

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

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

Студентка гр. 0382

Морева Е.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Для исследования организации управления памятью необходимо ориентироваться на тип основной памяти, реализованный в компьютере и способ организации, принятый в ОС. В лабораторной работе рассматривается нестраничная 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 типа .COM, который выбирает и распечатывает следующую информацию:

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

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

Запустить программу и внимательно оценить результаты.

Шаг 2. Изменить программу таким образом, чтобы она освобождала память, которую она не занимает. Для этого нужно использовать функцию 4Ah прерывания 21h. Запустить модифицированную программу. Сравнить выходные данные с результатами, полученными на предыдущем шаге.

Шаг 3. Изменить программу еще раз таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти функцией 48Н прерывания 21Н. Повторить эксперимент, запустив модифицированную программу. Сравнить выходные данные с результатами, полученными на предыдущих шагах.

Шаг 4. Изменить первоначальный вариант программы (с шага 2), запросив 64Кб памяти функцией 48Н прерывания 21Н до освобождения памяти. Обязательно обрабатывать завершение функций ядра, проверяя флаг CF.

Шаг 5. Оценить результаты, полученные на предыдущих шагах. Ответить на контрольные вопросы .

Выполнение работы.

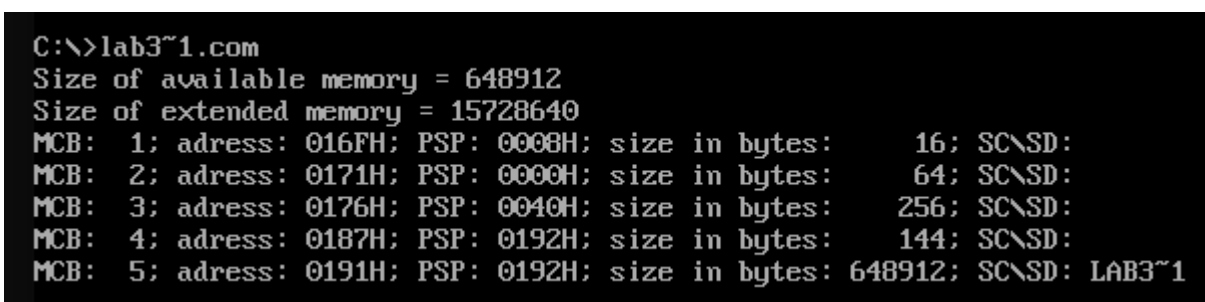
Функции, используемые в программе:

1. *PRINT* — печатает строку по адресу из DX.
2. *HEX_TO_DEC* — записывает переводит число в регистре AX из шестнадцатеричной системы в десятичную число в регистре AX в память в десятичном представлении, по адресу SI.
3. *PAR_TO_DEC* — с помощью функции *HEX_TO_DEC* , переводит число в AX из размера в параграфах в размер в байтах и записывает начиная с адреса из SI.
4. *kByteToByte* — с помощью *HEX_TO_DEC* переводит число а AX из килобайтов в байты и записывает по адресу, начиная с SI.

5. *printMemorySize* — находит размер доступной памяти в байтах(с помощью функции *PAR_TO_DEC*) и выводит со строкой *memorySize*.
6. *printExtMemorySize* — находит размер расширенной памяти в байтах(с помощью функции *kByteToByte*) и выводит со строкой *extMemorySize*.
7. *nextMCB* — выводит очередной список MCB из списка списков в строку адрес которой лежит в SI. Использует строку *MCB*.
8. *printMCB* — выводит весь список списков.
9. *freeingMemory* — высвобождает память которую не занимает программа.
10. *queryMemory* — запрашивает память в размере 64 КБ. При ошибке выводится сообщение *perror*.

Шаг 1.

Написана программа которая с помощью функций *printMemorySize*, *printExtMemorySize*, *nextMCB*, *printMCB* выводит информацию о количестве доступной памяти, размере расширенной памяти, а так же цепочку блоков управления памятью. (Результат работы программы представленна рис. 1.)

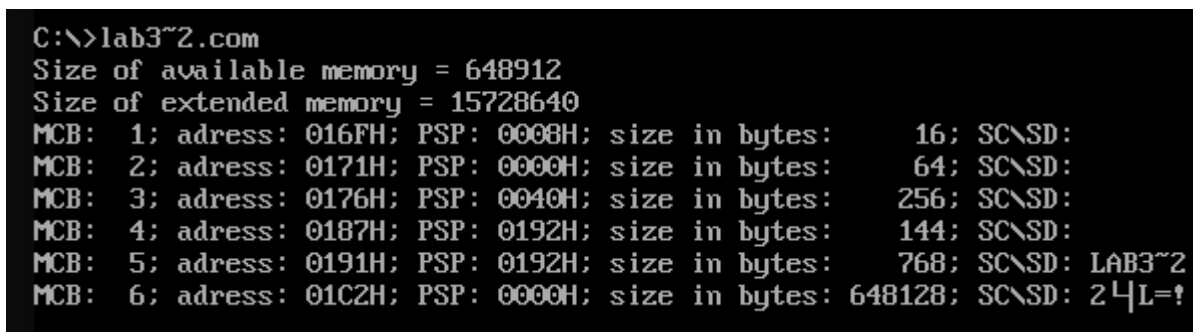


```
C:\>lab3~1.com
Size of available memory = 648912
Size of extended memory = 15728640
MCB: 1; adress: 016FH; PSP: 0008H; size in bytes: 16; SC\SD:
MCB: 2; adress: 0171H; PSP: 0000H; size in bytes: 64; SC\SD:
MCB: 3; adress: 0176H; PSP: 0040H; size in bytes: 256; SC\SD:
MCB: 4; adress: 0187H; PSP: 0192H; size in bytes: 144; SC\SD:
MCB: 5; adress: 0191H; PSP: 0192H; size in bytes: 648912; SC\SD: LAB3~1
```

Рисунок 1: Результат запуска первого варианта программы(модуль lab3~1.com)

Шаг 2.

Написана функция *freeingMemory*, которая освобождает память незанимаемую программой. (Результат работы программы представлен на рис. 2)



```
C:\>lab3~2.com
Size of available memory = 648912
Size of extended memory = 15728640
MCB: 1; adress: 016FH; PSP: 0008H; size in bytes: 16; SC\SD:
MCB: 2; adress: 0171H; PSP: 0000H; size in bytes: 64; SC\SD:
MCB: 3; adress: 0176H; PSP: 0040H; size in bytes: 256; SC\SD:
MCB: 4; adress: 0187H; PSP: 0192H; size in bytes: 144; SC\SD:
MCB: 5; adress: 0191H; PSP: 0192H; size in bytes: 768; SC\SD: LAB3~2
MCB: 6; adress: 01C2H; PSP: 0000H; size in bytes: 648128; SC\SD: 24L=!
```

Рисунок 2: Результат запуска второго варианта программы
(модуль lab3~2.com)

При сравнении рисунков видно, что изначально (на первом шаге) 5 блок MCB — память, выделенная программе, был размером почти со всю доступную память, то есть столько занимала программа. На втором шаге после освобождения лишней памяти, свободная память выделилась в отдельный шестой блок свободной памяти, а пятый блок стал размером, который точно требуется для размещения программы.

Шаг 3.

Написана функция *queryMemory*, которая запрашиваем память размером 64 КБ. (Результат работы программы представлен на рис. 3.)

```
C:\>lab3~3.com
Size of available memory = 648912
Size of extended memory = 15728640
MCB: 1; address: 016FH; PSP: 0008H; size in bytes: 16; SC\SD:
MCB: 2; address: 0171H; PSP: 0000H; size in bytes: 64; SC\SD:
MCB: 3; address: 0176H; PSP: 0040H; size in bytes: 256; SC\SD:
MCB: 4; address: 0187H; PSP: 0192H; size in bytes: 144; SC\SD:
MCB: 5; address: 0191H; PSP: 0192H; size in bytes: 832; SC\SD: LAB3~3
MCB: 6; address: 01C6H; PSP: 0192H; size in bytes: 65536; SC\SD: LAB3~3
MCB: 7; address: 11C7H; PSP: 0000H; size in bytes: 582512; SC\SD:
```

Рисунок 3: Результат запуска третьего варианта программы
(модуль lab3~3.com)

По итогу работы программы видно, что теперь программе принадлежит два блока памяти: первый — под номером 5, получился после освобождения неиспользуемой памяти, второй — под номером 6, выделенный в ходе запроса памяти размером 64 КБ (65536 байта).

Шаг 4.

Пробуем выделить 64Кб памяти до ее высвобождения. (Результат работы программы представлен на рис. 4.)

```
C:\>lab3~4.com
Size of available memory = 648912
Size of extended memory = 15728640
queryMemory ERROR
MCB: 1; adress: 016FH; PSP: 0008H; size in bytes: 16; SC\SD:
MCB: 2; adress: 0171H; PSP: 0000H; size in bytes: 64; SC\SD:
MCB: 3; adress: 0176H; PSP: 0040H; size in bytes: 256; SC\SD:
MCB: 4; adress: 0187H; PSP: 0192H; size in bytes: 144; SC\SD:
MCB: 5; adress: 0191H; PSP: 0192H; size in bytes: 832; SC\SD: LAB3~4
MCB: 6; adress: 01C6H; PSP: 0000H; size in bytes: 648064; SC\SD: LAB3~3
```

Рисунок 4: Результат запуска четвертого варианта программы
(модуль lab3_4.com)

По рисунку видно, что выделение памяти провалилось, т. к. вся свободная память уже принадлежала программе, следовательно выделить еще памяти не получилось. А освобождение памяти произошло успешно, это видно по 6 строке в таблице, там показана свободная память.

Исходный код программ см. в приложении А

Ответы на контрольные вопросы.

1. Что означает "доступный объем памяти"?

Доступный бъем памяти это объем оперативной памяти, который может быть выделен для модуля программы. Этот объем необязательно использовать целиком, его можно высвободить если он не используется программой.

2. Где МСВ блок Вашей программы в списке?

Его легко найти в списке по названию программы, написанном в графе SC\SD. Номер модуля соответственно указан в первой графе. В первом, втором и четвертом шаге это — строка номер 5, на третьем — 5 и 6.

3. Какой размер памяти занимает программа в каждом случае?

На первом шаге: 648912 байт (вся доступная память)

На втором шаге: 768 байт (объем памяти, которого ровно хватит для программы, необходимая память)

На третьем шаге: $832 + 65536 = 66368$ байт (необходимая память + память, запрошенная программой в размере 64 Кбайт).

На четвертом шаге: 832 байта (необходимая память, но без дополнительно запрошенной памяти, т. к. выделить ее не удалось).

Выводы.

В ходе работы была изучена организация управления памятью. Было исследовано устройство нестраничной памяти и способ управления динамическими разделами. Исследованы структуры данных и работа функций управления памятью ядра операционной системы.

ПРИЛОЖЕНИЕ А

КОД МОДУЛЕЙ

Название файла: lab3~1.asm

AStack SEGME

MainSeg SEGMENT

ASSUME CS:MainSeg, DS:MainSeg, ES:NOTHING, SS:NOTHING

ORG 100H

START:

jmp BEGIN

DATA:

memorySize db "Size of available memory = ", 0DH, 0AH, "\$"

extMemorySize db "Size of extended memory = ", 0DH, 0AH, "\$"

MCB db "MCB: ; adress: H; PSP: H; size in bytes: ; SC\SD: ", 0DH, 0AH, "\$"

BYTE_TO_DEC PROC NEAR

; AL - number, SI - adress of last symbol

push CX

push DX

push AX

xor AH,AH

xor DX,DX

mov CX,10

loop_bd:

div CX

or DL,30H

mov [SI],DL

```
dec SI
xor DX,DX
cmp AX,10
jae loop_bd
```

```
cmp AL,00H
je end_l
```

```
or AL,30H
mov [SI],AL
```

```
end_l:
pop AX
pop DX
pop CX
```

```
ret
```

```
BYTE_TO_DEC ENDP
```

```
TETR_TO_HEX PROC NEAR
```

```
and AL,0FH ; save only last part of byte
```

```
cmp AL,09
```

```
jbe next
```

```
add AL,07
```

```
next:
```

```
add AL,30H
```

```
ret
```

```
TETR_TO_HEX ENDP
```

```
BYTE_TO_HEX PROC NEAR
```

```
; AL - number -> 2 symbols in 16 numb. syst. in AX
```

```

push CX

mov AH,AL ; save AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX ; AL - high numb ascii, AH - low numb ascii

pop CX
ret
BYTE_TO_HEX ENDP

```

```

WRD_TO_HEX PROC NEAR
; AX - number, DI - last symbol adress

```

```

push BX
push AX

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 AX

```

```
    pop BX
    ret
WRD_TO_HEX ENDP
```

```
PRINT PROC near
```

```
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
PRINT ENDP
```

```
HEX_TO_DEC PROC NEAR
```

```
    ; AX - size in paragraphs, SI - adress of last symbol in result string
```

```
    mov BX,0AH
```

```
loop_wr:
```

```
    div BX
```

```
    add DX,30H
```

```
    mov [SI],DL
```

```
    xor DX,DX
```

```
    dec SI
```

```
    cmp AX,0000H
```

```
    jne loop_wr
```

```
    ret
```

```
HEX_TO_DEC ENDP
```

```
PAR_TO_DEC PROC NEAR
```

```
    ; AX - size in paragraphs, SI - adress of last symbol in result string
```

```

push AX
push BX
push DX
push SI

mov BX,10H
mul BX    ; AX*16

call HEX_TO_DEC

pop SI
pop DX
pop BX
pop AX
ret
PAR_TO_DEC ENDP

kByteToByte PROC NEAR
push AX
push BX
push DX
push SI

mov BX,10000 ; separated 4 chars from DX AX, AX = (DX AX) div BX,
div BX ; DX = (DX AX) mod BX
push AX
mov AX,DX ; explore DX AX - last 6 (in 10 numb syst) chars from DX AX
xor DX,DX

call HEX_TO_DEC

pop AX ; explore DX AX - first 5 (in 10 numb syst) chars

```

```

call HEX_TO_DEC

pop SI
pop DX
pop BX
pop AX
ret
kByteToByte ENDP

printMemorySize PROC NEAR
    mov AH,4AH
    mov BX,0FFFFH
    int 21H

    mov AX,BX
    mov SI,offset memorySize + 32
    call PAR_TO_DEC
    mov DX,offset memorySize
    call PRINT
    ret
printMemorySize ENDP

printExtMemorySize PROC NEAR
    mov AL,30H
    out 70H,AL
    in AL,71H
    mov BL,AL
    mov AL,31H
    out 70H,AL
    in AL,71H
    mov AH,AL

```

```

mov AL,BL

mov SI,offset extMemorySize + 33
mov BX,400H ; multiply 1024
mul BX

call kByteToByte

mov DX,offset extMemorySize
call PRINT
ret
printExtMemorySize ENDP

nextMCB PROC NEAR

push AX
push ES
push CX
push DX
push BX

mov AX,CX
mov SI,offset mcb + 6
call BYTE_TO_DEC

mov AX,ES
mov DI,offset mcb + 20
call WRD_TO_HEX

mov AX,ES:[01H]
mov DI,offset mcb + 32
call WRD_TO_HEX

```



```
mov AX,ES:[03H]
mov SI,offset mcb + 56
call PAR_TO_DEC
```

```
mov BX,08H
mov CX,7
mov SI,offset mcb + 66
one_mcb_lp:
    mov DX,ES:[BX]
    mov [SI],DX
    inc BX
    inc SI
    loop one_mcb_lp
```

```
mov DX,offset mcb
call PRINT
```

```
pop BX
pop DX
pop CX
pop ES
pop AX
ret
```

```
nextMCB ENDP
```

```
printMCB PROC NEAR
```

```
    mov AH,52H
    int 21H
```

```
    mov AX,ES:[BX-2]
    mov ES,AX
```

```

xor CX,CX
mov CX,1H

mcb_lp:
    call nextMCB

    mov AL,ES:[00H]
    cmp AL,5AH
    je end_mcb

    mov BX,ES:[03H]
    mov AX,ES
    add AX,BX
    inc AX
    mov ES,AX
    inc CX
    jmp mcb_lp

end_mcb:
    ret
printMCB ENDP

BEGIN:
    call printMemorySize
    call printExtMemorySize
    call printMCB
    xor AL,AL
        mov AH,4Ch
        int 21H

MainSeg ENDS

```

END START

Название файла: lab3~2.asm

 MainSeg SEGMENT

 ASSUME CS:MainSeg, DS:MainSeg, ES:NOTHING, SS:NOTHING

 ORG 100H

START:

 jmp BEGIN

DATA:

 memorySize db "Size of available memory = ", 0DH, 0AH, "\$"

 extMemorySize db "Size of extended memory = ", 0DH, 0AH, "\$"

 MCB db "MCB: ; adress: H; PSP: H; size in bytes: ; SC\SD: ", 0DH, 0AH, "\$"

BYTE_TO_DEC PROC NEAR

 ; AL - number, SI - adress of last symbol

 push CX

 push DX

 push AX

 xor AH,AH

 xor DX,DX

 mov CX,10

loop_bd:

 div CX

```
or DL,30H
mov [SI],DL
dec SI
xor DX,DX
cmp AX,10
jae loop_bd
```

```
cmp AL,00H
je end_l
```

```
or AL,30H
mov [SI],AL
```

```
end_l:
pop AX
pop DX
pop CX
```

```
ret
```

```
BYTE_TO_DEC ENDP
```

```
TETR_TO_HEX PROC NEAR
```

```
and AL,0FH ; save only last part of byte
cmp AL,09
jbe next
add AL,07
```

```
next:
add AL,30H
ret
```

```
TETR_TO_HEX ENDP
```

```
BYTE_TO_HEX PROC NEAR
```

; AL - number -> 2 symbols in 16 numb. syst. in AX

push CX

mov AH,AL ; save AL

call TETR_TO_HEX

xchg AL,AH

mov CL,4

shr AL,CL

call TETR_TO_HEX ; AL - high numb ascii, AH - low numb ascii

pop CX

ret

BYTE_TO_HEX ENDP

WRD_TO_HEX PROC NEAR

; AX - number, DI - last symbol adress

push BX

push AX

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 AX
    pop BX
    ret
WRD_TO_HEX ENDP
```

```
PRINT PROC near
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
PRINT ENDP
```

```
HEX_TO_DEC PROC NEAR
    ; AX - size in paragraphs, SI - adress of last symbol in result string
```

```
    mov BX,0AH
```

```
loop_wr:
    div BX
    add DX,30H
    mov [SI],DL
    xor DX,DX
    dec SI
    cmp AX,0000H
    jne loop_wr
```

```
    ret
HEX_TO_DEC ENDP
```

```
PAR_TO_DEC PROC NEAR
```

; AX - size in paragraphs, SI - adress of last symbol in result string

push AX

push BX

push DX

push SI

mov BX,10H

mul BX ; AX*16

call HEX_TO_DEC

pop SI

pop DX

pop BX

pop AX

ret

PAR_TO_DEC ENDP

kByteToByte PROC NEAR

push AX

push BX

push DX

push SI

mov BX,10000 ; separated 4 chars from DX AX, AX = (DX AX) div BX,

div BX ; DX = (DX AX) mod BX

push AX

mov AX,DX ; explore DX AX - last 6 (in 10 numb syst) chars from DX AX

xor DX,DX

call HEX_TO_DEC

pop AX ; explore DX AX - first 5 (in 10 numb syst) chars

call HEX_TO_DEC

pop SI

pop DX

pop BX

pop AX

ret

kByteToByte ENDP

printMemorySize PROC NEAR

mov AH,4AH

mov BX,0FFFFH

int 21H

mov AX,BX

mov SI,offset memorySize + 32

call PAR_TO_DEC

mov DX,offset memorySize

call PRINT

ret

printMemorySize ENDP

printExtMemorySize PROC NEAR

mov AL,30H

out 70H,AL

in AL,71H

mov BL,AL

mov AL,31H

out 70H,AL


```

in AL,71H
mov AH,AL
mov AL,BL

mov SI,offset extMemorySize + 33
mov BX,400H ; multiply 1024
mul BX

call kByteToByte

mov DX,offset extMemorySize
call PRINT
ret
printExtMemorySize ENDP

nextMCB PROC NEAR

push AX
push ES
push CX
push DX
push BX

mov AX,CX
mov SI,offset mcb + 6
call BYTE_TO_DEC

mov AX,ES
mov DI,offset mcb + 20
call WRD_TO_HEX

mov AX,ES:[01H]

```

```
mov DI,offset mcb + 32  
call WRD_TO_HEX
```

```
mov AX,ES:[03H]  
mov SI,offset mcb + 56  
call PAR_TO_DEC
```

```
mov BX,08H  
mov CX,7  
mov SI,offset mcb + 66  
one_mcb_lp:  
    mov DX,ES:[BX]  
    mov [SI],DX  
    inc BX  
    inc SI  
    loop one_mcb_lp
```

```
mov DX,offset mcb  
call PRINT
```

```
pop BX  
pop DX  
pop CX  
pop ES  
pop AX  
ret
```

```
nextMCB ENDP
```

```
printMCB PROC NEAR  
    mov AH,52H  
    int 21H
```

```

mov AX,ES:[BX-2]
mov ES,AX

xor CX,CX
mov CX,1H

mcb_lp:
    call nextMCB

    mov AL,ES:[00H]
    cmp AL,5AH
    je end_mcb

    mov BX,ES:[03H]
    mov AX,ES
    add AX,BX
    inc AX
    mov ES,AX
    inc CX
    jmp mcb_lp

end_mcb:
    ret
printMCB ENDP

freeingMemory PROC NEAR
    push AX
    push BX
    push DX
    lea AX,ENDPROGRAMM
    mov BX,10H ; size of paragraph
    xor DX,DX

```

```
div BX
mov BX,AX
mov AH,4AH
int 21H

    pop AX
    pop BX
    pop DX

ret
freeingMemory ENDP
```

```
BEGIN:
    call printMemorySize
    call printExtMemorySize
        call freeingMemory
    call printMCB
    xor AL,AL
        mov AH,4Ch
        int 21H
```

```
ENDPROGRAMM:
MainSeg ENDS
END START
```

Название файла: lab3~3.asm

```
    MainSeg SEGMENT
ASSUME CS:MainSeg, DS:MainSeg, ES:NOTHING, SS:NOTHING
ORG 100H
```

START:

jmp BEGIN

DATA:

memorySize db "Size of available memory = ", 0DH, 0AH, "\$"

extMemorySize db "Size of extended memory = ", 0DH, 0AH, "\$"

MCB db "MCB: ; adress: H; PSP: H; size in bytes: ; SC\SD: ", 0DH, 0AH, "\$"

perror db "queryMemory ERROR", 0DH, 0AH, "\$"

BYTE_TO_DEC PROC NEAR

; AL - number, SI - adress of last symbol

push CX

push DX

push AX

xor AH,AH

xor DX,DX

mov CX,10

loop_bd:

div CX

or DL,30H

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop_bd

cmp AL,00H

je end_l

```

    or AL,30H
    mov [SI],AL

end_l:
    pop AX
    pop DX
    pop CX

    ret
BYTE_TO_DEC ENDP

TETR_TO_HEX PROC NEAR
    and AL,0FH ; save only last part of byte
    cmp AL,09
    jbe next
    add AL,07
next:
    add AL,30H
    ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC NEAR
    ; AL - number -> 2 symbols in 16 numb. syst. in AX

    push CX

    mov AH,AL ; save AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX ; AL - high numb ascii, AH - low numb ascii

```

```
    pop CX
    ret
BYTE_TO_HEX ENDP
```

```
WRD_TO_HEX PROC NEAR
    ; AX - number, DI - last symbol adress
```

```
    push BX
    push AX
```

```
    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 AX
    pop BX
    ret
WRD_TO_HEX ENDP
```

```
PRINT PROC near
    push AX
    mov AH,09h
    int 21h
```

```
pop AX
ret
PRINT ENDP
```

```
HEX_TO_DEC PROC NEAR
```

```
; AX - size in paragraphs, SI - adress of last symbol in result string
```

```
mov BX,0AH
```

```
loop_wr:
```

```
div BX
```

```
add DX,30H
```

```
mov [SI],DL
```

```
xor DX,DX
```

```
dec SI
```

```
cmp AX,0000H
```

```
jne loop_wr
```

```
ret
```

```
HEX_TO_DEC ENDP
```

```
PAR_TO_DEC PROC NEAR
```

```
; AX - size in paragraphs, SI - adress of last symbol in result string
```

```
push AX
```

```
push BX
```

```
push DX
```

```
push SI
```

```
mov BX,10H
```

```
mul BX ; AX*16
```



```
call HEX_TO_DEC
```

```
pop SI
```

```
pop DX
```

```
pop BX
```

```
pop AX
```

```
ret
```

```
PAR_TO_DEC ENDP
```

```
kByteToByte PROC NEAR
```

```
push AX
```

```
push BX
```

```
push DX
```

```
push SI
```

```
mov BX,10000 ; separated 4 chars from DX AX,  $AX = (DX\ AX) \div BX$ ,
```

```
div BX ;  $DX = (DX\ AX) \bmod BX$ 
```

```
push AX
```

```
mov AX,DX ; explore DX AX - last 6 (in 10 numb syst) chars from DX AX
```

```
xor DX,DX
```

```
call HEX_TO_DEC
```

```
pop AX ; explore DX AX - first 5 (in 10 numb syst) chars
```

```
call HEX_TO_DEC
```

```
pop SI
```

```
pop DX
```

```
pop BX
```

```
pop AX
```

```
ret
```

kByteToByte ENDP

printMemorySize PROC NEAR

mov AH,4AH

mov BX,0FFFFH

int 21H

mov AX,BX

mov SI,offset memorySize + 32

call PAR_TO_DEC

mov DX,offset memorySize

call PRINT

ret

printMemorySize ENDP

printExtMemorySize PROC NEAR

mov AL,30H

out 70H,AL

in AL,71H

mov BL,AL

mov AL,31H

out 70H,AL

in AL,71H

mov AH,AL

mov AL,BL

mov SI,offset extMemorySize + 33

mov BX,400H ; multiply 1024

mul BX

call kByteToByte

```
    mov DX,offset extMemorySize
    call PRINT
    ret
printExtMemorySize ENDP
```

```
nextMCB PROC NEAR
```

```
    push AX
    push ES
    push CX
    push DX
    push BX
```

```
    mov AX,CX
    mov SI,offset mcb + 6
    call BYTE_TO_DEC
```

```
    mov AX,ES
    mov DI,offset mcb + 20
    call WRD_TO_HEX
```

```
    mov AX,ES:[01H]
    mov DI,offset mcb + 32
    call WRD_TO_HEX
```

```
    mov AX,ES:[03H]
    mov SI,offset mcb + 56
    call PAR_TO_DEC
```

```
    mov BX,08H
    mov CX,7
    mov SI,offset mcb + 66
```

```

one_mcb_lp:
    mov DX,ES:[BX]
    mov [SI],DX
    inc BX
    inc SI
    loop one_mcb_lp

    mov DX,offset mcb
    call PRINT

    pop BX
    pop DX
    pop CX
    pop ES
    pop AX
    ret
nextMCB ENDP

printMCB PROC NEAR
    mov AH,52H
    int 21H

    mov AX,ES:[BX-2]
    mov ES,AX

    xor CX,CX
    mov CX,1H

mcb_lp:
    call nextMCB

    mov AL,ES:[00H]

```

```

    cmp AL,5AH
    je end_mcb

    mov BX,ES:[03H]
    mov AX,ES
    add AX,BX
    inc AX
    mov ES,AX
    inc CX
    jmp mcb_lp

end_mcb:
    ret
printMCB ENDP

freeingMemory PROC NEAR
    push AX
    push BX
    push DX

    lea AX,ENDPROGRAMM
    mov BX,10H ; size of paragraph
    xor DX,DX
    div BX
    inc AX
    mov BX,AX
    mov AH,4AH
    int 21H
    pop DX

    pop BX

```

```

        pop AX

        ret
freeingMemory ENDP

queryMemory PROC NEAR
    xor AX,AX
    mov BX,1000H
    mov AH,48H
    int 21H

    jnc final
    mov DX,offset perror
    call PRINT

final:

    ret
queryMemory ENDP

BEGIN:
    call printMemorySize
    call printExtMemorySize
        call freeingMemory
        call queryMemory
    call printMCB

    xor AL,AL
        mov AH,4Ch
        int 21H

ENDPROGRAMM:

```

MainSeg ENDS

END START

Название файла: lab3~4.asm

MainSeg SEGMENT

ASSUME CS:MainSeg, DS:MainSeg, ES:NOTHING, SS:NOTHING

ORG 100H

START:

jmp BEGIN

DATA:

memorySize db "Size of available memory = ", 0DH, 0AH, "\$"

extMemorySize db "Size of extended memory = ", 0DH, 0AH, "\$"

MCB db "MCB: ; adress: H; PSP: H; size in bytes: ; SC\SD: ", 0DH, 0AH, "\$"

perror db "queryMemory ERROR", 0DH, 0AH, "\$"

BYTE_TO_DEC PROC NEAR

; AL - number, SI - adress of last symbol

push CX

push DX

push AX

xor AH,AH

xor DX,DX

mov CX,10

```
loop_bd:
    div CX
    or DL,30H
    mov [SI],DL
    dec SI
    xor DX,DX
    cmp AX,10
    jae loop_bd
```

```
    cmp AL,00H
    je end_l
```

```
    or AL,30H
    mov [SI],AL
```

```
end_l:
    pop AX
    pop DX
    pop CX
```

```
    ret
```

```
BYTE_TO_DEC ENDP
```

```
TETR_TO_HEX PROC NEAR
```

```
    and AL,0FH ; save only last part of byte
```

```
    cmp AL,09
```

```
    jbe next
```

```
    add AL,07
```

```
next:
```

```
    add AL,30H
```

```
    ret
```

```
TETR_TO_HEX ENDP
```


BYTE_TO_HEX PROC NEAR

; AL - number -> 2 symbols in 16 numb. syst. in AX

push CX

mov AH,AL ; save AL

call TETR_TO_HEX

xchg AL,AH

mov CL,4

shr AL,CL

call TETR_TO_HEX ; AL - high numb ascii, AH - low numb ascii

pop CX

ret

BYTE_TO_HEX ENDP

WRD_TO_HEX PROC NEAR

; AX - number, DI - last symbol adress

push BX

push AX

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 AX
    pop BX
    ret
WRD_TO_HEX ENDP
```

```
PRINT PROC near
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
PRINT ENDP
```

```
HEX_TO_DEC PROC NEAR
    ; AX - size in paragraphs, SI - adress of last symbol in result string
```

```
    mov BX,0AH

loop_wr:
    div BX
    add DX,30H
    mov [SI],DL
    xor DX,DX
    dec SI
    cmp AX,0000H
    jne loop_wr

    ret
HEX_TO_DEC ENDP
```

PAR_TO_DEC PROC NEAR

; AX - size in paragraphs, SI - adress of last symbol in result string

push AX

push BX

push DX

push SI

mov BX,10H

mul BX ; AX*16

call HEX_TO_DEC

pop SI

pop DX

pop BX

pop AX

ret

PAR_TO_DEC ENDP

kByteToByte PROC NEAR

push AX

push BX

push DX

push SI

mov BX,10000 ; separated 4 chars from DX AX, AX = (DX AX) div BX,

div BX ; DX = (DX AX) mod BX

push AX

mov AX,DX ; explore DX AX - last 6 (in 10 numb syst) chars from DX AX

xor DX,DX

```
call HEX_TO_DEC
```

```
pop AX ; explore DX AX - first 5 (in 10 numb syst) chars
```

```
call HEX_TO_DEC
```

```
pop SI
```

```
pop DX
```

```
pop BX
```

```
pop AX
```

```
ret
```

```
kByteToByte ENDP
```

```
printMemorySize PROC NEAR
```

```
mov AH,4AH
```

```
mov BX,0FFFFH
```

```
int 21H
```

```
mov AX,BX
```

```
mov SI,offset memorySize + 32
```

```
call PAR_TO_DEC
```

```
mov DX,offset memorySize
```

```
call PRINT
```

```
ret
```

```
printMemorySize ENDP
```

```
printExtMemorySize PROC NEAR
```

```
mov AL,30H
```

```
out 70H,AL
```

```
in AL,71H
```

```
mov BL,AL
```

```

mov AL,31H
out 70H,AL
in AL,71H
mov AH,AL
mov AL,BL

mov SI,offset extMemorySize + 33
mov BX,400H ; multiply 1024
mul BX

call kByteToByte

mov DX,offset extMemorySize
call PRINT
ret
printExtMemorySize ENDP

nextMCB PROC NEAR

push AX
push ES
push CX
push DX
push BX

mov AX,CX
mov SI,offset mcb + 6
call BYTE_TO_DEC

mov AX,ES
mov DI,offset mcb + 20
call WRD_TO_HEX

```

```
mov AX,ES:[01H]
mov DI,offset mcb + 32
call WRD_TO_HEX
```

```
mov AX,ES:[03H]
mov SI,offset mcb + 56
call PAR_TO_DEC
```

```
mov BX,08H
mov CX,7
mov SI,offset mcb + 66
one_mcb_lp:
    mov DX,ES:[BX]
    mov [SI],DX
    inc BX
    inc SI
    loop one_mcb_lp
```

```
mov DX,offset mcb
call PRINT
```

```
pop BX
pop DX
pop CX
pop ES
pop AX
ret
```

```
nextMCB ENDP
```

```
printMCB PROC NEAR
    mov AH,52H
```

int 21H

mov AX,ES:[BX-2]

mov ES,AX

xor CX,CX

mov CX,1H

mcb_lp:

call nextMCB

mov AL,ES:[00H]

cmp AL,5AH

je end_mcb

mov BX,ES:[03H]

mov AX,ES

add AX,BX

inc AX

mov ES,AX

inc CX

jmp mcb_lp

end_mcb:

ret

printMCB ENDP

freeingMemory PROC NEAR

push AX

push BX

push DX

```

        lea AX,ENDPROGRAMM
mov BX,10H ; size of paragraph
xor DX,DX
div BX

        inc AX
mov BX,AX
mov AH,4AH
int 21H

        pop DX

        pop BX

        pop AX

ret

freeingMemory ENDP

queryMemory PROC NEAR
    xor AX,AX
    mov BX,1000H
    mov AH,48H
    int 21H

    jnc final
    mov DX,offset perror
    call PRINT

final:

    ret
queryMemory ENDP

```


BEGIN:

```
call printMemorySize  
call printExtMemorySize  
    call queryMemory  
    call freeingMemory  
call printMCB
```

```
xor AL,AL  
    mov AH,4Ch  
    int 21H
```

ENDPROGRAMM:

MainSeg ENDS

END START