset MCB_LIST_NUMBER
    call PRINT_MESSAGE
    mov AX, ES
    mov DI, offset MCB_LIST_ADDRESS
    add DI, 12
    call WRD_TO_HEX
    mov DX, offset MCB_LIST_ADDRESS
    call PRINT_MESSAGE
    mov AX, ES:[1]
    call PRINT_MCB
    mov AX, ES:[3]
    mov DI, offset MCB_LIST_SIZE
    add DI, 9
    call WRD_TO_HEX
    mov DX, offset MCB_LIST_SIZE
    call PRINT_MESSAGE
    mov BX, 8
    mov DX, offset MCB_LIST_SC_SD
    call PRINT_MESSAGE
    push CX
    mov CX, 7

    loop_for_print_sc_sd:
        mov DL, ES:[BX]
        call PRINT_MESSAGE_BYTE
        inc BX
        loop loop_for_print_sc_sd

    call PRINT_EOF
    pop CX
```

```
    mov AH, ES:[0]
    cmp AH, 5ah
    je end_LIST_MCB
    mov BX, ES:[3]
    inc BX
    mov AX, ES
    add AX, BX
    mov ES, AX
    inc CL
    jmp loop_for_mcb

end_LIST_MCB:
    pop DI
    pop SI
    pop CX
    pop DX
    pop BX
    pop AX
    ret
LIST_MCB ENDP

REQUEST_MEMORY PROC NEAR
    push AX
    push BX
    push DX
    mov BX, 1000h
    mov AH, 48h
    int 21h
    call PRINT_EOF
    jc fail
    jne succes

fail:
    mov DX, offset REQUEST_FAILED
    call PRINT_MESSAGE
    jmp request_end

succes:
    mov DX, offset REQUEST_SUCCESS
    call PRINT_MESSAGE

request_end:
    call PRINT_EOF
    pop DX
    pop BX
    pop AX
    ret
REQUEST_MEMORY ENDP

FREE_MEMORY PROC NEAR
    push AX
    push BX
    push DX
    xor DX, DX
    lea AX, end_programm
    mov BX, 10h
    div BX
    add AX, DX
```

```
    mov BX,AX
    xor AX, AX
    mov AH,4Ah
    int 21h
    pop DX
    pop BX
    pop AX
    ret
FREE_MEMORY ENDP

BEGIN:
    mov AH, 4ah
    mov BX, 0ffffh
    int 21h
    call AV_MEM
    call EX_MEM
    call REQUEST_MEMORY
    call FREE_MEMORY
    call LIST_MCB
    xor AL, AL
    mov AH, 4ch
    int 21h
end_programm:
TESTPC  ENDS
        END START
```