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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Операционные системы»

Тема: Исследование организации управления основной памятью.

Студент гр. 9383 _____ Гладких А.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

ПОСТАНОВКА ЗАДАЧИ

Цель работы.

Исследование организации управления основной памятью, изучение нестраничной памяти, исследование структур данных и работы функций управления памятью ядра операционной системы.

Сведения о функциях и структурах данных управляющей программы.

Процедура	Описание
TETR_TO_HEX	Перевод десятичной цифры в код символа ASCII
BYTE_TO_HEX	Перевод байта в AL в два символа шестн. числа AX
WRD_TO_HEX	Перевод в 16 с/с 16-ти разрядного числа
BYTE_TO_DEC	Перевод в 10 с/с, SI – адрес поля младшей цифры
WRITEWRD	Функция печати строки на экран
WRITEBYTE	Функция печати символа на экран
ENDLINE	Функция печати символов переноса строки
TASK_A	Вывод количества доступной памяти
TASK_B	Вывод размера расширенной памяти
TASK_C	Вывод цепочки блоков управления памятью
FREE_UNUSED_MEMORY	Освобождение неиспользуемой памяти
ALLOC_MEMORY	Выделение блоков памяти

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. 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.

Исходный код.

Исходный код представлен в приложении А.

РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

Шаг 1. Был написан и отлажен .COM модуль, который выводит на экран требуемую в задании информацию.

```
D:\>lab_1.com
Available Memory (bytes):648912
Extended Memory (kbytes):15360

MCB Table:
MCB #1 Address: 016F  PSP TYPE: belongs MSDOS       Size: 0001  SC/SD:
MCB #2 Address: 0171  PSP TYPE: free area           Size: 0004  SC/SD: DPMILOA
MCB #3 Address: 0176  PSP TYPE: 0040                Size: 0010  SC/SD:
MCB #4 Address: 0187  PSP TYPE: 0192                Size: 0009  SC/SD:
MCB #5 Address: 0191  PSP TYPE: 0192                Size: 9E6D  SC/SD: LAB_1
```

Рисунок 1 - Иллюстрация работы .COM-модуля

Шаг 2. Был написан и отлажен .COM модуль, который выводит на экран требуемую в задании информацию, а также высвобождает память, которую не занимает, с помощью функции 4Ah прерывания 21h. Было произведено сравнение выходных данных с результатами, полученными на предыдущем шаге.

```
D:\>lab_2.com
Available Memory (bytes):648912
Extended Memory (kbytes):15360

MCB Table:
MCB #1 Address: 016F  PSP TYPE: belongs MSDOS       Size: 0001  SC/SD:
MCB #2 Address: 0171  PSP TYPE: free area           Size: 0004  SC/SD: DPMILOA
MCB #3 Address: 0176  PSP TYPE: 0040                Size: 0010  SC/SD:
MCB #4 Address: 0187  PSP TYPE: 0192                Size: 0009  SC/SD:
MCB #5 Address: 0191  PSP TYPE: 0192                Size: 004A  SC/SD: LAB_2
MCB #6 Address: 01DC  PSP TYPE: free area           Size: 9E22  SC/SD: ♥ú♥‼♥≈♥
```

Рисунок 2 - Иллюстрация работы .COM-модуля

Шаг 3. Был написан и отлажен .COM модуль, который выводит на экран требуемую в задании информацию, а также высвобождает память, которую не занимает, с помощью функции 4Ah прерывания 21h. Также было сделано так, чтобы после освобождения памяти, программа запрашивала 64Кб памяти

функцией 48H прерывания 21H. Было произведено сравнение выходных данных с результатами, полученными на предыдущем шаге.

```
D:\>lab_3.com
Available Memory (bytes):648912
Extended Memory (kbytes):15360

MCB Table:
MCB #1 Address: 016F  PSP TYPE: belongs MSDOS       Size: 0001  SC/SD:
MCB #2 Address: 0171  PSP TYPE: free area           Size: 0004  SC/SD: DPMILOA
MCB #3 Address: 0176  PSP TYPE: 0040                Size: 0010  SC/SD:
MCB #4 Address: 0187  PSP TYPE: 0192                Size: 0009  SC/SD:
MCB #5 Address: 0191  PSP TYPE: 0192                Size: 004B  SC/SD: LAB_3
MCB #6 Address: 01DD  PSP TYPE: 0192                Size: 1000  SC/SD: LAB_3
MCB #7 Address: 11DE  PSP TYPE: free area           Size: 8E20  SC/SD: >c>
```

Рисунок 3 - Иллюстрация работы .COM-модуля

Шаг 4. Был изменен первоначальный вариант программы, который запрашивает 64Кб памяти функцией 48H прерывания 21H до освобождения памяти. Было произведено сравнение выходных данных с результатами, полученными на предыдущем шаге.

```
D:\>lab_4.com
Available Memory (bytes):648912
Extended Memory (kbytes):15360

! Allocation failed !

MCB Table:
MCB #1 Address: 016F  PSP TYPE: belongs MSDOS       Size: 0001  SC/SD:
MCB #2 Address: 0171  PSP TYPE: free area           Size: 0004  SC/SD: DPMILOA
MCB #3 Address: 0176  PSP TYPE: 0040                Size: 0010  SC/SD:
MCB #4 Address: 0187  PSP TYPE: 0192                Size: 0009  SC/SD:
MCB #5 Address: 0191  PSP TYPE: 0192                Size: 0056  SC/SD: LAB_4
MCB #6 Address: 01E8  PSP TYPE: free area           Size: 9E16  SC/SD: e to in
```

Рисунок 4 - Иллюстрация работы .COM-модуля

Шаг 5. Была произведена оценка результатов и были отвечены контрольные вопросы:

1. ***Что означает "доступный объем памяти"?***

   Ответ: это объем памяти, который выделяется управляющей программой самой программе на использование.

2. ***Где MCB блок Вашей программы в списке?***

Ответ: ответить на данный вопрос можно исходя из имени владельца блока в таблице MCB. На рисунке 1 и 2 блок моей программы — пятый. На рисунке 3 — пятый и специально выделенный шестой блок. На рисунке 4 блок моей программы — пятый в списке.

3. ***Какой размер памяти занимает программа в каждом случае?***

Ответ: в первом случае — программа занимает всю доступную память.
Во втором случае — только то, сколько ей действительно нужно — 1184 байт.
В третьем случае сама программа занимает 1200 байт и еще дополнительно выделены 65536 байт, то есть в сумме 66736 байт.
В четвертом случае выделить память не получилось, поэтому программа занимает только то, сколько ей нужно — 1376 байт.

Выводы.

Была исследована организация управления основной памятью, была изучена нестраничная память и были исследованы структуры данных и работы функций управления памятью ядра операционной системы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab_1.asm

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

start: jmp begin

AVAILABLE_MEM db 'Available Memory (bytes):$'
EXTENDED_MEM db 'Extended Memory (kbytes):$'
MCB_TABLE db 'MCB Table:', 0DH, 0AH, '$'
MCB_TABLE_NUMBER db 'MCB #  $'

MCB_ADDRESS db 'Address:      $'

PSP_TYPE db ' PSP TYPE: $'
PSP_FREE_AREA db 'free area            $'
PSP_OS_XMS_UMB db 'belongs to OS XMS UMB$'
PSP_RESERVED_FOR_DRIVERS db'reserved for drivers $'
PSP_BELONGS_MSDOS db 'belongs MSDOS        $'
PSP_BUSY_386MAX_UMB db 'busy with block 386MAX UMB$'
PSP_BLOCKED_386MAX db 'blocked 386MAX        $'
PSP_BELONGS_386MAX db 'belongs 386MAX        $'

DEFAULT_PSP_TYPE db '                     $'

MCB_TABLE_SIZE db 'Size:      $'

MCB_SC_SD db ' SC/SD: $'

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

WRITEWRD  PROC  NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEWRD  ENDP

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

ENDLINE PROC NEAR
    push ax
    push dx

    mov dl, 0dh
    call WRITEBYTE
```

```
    mov dl, 0ah
    call WRITEBYTE

    pop dx
    pop ax
    ret
ENDLINE ENDP

WRITE_AVAILABLE_MEM 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

division_loop_av:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_av

print_symbol_loop_av:
    pop dx

    add dl, 30h

    call WRITEBYTE

    loop print_symbol_loop_av

    pop di
```

```
    pop dx
    pop cx
    pop bx
    pop ax
    ret

WRITE_AVAILABLE_MEM ENDP

WRITE_EXTENDED_MEM PROC NEAR
    push ax
    push bx
    push cx
    push dx
    push di

    xor cx, cx
    xor dx, dx

    mov bx, 0ah

division_loop_ext:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_ext

print_symbol_loop_ext:
    pop dx

    add dl, 30h

    call WRITEBYTE

    loop print_symbol_loop_ext

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

```
    ret

WRITE_EXTENDED_MEM ENDP

TASK_A PROC NEAR

    push ax
    push bx
    push dx

    mov dx, offset AVAILABLE_MEM
    call WRITEWRD

    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx

    call WRITE_AVAILABLE_MEM
    call ENDLINE


    pop dx
    pop bx
    pop ax
    ret

TASK_A ENDP

TASK_B 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_MEM
    call WRITEWRD

    mov ax, bx
    call WRITE_EXTENDED_MEM
    call ENDLINE

    pop dx
    pop bx
    pop ax
    ret

TASK_B ENDP

MCB_PSP_TYPE proc near
    push ax
    push dx
    push di

    ;mov dx, offset TAB
    ;call PRINT_BUF

    mov dx, offset PSP_TYPE
    call WRITEWRD

    cmp ax, 0000h
    je print_free_area

    cmp ax, 0006h
    je print_belongs_OS_XMS_UMB

    cmp ax, 0007h
    je print_reserved_for_drivers

    cmp ax, 0008h
    je print_belongs_MS_DOS

    cmp ax, 0FFFAh
```

```
    je print_busy_386MAX_UMB

    cmp ax, 0FFFDh
    je print_blocked_386MAX

    cmp ax, 0FFFEh
    je print_belongs_386MAX_UMB

    jmp print_psp_type

print_free_area:
    mov dx, offset PSP_FREE_AREA
    call WRITEWRD
    jmp end_psp_type

print_belongs_OS_XMS_UMB:
    mov dx, offset PSP_OS_XMS_UMB
    call WRITEWRD
    jmp end_psp_type

print_reserved_for_drivers:
    mov dx, offset PSP_RESERVED_FOR_DRIVERS
    call WRITEWRD
    jmp end_psp_type

print_belongs_MS_DOS:
    mov dx, offset PSP_BELONGS_MSDOS
    call WRITEWRD
    jmp end_psp_type

print_busy_386MAX_UMB:
    mov dx, offset PSP_BUSY_386MAX_UMB
    call WRITEWRD
    jmp end_psp_type

print_blocked_386MAX:
    mov dx, offset PSP_BLOCKED_386MAX
    call WRITEWRD
    jmp end_psp_type

print_belongs_386MAX_UMB:
    mov dx, offset PSP_BELONGS_386MAX
```

```
    call WRITEWRD
    jmp end_psp_type

print_psp_type:
    mov di, offset DEFAULT_PSP_TYPE
    add di, 3
    call WRD_TO_HEX
    mov dx, offset DEFAULT_PSP_TYPE
    call WRITEWRD

end_psp_type:
    pop di
    pop dx
    pop ax
    ret
MCB_PSP_TYPE endp

TASK_C PROC NEAR


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

    call ENDLINE
    mov dx, offset MCB_TABLE
    call WRITEWRD


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

    xor cx, cx
    mov cl, 01h

mcb_for_each_loop:
    mov al, cl
```

```
mov si, offset MCB_TABLE_NUMBER
add si, 5

call BYTE_TO_DEC

mov dx, offset MCB_TABLE_NUMBER
call WRITEWRD

mov ax, es
mov di, offset MCB_ADDRESS
add di, 12
call WRD_TO_HEX

mov dx, offset MCB_ADDRESS
call WRITEWRD

mov ax, es:[1]

call MCB_PSP_TYPE

mov ax, es:[3]
mov di, offset MCB_TABLE_SIZE
add di, 9
call WRD_TO_HEX

mov dx, offset MCB_TABLE_SIZE
call WRITEWRD

mov bx, 8
mov dx, offset MCB_SC_SD
call WRITEWRD

push cx
mov cx, 7

print_scsd_loop:
    mov dl, es:[bx]
    call WRITEBYTE

    inc bx
    loop print_scsd_loop
```

```
    call ENDLINE

    pop cx

    mov ah, es:[0]
    cmp ah, 5ah
    je end_task_1_c

    mov bx, es:[3]
    inc bx

    mov ax, es
    add ax, bx
    mov es, ax

    inc cl

    jmp mcb_for_each_loop

end_task_1_c:
    pop di
    pop si
    pop cx
    pop dx
    pop bx
    pop ax
    ret

TASK_C ENDP

begin:

    call TASK_A
    call TASK_B
    call TASK_C

    ; Выход в DOS
    xor al, al
    mov ah, 4ch
    int 21h

TESTPC  ENDS
```

```
        END start
```

Название файла: lab_2.asm

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

start: jmp begin

AVAILABLE_MEM db 'Available Memory (bytes):$'
EXTENDED_MEM db 'Extended Memory (kbytes):$'
MCB_TABLE db 'MCB Table:', 0DH, 0AH, '$'
MCB_TABLE_NUMBER db 'MCB #  $'

MCB_ADDRESS db 'Address:      $'

PSP_TYPE db ' PSP TYPE: $'
PSP_FREE_AREA db 'free area            $'
PSP_OS_XMS_UMB db 'belongs to OS XMS UMB$'
PSP_RESERVED_FOR_DRIVERS db'reserved for drivers $'
PSP_BELONGS_MSDOS db 'belongs MSDOS        $'
PSP_BUSY_386MAX_UMB db 'busy with block 386MAX UMB$'
PSP_BLOCKED_386MAX db 'blocked 386MAX        $'
PSP_BELONGS_386MAX db 'belongs 386MAX        $'

DEFAULT_PSP_TYPE db '                     $'

MCB_TABLE_SIZE db 'Size:      $'

MCB_SC_SD db ' SC/SD: $'

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

WRITEWRD  PROC  NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEWRD  ENDP

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

ENDLINE PROC NEAR
    push ax
    push dx

    mov dl, 0dh
    call WRITEBYTE

    mov dl, 0ah
```

```
    call WRITEBYTE

    pop dx
    pop ax
    ret
ENDLINE ENDP

WRITE_AVAILABLE_MEM 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

division_loop_av:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_av

print_symbol_loop_av:
    pop dx

    add dl, 30h

    call WRITEBYTE

    loop print_symbol_loop_av

    pop di
    pop dx
    pop cx
```

```
    pop bx
    pop ax
    ret

WRITE_AVAILABLE_MEM ENDP

WRITE_EXTENDED_MEM PROC NEAR
    push ax
    push bx
    push cx
    push dx
    push di

    xor cx, cx
    xor dx, dx

    mov bx, 0ah

division_loop_ext:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_ext

print_symbol_loop_ext:
    pop dx

    add dl, 30h

    call WRITEBYTE

    loop print_symbol_loop_ext

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    ret
```

```
WRITE_EXTENDED_MEM ENDP

TASK_A PROC NEAR

    push ax
    push bx
    push dx

    mov dx, offset AVAILABLE_MEM
    call WRITEWRD

    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx

    call WRITE_AVAILABLE_MEM
    call ENDLINE


    pop dx
    pop bx
    pop ax
    ret

TASK_A ENDP

TASK_B 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_MEM
    call WRITEWRD

    mov ax, bx
    call WRITE_EXTENDED_MEM
    call ENDLINE

    pop dx
    pop bx
    pop ax
    ret

TASK_B ENDP

MCB_PSP_TYPE proc near
    push ax
    push dx
    push di

    ;mov dx, offset TAB
    ;call PRINT_BUF

    mov dx, offset PSP_TYPE
    call WRITEWRD

    cmp ax, 0000h
    je print_free_area

    cmp ax, 0006h
    je print_belongs_OS_XMS_UMB

    cmp ax, 0007h
    je print_reserved_for_drivers

    cmp ax, 0008h
    je print_belongs_MS_DOS

    cmp ax, 0FFFAh
    je print_busy_386MAX_UMB
```

```
    cmp ax, 0FFFDh
    je print_blocked_386MAX

    cmp ax, 0FFFEh
    je print_belongs_386MAX_UMB

    jmp print_psp_type

print_free_area:
    mov dx, offset PSP_FREE_AREA
    call WRITEWRD
    jmp end_psp_type

print_belongs_OS_XMS_UMB:
    mov dx, offset PSP_OS_XMS_UMB
    call WRITEWRD
    jmp end_psp_type

print_reserved_for_drivers:
    mov dx, offset PSP_RESERVED_FOR_DRIVERS
    call WRITEWRD
    jmp end_psp_type

print_belongs_MS_DOS:
    mov dx, offset PSP_BELONGS_MSDOS
    call WRITEWRD
    jmp end_psp_type

print_busy_386MAX_UMB:
    mov dx, offset PSP_BUSY_386MAX_UMB
    call WRITEWRD
    jmp end_psp_type

print_blocked_386MAX:
    mov dx, offset PSP_BLOCKED_386MAX
    call WRITEWRD
    jmp end_psp_type

print_belongs_386MAX_UMB:
    mov dx, offset PSP_BELONGS_386MAX
    call WRITEWRD
    jmp end_psp_type
```

```
print_psp_type:
    mov di, offset DEFAULT_PSP_TYPE
    add di, 3
    call WRD_TO_HEX
    mov dx, offset DEFAULT_PSP_TYPE
    call WRITEWRD

end_psp_type:
    pop di
    pop dx
    pop ax
    ret
MCB_PSP_TYPE endp

TASK_C PROC NEAR


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

    call ENDLINE
    mov dx, offset MCB_TABLE
    call WRITEWRD


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

    xor cx, cx
    mov cl, 01h

mcb_for_each_loop:
    mov al, cl
    mov si, offset MCB_TABLE_NUMBER
    add si, 5
```

```
call BYTE_TO_DEC

mov dx, offset MCB_TABLE_NUMBER
call WRITEWRD

mov ax, es
mov di, offset MCB_ADDRESS
add di, 12
call WRD_TO_HEX

mov dx, offset MCB_ADDRESS
call WRITEWRD

mov ax, es:[1]

call MCB_PSP_TYPE

mov ax, es:[3]
mov di, offset MCB_TABLE_SIZE
add di, 9
call WRD_TO_HEX

mov dx, offset MCB_TABLE_SIZE
call WRITEWRD

mov bx, 8
mov dx, offset MCB_SC_SD
call WRITEWRD

push cx
mov cx, 7

print_scsd_loop:
    mov dl, es:[bx]
    call WRITEBYTE

    inc bx
    loop print_scsd_loop

call ENDLINE
```

```
    pop cx

    mov ah, es:[0]
    cmp ah, 5ah
    je end_task_1_c

    mov bx, es:[3]
    inc bx

    mov ax, es
    add ax, bx
    mov es, ax

    inc cl

    jmp mcb_for_each_loop

end_task_1_c:
    pop di
    pop si
    pop cx
    pop dx
    pop bx
    pop ax
    ret

TASK_C ENDP

FREE_UNUSED_MEMORY PROC NEAR
    push ax
    push bx
    push dx

    xor dx, dx

    lea ax, lafin
    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_UNUSED_MEMORY ENDP

begin:

    mov ah, 4ah
    mov bx, 0ffffh
    int 21h

    call TASK_A
    call TASK_B

    call FREE_UNUSED_MEMORY
    call TASK_C

    ; Выход в DOS
    xor al, al
    mov ah, 4ch
    int 21h
lafin:
TESTPC  ENDS
        END start
```

Название файла: lab_3.asm

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

start: jmp begin

AVAILABLE_MEM db 'Available Memory (bytes):$'
EXTENDED_MEM db 'Extended Memory (kbytes):$'
```

```
MCB_TABLE db 'MCB Table:', 0DH, 0AH, '$'
MCB_TABLE_NUMBER db 'MCB #  $'

MCB_ADDRESS db 'Address:      $'

PSP_TYPE db ' PSP TYPE: $'
PSP_FREE_AREA db 'free area            $'
PSP_OS_XMS_UMB db 'belongs to OS XMS UMB$'
PSP_RESERVED_FOR_DRIVERS db'reserved for drivers $'
PSP_BELONGS_MSDOS db 'belongs MSDOS        $'
PSP_BUSY_386MAX_UMB db 'busy with block 386MAX UMB$'
PSP_BLOCKED_386MAX db 'blocked 386MAX        $'
PSP_BELONGS_386MAX db 'belongs 386MAX        $'

DEFAULT_PSP_TYPE db '                     $'

MCB_TABLE_SIZE db 'Size:      $'

MCB_SC_SD db ' SC/SD: $'

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

WRITEWRD  PROC  NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEWRD  ENDP

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

ENDLINE PROC NEAR
    push ax
    push dx

    mov dl, 0dh
    call WRITEBYTE

    mov dl, 0ah
    call WRITEBYTE

    pop dx
    pop ax
    ret
ENDLINE ENDP

WRITE_AVAILABLE_MEM 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

division_loop_av:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_av

print_symbol_loop_av:
    pop dx

    add dl, 30h

    call WRITEBYTE

    loop print_symbol_loop_av

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    ret

WRITE_AVAILABLE_MEM ENDP

WRITE_EXTENDED_MEM PROC NEAR
    push ax
    push bx
    push cx
    push dx
    push di
```

```
    xor cx, cx
    xor dx, dx

    mov bx, 0ah

division_loop_ext:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_ext

print_symbol_loop_ext:
    pop dx

    add dl, 30h

    call WRITEBYTE

    loop print_symbol_loop_ext

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    ret

WRITE_EXTENDED_MEM ENDP

TASK_A PROC NEAR

    push ax
    push bx
    push dx

    mov dx, offset AVAILABLE_MEM
    call WRITEWRD

    mov ah, 4ah
    mov bx, 0ffffh
```

```
    int 21h
    mov ax, bx

    call WRITE_AVAILABLE_MEM
    call ENDLINE


    pop dx
    pop bx
    pop ax
    ret

TASK_A ENDP

TASK_B 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_MEM
    call WRITEWRD

    mov ax, bx
    call WRITE_EXTENDED_MEM
    call ENDLINE

    pop dx
    pop bx
    pop ax
    ret
```

```
TASK_B ENDP

MCB_PSP_TYPE proc near
    push ax
    push dx
    push di

    ;mov dx, offset TAB
    ;call PRINT_BUF

    mov dx, offset PSP_TYPE
    call WRITEWRD

    cmp ax, 0000h
    je print_free_area

    cmp ax, 0006h
    je print_belongs_OS_XMS_UMB

    cmp ax, 0007h
    je print_reserved_for_drivers

    cmp ax, 0008h
    je print_belongs_MS_DOS

    cmp ax, 0FFFAh
    je print_busy_386MAX_UMB

    cmp ax, 0FFFDh
    je print_blocked_386MAX

    cmp ax, 0FFFEh
    je print_belongs_386MAX_UMB

    jmp print_psp_type

print_free_area:
    mov dx, offset PSP_FREE_AREA
    call WRITEWRD
    jmp end_psp_type
```

```
print_belongs_OS_XMS_UMB:
    mov dx, offset PSP_OS_XMS_UMB
    call WRITEWRD
    jmp end_psp_type

print_reserved_for_drivers:
    mov dx, offset PSP_RESERVED_FOR_DRIVERS
    call WRITEWRD
    jmp end_psp_type

print_belongs_MS_DOS:
    mov dx, offset PSP_BELONGS_MSDOS
    call WRITEWRD
    jmp end_psp_type

print_busy_386MAX_UMB:
    mov dx, offset PSP_BUSY_386MAX_UMB
    call WRITEWRD
    jmp end_psp_type

print_blocked_386MAX:
    mov dx, offset PSP_BLOCKED_386MAX
    call WRITEWRD
    jmp end_psp_type

print_belongs_386MAX_UMB:
    mov dx, offset PSP_BELONGS_386MAX
    call WRITEWRD
    jmp end_psp_type

print_psp_type:
    mov di, offset DEFAULT_PSP_TYPE
    add di, 3
    call WRD_TO_HEX
    mov dx, offset DEFAULT_PSP_TYPE
    call WRITEWRD

end_psp_type:
    pop di
    pop dx
    pop ax
    ret
```

```
MCB_PSP_TYPE endp

TASK_C PROC NEAR


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

    call ENDLINE
    mov dx, offset MCB_TABLE
    call WRITEWRD


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

    xor cx, cx
    mov cl, 01h

mcb_for_each_loop:
    mov al, cl
    mov si, offset MCB_TABLE_NUMBER
    add si, 5

    call BYTE_TO_DEC

    mov dx, offset MCB_TABLE_NUMBER
    call WRITEWRD

    mov ax, es
    mov di, offset MCB_ADDRESS
    add di, 12
    call WRD_TO_HEX

    mov dx, offset MCB_ADDRESS
    call WRITEWRD
```

```
mov ax, es:[1]

call MCB_PSP_TYPE

mov ax, es:[3]
mov di, offset MCB_TABLE_SIZE
add di, 9
call WRD_TO_HEX

mov dx, offset MCB_TABLE_SIZE
call WRITEWRD

mov bx, 8
mov dx, offset MCB_SC_SD
call WRITEWRD

push cx
mov cx, 7

print_scsd_loop:
    mov dl, es:[bx]
    call WRITEBYTE

    inc bx
    loop print_scsd_loop

call ENDLINE

pop cx

mov ah, es:[0]
cmp ah, 5ah
je end_task_1_c

mov bx, es:[3]
inc bx

mov ax, es
add ax, bx
mov es, ax
```

```
    inc cl

    jmp mcb_for_each_loop

end_task_1_c:
    pop di
    pop si
    pop cx
    pop dx
    pop bx
    pop ax
    ret

TASK_C ENDP

ALLOC_MEMORY PROC NEAR

    push ax
    push bx
    push dx

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

    pop dx
    pop bx
    pop ax

ALLOC_MEMORY ENDP

FREE_UNUSED_MEMORY PROC NEAR
    push ax
    push bx
    push dx

    xor dx, dx

    lea ax, lafin
    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_UNUSED_MEMORY ENDP

begin:

    mov ah, 4ah
    mov bx, 0ffffh
    int 21h

    call TASK_A
    call TASK_B

    call FREE_UNUSED_MEMORY
    call ALLOC_MEMORY

    call TASK_C

    ; Выход в DOS
    xor al, al
    mov ah, 4ch
    int 21h
lafin:
TESTPC  ENDS
        END start
```

Название файла: lab_4.asm

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

```
start: jmp begin

AVAILABLE_MEM db 'Available Memory (bytes):$'
EXTENDED_MEM db 'Extended Memory (kbytes):$'
MCB_TABLE db 'MCB Table:', 0DH, 0AH, '$'
MCB_TABLE_NUMBER db 'MCB #  $'

MCB_ADDRESS db 'Address:      $'

PSP_TYPE db ' PSP TYPE: $'
PSP_FREE_AREA db 'free area            $'
PSP_OS_XMS_UMB db 'belongs to OS XMS UMB$'
PSP_RESERVED_FOR_DRIVERS db'reserved for drivers $'
PSP_BELONGS_MSDOS db 'belongs MSDOS        $'
PSP_BUSY_386MAX_UMB db 'busy with block 386MAX UMB$'
PSP_BLOCKED_386MAX db 'blocked 386MAX        $'
PSP_BELONGS_386MAX db 'belongs 386MAX        $'

DEFAULT_PSP_TYPE db '                     $'

MCB_TABLE_SIZE db 'Size:      $'

MCB_SC_SD db ' SC/SD: $'

ALLOCATION_FAILED db '! Allocation failed !$'
ALLOCATION_SUCCESS db '! Allocation success !$'

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

WRITEWRD  PROC  NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEWRD  ENDP

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

ENDLINE PROC NEAR
    push ax
    push dx

    mov dl, 0dh
    call WRITEBYTE

    mov dl, 0ah
    call WRITEBYTE

    pop dx
    pop ax
    ret
```

```
ENDLINE ENDP

WRITE_AVAILABLE_MEM 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

division_loop_av:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_av

print_symbol_loop_av:
    pop dx

    add dl, 30h

    call WRITEBYTE

    loop print_symbol_loop_av

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    ret

WRITE_AVAILABLE_MEM ENDP
```

```
WRITE_EXTENDED_MEM PROC NEAR
    push ax
    push bx
    push cx
    push dx
    push di

    xor cx, cx
    xor dx, dx

    mov bx, 0ah

division_loop_ext:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_ext

print_symbol_loop_ext:
    pop dx

    add dl, 30h

    call WRITEBYTE

    loop print_symbol_loop_ext

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    ret

WRITE_EXTENDED_MEM ENDP

TASK_A PROC NEAR

    push ax
```

```
    push bx
    push dx

    mov dx, offset AVAILABLE_MEM
    call WRITEWRD

    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx

    call WRITE_AVAILABLE_MEM
    call ENDLINE


    pop dx
    pop bx
    pop ax
    ret

TASK_A ENDP

TASK_B 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_MEM
    call WRITEWRD
```

```
    mov ax, bx
    call WRITE_EXTENDED_MEM
    call ENDLINE

    pop dx
    pop bx
    pop ax
    ret

TASK_B ENDP

MCB_PSP_TYPE proc near
    push ax
    push dx
    push di

    ;mov dx, offset TAB
    ;call PRINT_BUF

    mov dx, offset PSP_TYPE
    call WRITEWRD

    cmp ax, 0000h
    je print_free_area

    cmp ax, 0006h
    je print_belongs_OS_XMS_UMB

    cmp ax, 0007h
    je print_reserved_for_drivers

    cmp ax, 0008h
    je print_belongs_MS_DOS

    cmp ax, 0FFFAh
    je print_busy_386MAX_UMB

    cmp ax, 0FFFDh
    je print_blocked_386MAX

    cmp ax, 0FFFEh
    je print_belongs_386MAX_UMB
```

```
    jmp print_psp_type

print_free_area:
    mov dx, offset PSP_FREE_AREA
    call WRITEWRD
    jmp end_psp_type

print_belongs_OS_XMS_UMB:
    mov dx, offset PSP_OS_XMS_UMB
    call WRITEWRD
    jmp end_psp_type

print_reserved_for_drivers:
    mov dx, offset PSP_RESERVED_FOR_DRIVERS
    call WRITEWRD
    jmp end_psp_type

print_belongs_MS_DOS:
    mov dx, offset PSP_BELONGS_MSDOS
    call WRITEWRD
    jmp end_psp_type

print_busy_386MAX_UMB:
    mov dx, offset PSP_BUSY_386MAX_UMB
    call WRITEWRD
    jmp end_psp_type

print_blocked_386MAX:
    mov dx, offset PSP_BLOCKED_386MAX
    call WRITEWRD
    jmp end_psp_type

print_belongs_386MAX_UMB:
    mov dx, offset PSP_BELONGS_386MAX
    call WRITEWRD
    jmp end_psp_type

print_psp_type:
    mov di, offset DEFAULT_PSP_TYPE
    add di, 3
    call WRD_TO_HEX
```

```
    mov dx, offset DEFAULT_PSP_TYPE
    call WRITEWRD

end_psp_type:
    pop di
    pop dx
    pop ax
    ret
MCB_PSP_TYPE endp

TASK_C PROC NEAR


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

    call ENDLINE
    mov dx, offset MCB_TABLE
    call WRITEWRD


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

    xor cx, cx
    mov cl, 01h

mcb_for_each_loop:
    mov al, cl
    mov si, offset MCB_TABLE_NUMBER
    add si, 5

    call BYTE_TO_DEC

    mov dx, offset MCB_TABLE_NUMBER
    call WRITEWRD
```

```
mov ax, es
mov di, offset MCB_ADDRESS
add di, 12
call WRD_TO_HEX

mov dx, offset MCB_ADDRESS
call WRITEWRD

mov ax, es:[1]

call MCB_PSP_TYPE

mov ax, es:[3]
mov di, offset MCB_TABLE_SIZE
add di, 9
call WRD_TO_HEX

mov dx, offset MCB_TABLE_SIZE
call WRITEWRD

mov bx, 8
mov dx, offset MCB_SC_SD
call WRITEWRD

push cx
mov cx, 7

print_scsd_loop:
    mov dl, es:[bx]
    call WRITEBYTE

    inc bx
    loop print_scsd_loop

call ENDLINE

pop cx

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

```
    mov bx, es:[3]
    inc bx

    mov ax, es
    add ax, bx
    mov es, ax

    inc cl

    jmp mcb_for_each_loop

end_task_c:
    pop di
    pop si
    pop cx
    pop dx
    pop bx
    pop ax
    ret

TASK_C ENDP

ALLOC_MEMORY PROC NEAR

    push ax
    push bx
    push dx

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

    call ENDLINE

    jc error
    jne allocated_succesfully

error:
    mov dx, offset ALLOCATION_FAILED
    call WRITEWRD
    jmp allocation_end
```

```
allocated_succesfully:
    mov dx, offset ALLOCATION_SUCCESS
    call WRITEWRD

allocation_end:
    call ENDLINE
    pop dx
    pop bx
    pop ax
    ret

ALLOC_MEMORY ENDP

FREE_UNUSED_MEMORY PROC NEAR
    push ax
    push bx
    push dx

    xor dx, dx

    lea ax, lafin
    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_UNUSED_MEMORY ENDP

begin:

    mov ah, 4ah
```

```
    mov bx, 0ffffh
    int 21h

    call TASK_A
    call TASK_B

    call ALLOC_MEMORY
    call FREE_UNUSED_MEMORY

    call TASK_C

    ; Выход в DOS
    xor al, al
    mov ah, 4ch
    int 21h
lafin:
TESTPC  ENDS
        END start
```