

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

Студент гр.0382

Андрющенко К.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследование структуры данных и работы функций управления памятью ядра операционной системы.

Задание.

1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .COM, который выбирает и распечатывает следующую информацию:

- количество доступной памяти;
- размер расширенной памяти;
- выводит цепочку блоков управления памятью.

Адреса при выводе представляются шестнадцатеричными числами. Объем памяти функциями управления выводится в параграфах. Необходимо преобразовать его в байты и выводить в виде десятичных чисел. Последние восемь байт MSB 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.

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.

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

предыдущем шаге. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

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.

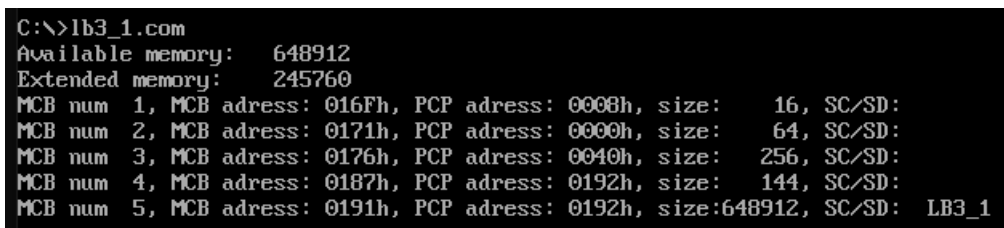
5. Оцените результаты, полученные на предыдущих шагах. Ответьте на контрольные вопросы и оформите отчет.

Выполнение работы.

1. За основу был взят шаблон .COM модуля из методического пособия, в котором реализованы процедуры преобразования двоичных кодов в символы шестнадцатеричных и десятичных чисел. В программу были добавлены следующие процедуры:

- процедура AM для вывода количества доступной памяти в байтах;
- процедура EM для вывода количества расширенной памяти в байтах;
- процедура MCB для вывода цепочки блоков управления памятью.

Результаты шага представлены на рисунке 1.



```
C:\>lb3_1.com
Available memory: 648912
Extended memory: 245760
MCB num 1, MCB address: 016Fh, PCP address: 0008h, size: 16, SC/SD:
MCB num 2, MCB address: 0171h, PCP address: 0000h, size: 64, SC/SD:
MCB num 3, MCB address: 0176h, PCP address: 0040h, size: 256, SC/SD:
MCB num 4, MCB address: 0187h, PCP address: 0192h, size: 144, SC/SD:
MCB num 5, MCB address: 0191h, PCP address: 0192h, size: 648912, SC/SD: LB3_1
```

Рисунок 1 – Результаты первого шага

Оценивая полученный результат, делаем вывод, что программа занимает всю доступную память.

2. Для выполнения данного шага в программу была добавлена процедура FREE_MEM. Результаты шага представлены на рисунке 2. На нем видно, что теперь программа занимает лишь ту область памяти, что необходима для ее хранения.

```

C:\>lb3_2.com
Available memory: 648912
Extended memory: 245760
MCB num 1, MCB address: 016Fh, PCP address: 0008h, size: 16, SC/SD:
MCB num 2, MCB address: 0171h, PCP address: 0000h, size: 64, SC/SD:
MCB num 3, MCB address: 0176h, PCP address: 0040h, size: 256, SC/SD:
MCB num 4, MCB address: 0187h, PCP address: 0192h, size: 144, SC/SD:
MCB num 5, MCB address: 0191h, PCP address: 0192h, size: 800, SC/SD: LB3_2
MCB num 6, MCB address: 01C4h, PCP address: 0000h, size: 648096, SC/SD:

```

Рисунок 2 – Результаты второго шага

3. Для выполнения данного шага в программу была добавлена процедура GET_MEM. Процедура вызывается после освобождения памяти процедурой FREE_MEM. Результаты представлены на рисунке 3.

```

C:\>lb3_3
Available memory: 648912
I get extra memory)
Extended memory: 245760
MCB num 1, MCB address: 016Fh, PCP address: 0008h, size: 16, SC/SD:
MCB num 2, MCB address: 0171h, PCP address: 0000h, size: 64, SC/SD:
MCB num 3, MCB address: 0176h, PCP address: 0040h, size: 256, SC/SD:
MCB num 4, MCB address: 0187h, PCP address: 0192h, size: 144, SC/SD:
MCB num 5, MCB address: 0191h, PCP address: 0192h, size: 880, SC/SD: LB3_3
MCB num 6, MCB address: 01C9h, PCP address: 0192h, size: 65536, SC/SD: LB3_3
MCB num 7, MCB address: 11CAh, PCP address: 0000h, size: 582464, SC/SD:

```

Рисунок 3 – Результаты третьего шага

В результате выполнения этого шага видно, что программе, после освобождения памяти, был выделен еще один блок памяти в размере 64Кб.

4. На этом шаге процедура GET_MEM вызывается до освобождения памяти процедурой FREE_MEM. Результаты представлены на рисунке 4.

```

C:\>lb3_4
Available memory: 648912
Extended memory: 245760
I cannot get extra memory(
MCB num 1, MCB address: 016Fh, PCP address: 0008h, size: 16, SC/SD:
MCB num 2, MCB address: 0171h, PCP address: 0000h, size: 64, SC/SD:
MCB num 3, MCB address: 0176h, PCP address: 0040h, size: 256, SC/SD:
MCB num 4, MCB address: 0187h, PCP address: 0192h, size: 144, SC/SD:
MCB num 5, MCB address: 0191h, PCP address: 0192h, size: 880, SC/SD: LB3_4
MCB num 6, MCB address: 01C9h, PCP address: 0000h, size: 648016, SC/SD:

```

Рисунок 4 – Результаты четвертого шага

В третьей строке вывода сообщается о невозможности получения дополнительной памяти. Это действительно так, поскольку до освобождения памяти программа занимала всю доступную память.

Исходный код программы см. в приложении А.

Ответы на вопросы.

1. Что означает “доступный объем памяти”?

Это объем памяти, выделенный управляющей программой для модуля

2. Где MCB блок вашей программы в списке?

Блок MCB программы в списке – это тот, у которого в графе SC/SD написано название программы (на третьем шаге таких блока два, т.к. выделена доп. память). На шаге 1 – это последний в списке блок, на шаге 2 – блок памяти программы предпоследний, за ним – блок свободной памяти, на шаге 3 – блоки 5 и 6, последний блок – свободный, на шаге 4 – предпоследний блок, за ним – блок свободной памяти.

3. Какой размер памяти занимает программа в каждом случае?

На шаге 1 – 648912 байта, на шаге 2 – 800 байт, на шаге 3 – 66416 байт (необходимая память после освобождения + дополнительно выделенная), шаг 4 – 880 байт.

Выводы.

В ходе работы были исследованы структуры данных и работа функций управления памятью ядра операционной системы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3_1.asm

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

; ДАННЫЕ
AVAIL_MEM db 'Available memory:           ', 0DH, 0AH, '$'
EXTEND_MEM db 'Extended memory:           ', 0DH, 0AH, '$'
MCB_I db 'MCB num      , MCB adress:      h, PCP adress:      h,
size:      , SC/SD:      ', 0DH, '$'

; ПРОЦЕДУРЫ
TETR_TO_HEX PROC near
    and AL, 0Fh
    cmp AL, 09
    jbe NEXT
    add AL, 07
NEXT:    add AL, 30h
    ret
TETR_TO_HEX ENDP
;-----
BYTE_TO_HEX PROC near
; байт в AL переводится в два символа 16-го числа в AX
    push CX
    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL
    call TETR_TO_HEX ; в AL старшая цифра
    pop CX           ; в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, в DI - адрес последнего символа
    push BX
    mov BH, AH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    dec DI
    mov AL, BH
```

```

        call BYTE_TO_HEX
        mov [DI], AH
        dec DI
        mov [DI], AL
        pop BX
        ret
WRD_TO_HEX ENDP
;-----
BYTE_TO_DEC PROC near
; перевод в 10 с/с, в SI - адрес поля младшей цифры
        push AX
        push CX
        push DX
        xor AH, AH
        xor DX, DX
        mov CX, 10
loop_bd: div CX
        or DL, 30h
        mov [SI], DL
        dec SI
        xor DX, DX
        cmp AX, 10
        jae loop_bd
        cmp AL, 00h
        je end_1
        or AL, 30h
        mov [SI], AL
end_1:
        pop DX
        pop CX
        pop AX
        ret
BYTE_TO_DEC ENDP
;-----
WORD_TO_DEC PROC near
        push AX
        push BX
        push DX
        push CX
        push SI

        mov BX, 10h
        mul BX
        mov BX, 0Ah
division:
        div BX
        or DX, 30h
        mov [SI], DL
        dec SI
        xor DX, DX
        cmp AX, 0h

```

```

        jne division

        pop SI
        pop CX
        pop DX
        pop BX
        pop AX
        ret
WORD_TO_DEC ENDP
;-----
PRINT PROC near
        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
PRINT ENDP
;-----
PRINT_SYM PROC near
        push AX
        mov AH, 02h
        int 21h
        pop AX
        ret
PRINT_SYM ENDP
;-----
AM PROC near
        push AX
        push BX
        push SI

        xor AX, AX
        mov AH, 4Ah
        mov BX, 0FFFFh
        int 21h
        mov AX, BX
        mov SI, offset AVAIL_MEM
        add SI, 25
        call WORD_TO_DEC
        mov DX, offset AVAIL_MEM
        call PRINT

        pop SI
        pop BX
        pop AX
        ret
AM ENDP
;-----
EM PROC near
        push AX
        push BX

```



```

    push SI
    xor AX, AX
    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 EXTEND_MEM
    add SI, 25
    call WORD_TO_DEC
    mov DX, offset EXTEND_MEM
    call PRINT
    pop SI
    pop BX
    pop AX
    ret
EM ENDP
;-----
MCB PROC near
    push AX
    push BX
    push CX
    push DX
    push ES
    push SI

    xor AX, AX
    mov AH, 52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES, AX
    mov CL, 1
loop_mcb:
    mov AL, CL
    mov SI, offset MCB_I
    add SI, 9
    call BYTE_TO_DEC

    mov AX, ES
    mov DI, offset MCB_I
    add DI, 27
    call WRD_TO_HEX

    mov AX, ES:[01h]
    mov DI, offset MCB_I
    add DI, 46
    call WRD_TO_HEX

```

```

    mov AX, ES:[03h]
    add SI, 52
    call WORD_TO_DEC

    mov BX, 8
    push CX
    mov CX, 7
    add SI, 11
loop_sc_sd:
    mov DX, ES:[BX]
    mov DS:[SI], DX
    inc BX
    inc SI
    loop loop_sc_sd

    mov DX, offset MCB_I
    call PRINT

    mov AH, ES:[0]
    cmp AH, 5Ah
    je end_mcb

    mov BX, ES:[3]
    mov AX, ES
    add AX, BX
    inc AX
    mov ES, AX
    pop CX
    inc CL
    jmp loop_mcb
end_mcb:
    pop SI
    pop ES
    pop DX
    pop CX
    pop BX
    pop AX
    ret
MCB ENDP
;-----
BEGIN:
    call AM
    call EM
    call MCB
    xor AL, AL
    mov AH, 4Ch
    int 21h
TESTPC ENDS
    END START

```

Название файла: lb3_2.asm

TESTPC SEGMENT

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

ORG 100H

START: jmp BEGIN

; ДАННЫЕ

AVAIL_MEM db 'Available memory: ', 0DH, 0AH, '\$'

EXTEND_MEM db 'Extended memory: ', 0DH, 0AH, '\$'

MCB_I db 'MCB num ', MCB adress: h, PCP adress: h,
size: , SC/SD: ', 0DH, '\$'

ERROR db 'I can't get extra memory(', 0DH, 0AH, '\$'

SUCCEC db 'I get extra memory)', 0DH, 0AH, '\$'

; ПРОЦЕДУРЫ

TETR_TO_HEX PROC near

and AL, 0Fh

cmp AL, 09

jbe NEXT

add AL, 07

NEXT: add AL, 30h

ret

TETR_TO_HEX ENDP

;-----

BYTE_TO_HEX PROC near

; байт в AL переводится в два символа 16-го числа в AX

push CX

mov AH, AL

call TETR_TO_HEX

xchg AL, AH

mov CL, 4

shr AL, CL

call TETR_TO_HEX ; в AL старшая цифра

pop CX ; в AH младшая

ret

BYTE_TO_HEX ENDP

;-----

WRD_TO_HEX PROC near

; перевод в 16 с/с 16-ти разрядного числа

; в AX - число, в DI - адрес последнего символа

push BX

mov BH, AH

call BYTE_TO_HEX

mov [DI], AH

dec DI

mov [DI], AL

dec DI

mov AL, BH

call BYTE_TO_HEX

mov [DI], AH

dec DI

mov [DI], AL

```

        pop BX
        ret
WRD_TO_HEX ENDP
;-----
BYTE_TO_DEC PROC near
; перевод в 10 с/с, в SI - адрес поля младшей цифры
        push AX
        push CX
        push DX
        xor AH, AH
        xor DX, DX
        mov CX, 10
loop_bd: div CX
        or DL, 30h
        mov [SI], DL
        dec SI
        xor DX, DX
        cmp AX, 10
        jae loop_bd
        cmp AL, 00h
        je end_1
        or AL, 30h
        mov [SI], AL
end_1:
        pop DX
        pop CX
        pop AX
        ret
BYTE_TO_DEC ENDP
;-----
WORD_TO_DEC PROC near
        push AX
        push BX
        push DX
        push CX
        push SI

        mov BX, 10h
        mul BX
        mov BX, 0Ah
division:
        div BX
        or DX, 30h
        mov [SI], DL
        dec SI
        xor DX, DX
        cmp AX, 0h
        jne division

        pop SI
        pop CX

```

```

        pop DX
        pop BX
        pop AX
        ret
WORD_TO_DEC ENDP
;-----
PRINT PROC near
        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
PRINT ENDP
;-----
PRINT_SYM PROC near
        push AX
        mov AH, 02h
        int 21h
        pop AX
        ret
PRINT_SYM ENDP
;-----
AM PROC near
        push AX
        push BX
        push SI

        xor AX, AX
        mov AH, 4Ah
        mov BX, 0FFFFh
        int 21h
        mov AX, BX
        mov SI, offset AVAIL_MEM
        add SI, 25
        call WORD_TO_DEC
        mov DX, offset AVAIL_MEM
        call PRINT

        pop SI
        pop BX
        pop AX
        ret
AM ENDP
;-----
EM PROC near
        push AX
        push BX
        push SI
        xor AX, AX
        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 EXTEND_MEM
    add SI, 25
    call WORD_TO_DEC
    mov DX, offset EXTEND_MEM
    call PRINT
    pop SI
    pop BX
    pop AX
    ret
EM ENDP
;-----
MCB PROC near
    push AX
    push BX
    push CX
    push DX
    push ES
    push SI

    xor AX, AX
    mov AH, 52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES, AX
    mov CL, 1
loop_mcb:
    mov AL, CL
    mov SI, offset MCB_I
    add SI, 9
    call BYTE_TO_DEC

    mov AX, ES
    mov DI, offset MCB_I
    add DI, 27
    call WRD_TO_HEX

    mov AX, ES:[01h]
    mov DI, offset MCB_I
    add DI, 46
    call WRD_TO_HEX

    mov AX, ES:[03h]
    add SI, 52
    call WORD_TO_DEC

```

```

        mov BX, 8
        push CX
        mov CX, 7
        add SI, 11
loop_sc_sd:
        mov DX, ES:[BX]
        mov DS:[SI], DX
        inc BX
        inc SI
        loop loop_sc_sd

        mov DX, offset MCB_I
        call PRINT

        mov AH, ES:[0]
        cmp AH, 5Ah
        je end_mcb

        mov BX, ES:[3]
        mov AX, ES
        add AX, BX
        inc AX
        mov ES, AX
        pop CX
        inc CL
        jmp loop_mcb
end_mcb:
        pop SI
        pop ES
        pop DX
        pop CX
        pop BX
        pop AX
        ret
MCB ENDP
;-----
FREE_MEM PROC NEAR
        push AX
        push BX
        push DX

        lea AX, end_programm
        mov BX, 10h
        xor DX, DX
        div BX
        inc AX
        mov BX, AX
        xor AX, AX
        mov AH, 4Ah
        int 21h

```

```

        pop DX
        pop BX
        pop AX
        ret
FREE_MEM ENDP
;-----
GET_MEM PROC near
        mov BX, 1000h
        xor AX, AX
        mov AH, 48h
        int 21h

GET_MEM ENDP
;-----
BEGIN:
        call AM
        call FREE_MEM
        call EM
        call MCB
        xor AL, AL
        mov AH, 4Ch
        int 21h
end_programm:
TESTPC ENDS
        END START

```

Название файла: lb3_3.asm

```

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

; ДАННЫЕ
AVAIL_MEM db 'Available memory:          ', 0DH, 0AH, '$'
EXTEND_MEM db 'Extended memory:          ', 0DH, 0AH, '$'
MCB_I db 'MCB num      , MCB adress:      h, PCP adress:      h,
size:    , SC/SD:          ', 0DH, '$'
ERROR db 'I cannot get extra memory(', 0DH, 0AH, '$'
SUCCESS db 'I get extra memory)', 0DH, 0AH, '$'
; ПРОЦЕДУРЫ
TETR_TO_HEX PROC near
        and AL, 0Fh
        cmp AL, 09
        jbe NEXT
        add AL, 07
NEXT:    add AL, 30h
        ret
TETR_TO_HEX ENDP
;-----

```



```

BYTE_TO_HEX PROC near
; байт в AL переводится в два символа 16-го числа в AX
    push CX
    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL
    call TETR_TO_HEX ; в AL старшая цифра
    pop CX           ; в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, в DI - адрес последнего символа
    push BX
    mov BH, AH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    dec DI
    mov AL, BH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    pop BX
    ret
WRD_TO_HEX ENDP
;-----
BYTE_TO_DEC PROC near
; перевод в 10 с/с, в SI - адрес поля младшей цифры
    push AX
    push CX
    push DX
    xor AH, AH
    xor DX, DX
    mov CX, 10
loop_bd: div CX
    or DL, 30h
    mov [SI], DL
    dec SI
    xor DX, DX
    cmp AX, 10
    jae loop_bd
    cmp AL, 00h
    je end_1
    or AL, 30h
    mov [SI], AL

```

```

end_1:
    pop DX
    pop CX
    pop AX
    ret
BYTE_TO_DEC ENDP
;-----
WORD_TO_DEC PROC near
    push AX
    push BX
    push DX
    push CX
    push SI

    mov BX, 10h
    mul BX
    mov BX, 0Ah
division:
    div BX
    or DX, 30h
    mov [SI], DL
    dec SI
    xor DX, DX
    cmp AX, 0h
    jne division

    pop SI
    pop CX
    pop DX
    pop BX
    pop AX
    ret
WORD_TO_DEC ENDP
;-----
PRINT PROC near
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
PRINT ENDP
;-----
PRINT_SYM PROC near
    push AX
    mov AH, 02h
    int 21h
    pop AX
    ret
PRINT_SYM ENDP
;-----
AM PROC near

```

```

    push AX
    push BX
    push SI

    xor AX, AX
    mov AH, 4Ah
    mov BX, 0FFFFh
    int 21h
    mov AX, BX
    mov SI, offset AVAIL_MEM
    add SI, 25
    call WORD_TO_DEC
    mov DX, offset AVAIL_MEM
    call PRINT

    pop SI
    pop BX
    pop AX
    ret
AM ENDP
;-----
EM PROC near
    push AX
    push BX
    push SI
    xor AX, AX
    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 EXTEND_MEM
    add SI, 25
    call WORD_TO_DEC
    mov DX, offset EXTEND_MEM
    call PRINT
    pop SI
    pop BX
    pop AX
    ret
EM ENDP
;-----
MCB PROC near
    push AX
    push BX
    push CX
    push DX

```

```

push ES
push SI

xor AX, AX
mov AH, 52h
int 21h
mov AX, ES:[BX-2]
mov ES, AX
mov CL, 1
loop_mcb:
mov AL, CL
mov SI, offset MCB_I
add SI, 9
call BYTE_TO_DEC

mov AX, ES
mov DI, offset MCB_I
add DI, 27
call WRD_TO_HEX

mov AX, ES:[01h]
mov DI, offset MCB_I
add DI, 46
call WRD_TO_HEX

mov AX, ES:[03h]
add SI, 52
call WORD_TO_DEC

mov BX, 8
push CX
mov CX, 7
add SI, 11
loop_sc_sd:
mov DX, ES:[BX]
mov DS:[SI], DX
inc BX
inc SI
loop loop_sc_sd

mov DX, offset MCB_I
call PRINT

mov AH, ES:[0]
cmp AH, 5Ah
je end_mcb

mov BX, ES:[3]
mov AX, ES
add AX, BX
inc AX

```

```

        mov ES, AX
        pop CX
        inc CL
        jmp loop_mcb
end_mcb:
        pop SI
        pop ES
        pop DX
        pop CX
        pop BX
        pop AX
        ret
MCB ENDP
;-----
FREE_MEM PROC NEAR
        push AX
        push BX
        push DX

        lea AX, end_programm
        mov BX, 10h
        xor DX, DX
        div BX
        inc AX
        mov BX, AX
        xor AX, AX
        mov AH, 4Ah
        int 21h

        pop DX
        pop BX
        pop AX
        ret
FREE_MEM ENDP
;-----
GET_MEM PROC near
        push AX
        push BX
        push DX

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

        jc CF_ERROR

        mov DX, offset SUCCESS
        call PRINT
        jmp end_gm

```

```

CF_ERROR:
    mov DX, offset ERROR
    call PRINT
    jmp end_gm

end_gm:
    pop DX
    pop BX
    pop AX
    ret
GET_MEM ENDP
;-----
BEGIN:
    call AM
    call EM
    call FREE_MEM
    call GET_MEM
    call MCB
    xor AL, AL
    mov AH, 4Ch
    int 21h
end_programm:
TESTPC ENDS
    END START

```

Название файла: lb3_4.asm

```

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

; ДАННЫЕ
AVAIL_MEM db 'Available memory:          ', 0DH, 0AH, '$'
EXTEND_MEM db 'Extended memory:          ', 0DH, 0AH, '$'
MCB_I db 'MCB num      , MCB adress:      h, PCP adress:      h,
size:      , SC/SD:          ', 0DH, '$'
ERROR db 'I cannot get extra memory(', 0DH, 0AH, '$'
SUCCESS db 'I get extra memory)', 0DH, 0AH, '$'
; ПРОЦЕДУРЫ
TETR_TO_HEX PROC near
    and AL, 0Fh
    cmp AL, 09
    jbe NEXT
    add AL, 07
NEXT:    add AL, 30h
    ret
TETR_TO_HEX ENDP
;-----
BYTE_TO_HEX PROC near
; байт в AL переводится в два символа 16-го числа в AX
    push CX

```

```

    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL
    call TETR_TO_HEX ; в AL старшая цифра
    pop CX           ; в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, в DI - адрес последнего символа
    push BX
    mov BH, AH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    dec DI
    mov AL, BH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    pop BX
    ret
WRD_TO_HEX ENDP
;-----
BYTE_TO_DEC PROC near
; перевод в 10 с/с, в SI - адрес поля младшей цифры
    push AX
    push CX
    push DX
    xor AH, AH
    xor DX, DX
    mov CX, 10
loop_bd: div CX
    or DL, 30h
    mov [SI], DL
    dec SI
    xor DX, DX
    cmp AX, 10
    jae loop_bd
    cmp AL, 00h
    je end_1
    or AL, 30h
    mov [SI], AL
end_1:
    pop DX
    pop CX

```

```

        pop AX
        ret
BYTE_TO_DEC ENDP
;-----
WORD_TO_DEC PROC near
    push AX
    push BX
    push DX
    push CX
    push SI

    mov BX, 10h
    mul BX
    mov BX, 0Ah
division:
    div BX
    or DX, 30h
    mov [SI], DL
    dec SI
    xor DX, DX
    cmp AX, 0h
    jne division

    pop SI
    pop CX
    pop DX
    pop BX
    pop AX
    ret
WORD_TO_DEC ENDP
;-----
PRINT PROC near
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
PRINT ENDP
;-----
PRINT_SYM PROC near
    push AX
    mov AH, 02h
    int 21h
    pop AX
    ret
PRINT_SYM ENDP
;-----
AM PROC near
    push AX
    push BX
    push SI

```



```

        xor AX, AX
        mov AH, 4Ah
        mov BX, 0FFFFh
        int 21h
        mov AX, BX
        mov SI, offset AVAIL_MEM
        add SI, 25
        call WORD_TO_DEC
        mov DX, offset AVAIL_MEM
        call PRINT

        pop SI
        pop BX
        pop AX
        ret
AM ENDP
;-----
EM PROC near
    push AX
    push BX
    push SI
    xor AX, AX
    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 EXTEND_MEM
    add SI, 25
    call WORD_TO_DEC
    mov DX, offset EXTEND_MEM
    call PRINT
    pop SI
    pop BX
    pop AX
    ret
EM ENDP
;-----
MCB PROC near
    push AX
    push BX
    push CX
    push DX
    push ES
    push SI

```

```

        xor AX, AX
        mov AH, 52h
        int 21h
        mov AX, ES:[BX-2]
        mov ES, AX
        mov CL, 1
loop_mcb:
        mov AL, CL
        mov SI, offset MCB_I
        add SI, 9
        call BYTE_TO_DEC

        mov AX, ES
        mov DI, offset MCB_I
        add DI, 27
        call WRD_TO_HEX

        mov AX, ES:[01h]
        mov DI, offset MCB_I
        add DI, 46
        call WRD_TO_HEX

        mov AX, ES:[03h]
        add SI, 52
        call WORD_TO_DEC

        mov BX, 8
        push CX
        mov CX, 7
        add SI, 11
loop_sc_sd:
        mov DX, ES:[BX]
        mov DS:[SI], DX
        inc BX
        inc SI
        loop loop_sc_sd

        mov DX, offset MCB_I
        call PRINT

        mov AH, ES:[0]
        cmp AH, 5Ah
        je end_mcb

        mov BX, ES:[3]
        mov AX, ES
        add AX, BX
        inc AX
        mov ES, AX
        pop CX
        inc CL

```

```

        jmp loop_mcb
end_mcb:
        pop SI
        pop ES
        pop DX
        pop CX
        pop BX
        pop AX
        ret
MCB ENDP
;-----
FREE_MEM PROC NEAR
        push AX
        push BX
        push DX

        lea AX, end_programm
        mov BX, 10h
        xor DX, DX
        div BX
        inc AX
        mov BX, AX
        xor AX, AX
        mov AH, 4Ah
        int 21h

        pop DX
        pop BX
        pop AX
        ret
FREE_MEM ENDP
;-----
GET_MEM PROC near
        push AX
        push BX
        push DX

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

        jc CF_ERROR

        mov DX, offset SUCCESS
        call PRINT
        jmp end_gm

CF_ERROR:
        mov DX, offset ERROR
        call PRINT

```

```

        jmp end_gm

end_gm:
        pop DX
        pop BX
        pop AX
        ret
GET_MEM ENDP
;-----
BEGIN:
        call AM
        call EM
        call GET_MEM
        call FREE_MEM
        call MCB
        xor AL, AL
        mov AH, 4Ch
        int 21h
end_programm:
TESTPC ENDS
        END START

```