

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

Деткова А.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* — напечатать строку, адрес смещения до которой лежит в регистре DX.
2. *byte_to_dec* — записывает содержимое байта AL в память, адрес последнего символа в памяти находится в SI.
3. *tetr_to_hex*, *byte_to_hex*, *word_to_hex* — записывает в память, начиная с адреса последнего символа в строке, который лежит в DI, слово, лежащее в AX, в шест. представлении.
4. *hex_to_dec* — записывает число в регистре AX в память в десятичном представлении, по адресу SI.
5. *para_to_dec* — использует функцию *hex_to_dec*, переводит число в AX (размер в параграфах) в байты и записывает начиная с адреса SIю
6. *kbyte_to_byte* — число в AX дано в кбайтах, переводит в байты и записывает по адресу, начиная с SI, также использует функцию *hex_to_dec*.

7. *print_mem_size* — получает и выводит размер доступной памяти в байтах. Использует функцию *par_to_dec*. Использует строку *mem_size*.
8. *print_ext_mem_size* — получает и выводит размер расширенной памяти в байтах, использует функцию *kbyte_to_byte*. Использует строку *ext_mem_size*.
9. *one_mcb* — выводит один список *mcb* из списка списков (так называемый «List of lists») в строку адрес которой лежит в *SI*. Использует строку *mcb*.
10. *print_table_mcb* — выводит весь список списков.
11. *freeup_mem* — высвобождает неиспользуемую память.
12. *ask_mem* — запрашивает память в размере 64 КБ. При невозможности выделить запрашиваемую память выводится сообщение об ошибке *err_msg*.
13. *Main* — вызывает функции для выполнения задания.

Шаг 1.

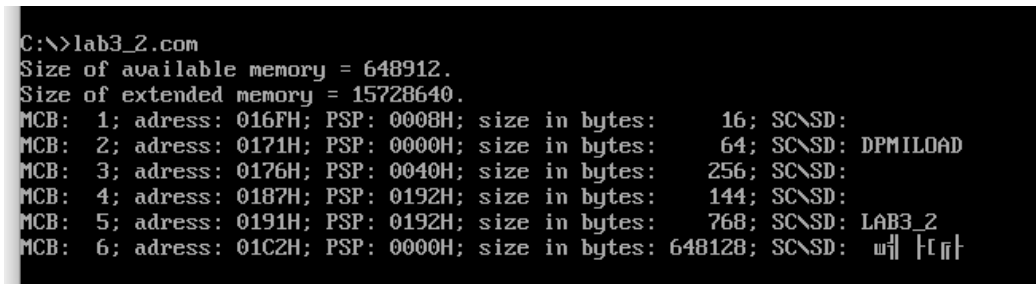
На первом шаге написана программа, которая печатает информацию о памяти устройства: количество доступной памяти, размер расширенной памяти, цепочку блоков управления памятью. Написаны и использованы функции *par_to_dec*, *kbyte_to_byte*, *print_mem_size*, *print_ext_mem_size*, *one_mcb*, *print_table_mcb*. Они позволяют выполнить требования: вывод информации о памяти компьютера. Результат работы программы представлен на 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: DPMILOAD
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.

На втором шаге добавлена функция `freeup_mem`, которая освобождает память, которую программа не занимает. Результат работы программы представлен на рас. 3.



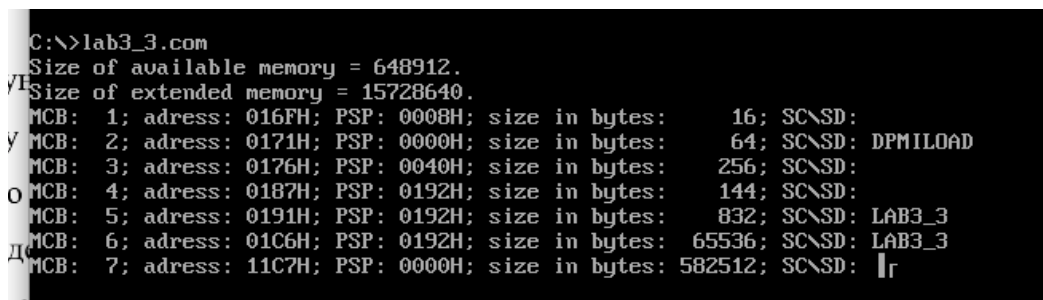
```
C:\>lab3_2.com
Size of available memory = 648912.
Size of extended memory = 15728640.
MCB: 1; adress: 016FH; PSP: 0008H; size in bytes: 16; SC\SD:
MCB: 2; adress: 0171H; PSP: 0000H; size in bytes: 64; SC\SD: DPMILOAD
MCB: 3; adress: 0176H; PSP: 0040H; size in bytes: 256; SC\SD:
MCB: 4; adress: 0187H; 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:  wll  lrl
```

Рисунок 2: Результат запуска второго варианта программы (модуль `lab3_2.com`)

При сравнении рисунков видно, что изначально (на первом шаге) 5 блок `msb` — 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 память, а пятый блок стал размером, который точно требуется для размещения программы.

Шаг 3.

На третьем шаге добавлена функция `ask_mem`, которая запрашиваем память размером 64 КБ. Функция вызывается после освобождения памяти. Результат работы программы представлен на рис. 3.



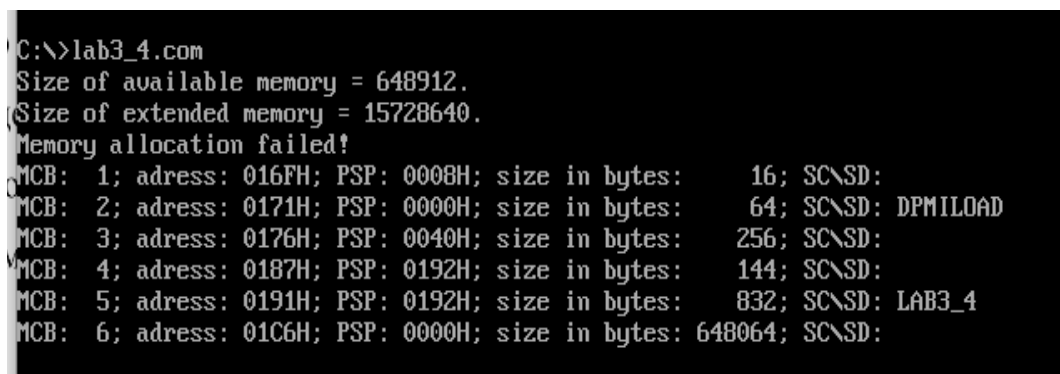
```
C:\>lab3_3.com
Size of available memory = 648912.
Size of extended memory = 15728640.
MCB: 1; adress: 016FH; PSP: 0008H; size in bytes: 16; SC\SD:
MCB: 2; adress: 0171H; PSP: 0000H; size in bytes: 64; SC\SD: DPMILOAD
MCB: 3; adress: 0176H; PSP: 0040H; size in bytes: 256; SC\SD:
MCB: 4; adress: 0187H; PSP: 0192H; size in bytes: 144; SC\SD:
MCB: 5; adress: 0191H; PSP: 0192H; size in bytes: 832; SC\SD: LAB3_3
MCB: 6; adress: 01C6H; PSP: 0192H; size in bytes: 65536; SC\SD: LAB3_3
MCB: 7; adress: 11C7H; PSP: 0000H; size in bytes: 582512; SC\SD: lrl
```

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.

На четвертом шаге выполнялись аналогичные действия, только выделение памяти осуществлялось до ее высвобождения. Результат работы программы представлен на рис. 4.



```
C:\>lab3_4.com
Size of available memory = 648912.
Size of extended memory = 15728640.
Memory allocation failed!
MCB: 1; adress: 016FH; PSP: 0008H; size in bytes: 16; SC\SD:
MCB: 2; adress: 0171H; PSP: 0000H; size in bytes: 64; SC\SD: DPMILOAD
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:
```

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.

Ответы на контрольные вопросы.

1. Что означает "доступный объем памяти"?

Объем оперативной памяти, которое может быть выделено для модуля программы (эта память может использоваться целиком или частично, ее можно запросить для модуля или высвободить, если не она не используется программой).

2. Где MCB блок Вашей программы в списке?

МСВ блок, который принадлежит программе, в графе SC\SD содержит имя модуля программы: LAB3_№ (№ - номер, соответствующий номеру шага в выполнении задания). На первом шаге — строка номер 5, на втором — тоже номер 5, на третьем — 5 и 6, на четвертом — 5.

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

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

start:
    jmp begin

data:
    mem_size db "Size of available memory =      .", 0DH, 0AH, "$"
    ext_mem_size db "Size of extended memory =      .", 0DH, 0AH,
"$"
    mcb db "MCB:      ; adress:      H; PSP:      H; size in bytes:      ;
SC\SD:      ", 0DH, 0AH, "$"

begin:
    call main
    xor AL,AL
    mov AH,4CH
    int 21H

_print PROC NEAR

    push AX

    mov AH, 09H
    int 21H

    pop AX

    ret

_print ENDP

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

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

kbyte_to_byte 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

kbyte_to_byte ENDP

print_mem_size PROC NEAR

        mov AH,4AH
        mov BX,0FFFFH
        int 21H

        mov AX,BX
        mov SI,offset mem_size + 32
        call par_to_dec
        mov DX,offset mem_size
        call _print
        ret

print_mem_size ENDP

print_ext_mem_size 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 ext_mem_size + 33
    mov BX,400H ; multiply 1024
    mul BX

    call kbyte_to_byte

    mov DX,offset ext_mem_size
    call _print
    ret

print_ext_mem_size ENDP

one_mcb 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

one_mcb ENDP

print_table_mcb PROC NEAR

    mov AH,52H
    int 21H

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

    xor CX,CX
    mov CX,1H

mcb_lp:
    call one_mcb

    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

print_table_mcb ENDP

main PROC NEAR

    call print_mem_size
    call print_ext_mem_size
    call print_table_mcb
    ret

main ENDP

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:
    mem_size db "Size of available memory =      .", 0DH, 0AH, "$"
    ext_mem_size db "Size of extended memory =      .", 0DH, 0AH,
"$"
    mcb db "MCB:      ; adress:      H; PSP:      H; size in bytes:      ;
SC\SD:      ", 0DH, 0AH, "$"

begin:
    call main
    xor AL,AL
    mov AH,4CH
    int 21H

_print PROC NEAR

    push AX

    mov AH, 09H
    int 21H

    pop AX

    ret

_print ENDP

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

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

kbyte_to_byte 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

kbyte_to_byte ENDP

print_mem_size PROC NEAR

        mov AH,4AH
        mov BX,0FFFFH
        int 21H

        mov AX,BX
        mov SI,offset mem_size + 32
        call par_to_dec
        mov DX,offset mem_size
        call _print
        ret

print_mem_size ENDP

print_ext_mem_size 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 ext_mem_size + 33
    mov BX,400H ; multiply 1024
    mul BX

    call kbyte_to_byte

    mov DX,offset ext_mem_size
    call _print
    ret

print_ext_mem_size ENDP

one_mcb 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

one_mcb ENDP

print_table_mcb PROC NEAR

        mov AH,52H
        int 21H

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

        xor CX,CX
        mov CX,1H

mcb_lp:
        call one_mcb

        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

print_table_mcb ENDP

freeup_mem PROC NEAR

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

        ret

freeup_mem ENDP

main PROC NEAR

```

```
    call print_mem_size
    call print_ext_mem_size
    call freeup_mem
    call print_table_mcb
    ret

main ENDP

end_progr:

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:
    mem_size db "Size of available memory =      .", 0DH, 0AH, "$"
    ext_mem_size db "Size of extended memory =      .", 0DH, 0AH,
"$"
    mcb db "MCB:      ; adress:      H; PSP:      H; size in bytes:      ;
SC\SD:      ", 0DH, 0AH, "$"
    err_msg db "Memory allocation failed!", 0DH, 0AH, "$"

begin:
    call main
    xor AL,AL
    mov AH,4CH
    int 21H

_print PROC NEAR

    push AX

    mov AH, 09H
    int 21H

    pop AX

    ret

_print ENDP

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

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

kbyte_to_byte 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

kbyte_to_byte ENDP

print_mem_size PROC NEAR

        mov AH,4AH
        mov BX,0FFFFH
        int 21H

        mov AX,BX
        mov SI,offset mem_size + 32
        call par_to_dec
        mov DX,offset mem_size
        call _print
        ret

print_mem_size ENDP

print_ext_mem_size 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 ext_mem_size + 33
    mov BX,400H ; multiply 1024
    mul BX

    call kbyte_to_byte

    mov DX,offset ext_mem_size
    call _print
    ret

print_ext_mem_size ENDP

one_mcb 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

one_mcb ENDP

print_table_mcb PROC NEAR

        mov AH,52H
        int 21H

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

        xor CX,CX
        mov CX,1H

mcb_lp:
        call one_mcb

        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

print_table_mcb ENDP

freeup_mem PROC NEAR

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

        ret

freeup_mem ENDP

```

```

ask_mem PROC NEAR

    xor AX,AX
    mov BX,1000H
    mov AH,48H
    int 21H

    jnc end_ask
    mov DX,offset err_msg
    call _print

end_ask:
    ret

ask_mem ENDP

main PROC NEAR

    call print_mem_size
    call print_ext_mem_size
    call freeup_mem
    call ask_mem
    call print_table_mcb
    ret

main ENDP

end_progr:

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:
    mem_size db "Size of available memory =      .", 0DH, 0AH, "$"
    ext_mem_size db "Size of extended memory =      .", 0DH, 0AH,
"$"
    mcb db "MCB:      ; adress:      H; PSP:      H; size in bytes:      ;
SC\SD:      ", 0DH, 0AH, "$"
    err_msg db "Memory allocation failed!", 0DH, 0AH, "$"

begin:
    call main
    xor AL,AL
    mov AH,4CH
    int 21H

_print PROC NEAR

    push AX

    mov AH, 09H
    int 21H

    pop AX

    ret

_print ENDP

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

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

kbyte_to_byte 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

kbyte_to_byte ENDP

print_mem_size PROC NEAR

        mov AH,4AH
        mov BX,0FFFFH
        int 21H

        mov AX,BX
        mov SI,offset mem_size + 32
        call par_to_dec
        mov DX,offset mem_size
        call _print
        ret

print_mem_size ENDP

print_ext_mem_size 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 ext_mem_size + 33
    mov BX,400H ; multiply 1024
    mul BX

    call kbyte_to_byte

    mov DX,offset ext_mem_size
    call _print
    ret

print_ext_mem_size ENDP

one_mcb 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

one_mcb ENDP

print_table_mcb PROC NEAR

        mov AH,52H
        int 21H

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

        xor CX,CX
        mov CX,1H

mcb_lp:
        call one_mcb

        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

print_table_mcb ENDP

freeup_mem PROC NEAR

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

        ret

freeup_mem ENDP

```

```

ask_mem PROC NEAR

    xor AX,AX
    mov BX,1000H
    mov AH,48H
    int 21H

    jnc end_ask
    mov DX,offset err_msg
    call _print

    end_ask:
        ret

ask_mem ENDP

main PROC NEAR

    call print_mem_size
    call print_ext_mem_size
    call freeup_mem
    call ask_mem
    call print_table_mcb
    ret

main ENDP

end_progr:

MainSeg ENDS
END start

```