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 Самулевич В. А.

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 не страничная 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, предусматривают и преобразуют этот список.

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

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) Выводит цепочку блоков управления памятью.

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

Запустите программу и внимательно оцените результаты. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 2. Измените программу таким образом, чтобы она освобождала память, которую она не занимает. Для этого используйте функцию 4Ah прерывания 21h (пример в разделе «Использование функции 4AH»). Повторите эксперимент, запустив модифицированную программу.

Сравните выходные данные с результатами, полученными на предыдущем шаге. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 3. Измените программу еще раз таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти функцией 48H прерывания 21H. Повторите эксперимент, запустив модифицированную программу. Сравните выходные данные с результатами, полученными на предыдущих шагах. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 4. Измените первоначальный вариант программы, запросив 64Кб памяти функцией 48H прерывания 21H до освобождения памяти. Обязательно обрабатывайте завершение функций ядра, проверяя флаг CF. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

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

Выполнение работы.

Шаг 1:

Для получения требуемой информации и вывода её на экран, были реализованы следующие функции:

- print_available_memory – с помощью int 21h(AH = 4ah),получает размер доступной для программы памяти, после чего заносит это число в строку available_memory и печатает результат.

- print_extended_memory – Используя ячейки 30h и 31h CMOS получает размер расширенной памяти в килобайтах, переводит их в байты и выводит получившееся число на экран, вместе со строкой extended_memory.

- print_one_MCB – выводит информацию об одном блоке памяти в следующем формате: MCB#<номер блока>, address: <сегментный адрес блока> , owner: <сегментный адрес владельца блока>, size: <размер блока в байтах>, SD/SC: < 8 символов>

- print_all_MCB – используя функцию print_one_MCB, выводит информацию обо всех блоках управления памятью.

- Clear_MCB_info –Удаляет из строки MCB_info все числа. Вызывается в конце print_one_MCB.

- kilobytes_to_bytes – переводит килобайты, количество которых указано в AX, в байты и записывает получившееся число в память, начиная со смещения DI.

- paragraphs_to_bytes – переводит параграфы, указанные в AX, в байты и записывает результат в память, начиная со смещения DI.

- division_32 – делит 32 битное число, старшие 2 байта которого указаны в DX, а младшие – в AX, на 16 битное, указанное в BX. Старшие 2 байта результата помещаются в DX, младшие - в AX, а остаток – в CX. Эта функция используется при переводе килобайт в байты.

- byte_to_dec и word_to_hex - вспомогательные функции, строящие по числу его строковое представление в 10 и 16 сс. соответственно.

- print – с помощью прерывания int 21h(AH = 09h) выводит на экран строку, которая начинается со смещения, указанного в DX и заканчивается символом ‘$’.

Результат работы программы, написанной на первом шаге, представлен на рисунке 1.

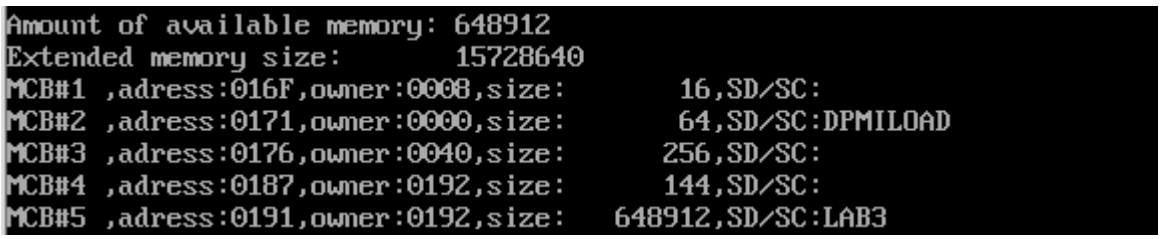

Рисунок 1- результат работы программы на 1 шаге.

Шаг 2:

С целью освобождения не занимаемой памяти была реализована функция free_memory.Результат работы программы на втором шаге представлен на рисунке 2.

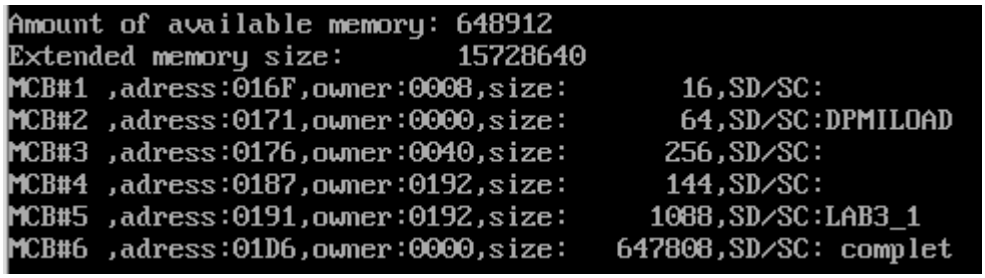

Рисунок 2 – результат работы программы на 2 шаге.

Можно заметить, что на этом шаге программа занимает не всю доступную память (как это было в шаге 1), а лишь необходимый минимум – 1088байт (MCB#5). В свою очередь оставшееся свободная память выделилась в отдельный блок(MCB#6).

Шаг 3:

Для запроса дополнительных 64 килобайт памяти была реализована функция request_memory. Ее вызов происходит сразу после освобождения незанятой памяти. Полученный результат работы представлен на рисунке 3.

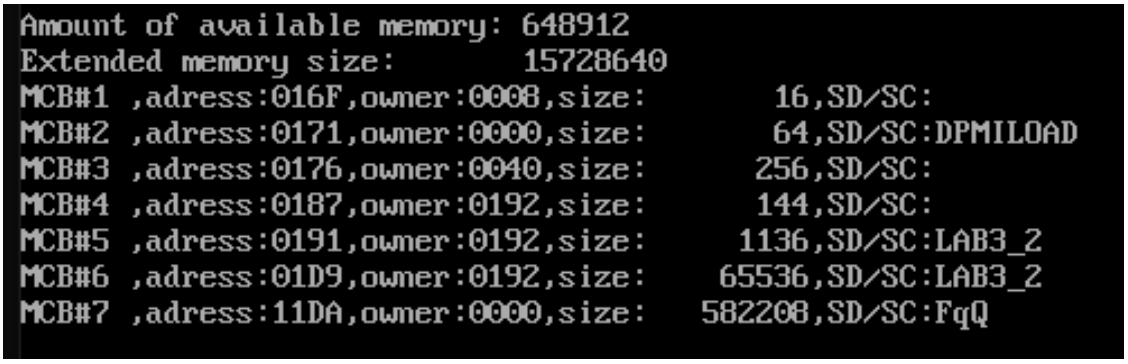

Рисунок 3 – результат работы программы на 3 шаге.

В этой модификации под программу отводятся два блока – MCB#5, содержащий непосредственно код, и MCB#6, в котором лежат выделенные 64 килобайта.

Шаг 4:

На этом шаге запрос дополнительных 64 килобайт был поставлен до освобождения памяти. Полученный результат показан на рисунке 4.

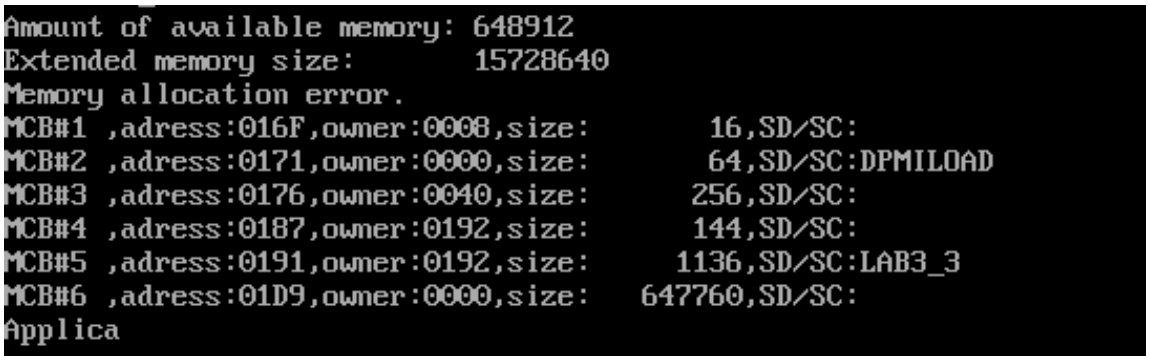

Рисунок 4- результат работы программы на 4 шаге.

В этом случае, т.к. программа уже занимала всю доступную память, выделение еще 64 кб прошло неудачно, о чем вывелось соответствующее сообщение.

Исходный код программ см. в приложении А.

Ответы на контрольные вопросы.

1) Что означает доступный объем памяти?

   Доступный объем памяти – количество оперативной памяти, которым может пользоваться программа в процессе своего выполнения.

2) Где MCB блок вашей программы в списке?

   На 1,2 и 4 шаге – MCB#5, на 6 шаге – MCB#5 и MCB#6.

3) Какой размер памяти занимает программа в каждом случае?

   На 1 шаге – весь доступный объем памяти (648912 байт), на 2 шаге и 4 – минимально необходимый объем (1088 байт и 1136 байт соответственно), на 3 шаге – минимально необходимый объем (1136 байт) + 64 килобайта.

Выводы.

В ходе работы было произведено исследование организации управления основной памятью, а также работа функций управления памятью ядра операционной системы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab3.asm

```
CODE SEGMENT
ORG 100h
ASSUME CS:CODE, DS:CODE,CS:CODE,ES:NOTHING

    programm_start: jmp MAIN
    available_memory DB "Amount of available memory:
",0DH,0AH,'$'
    extended_memory DB "Extended memory size:
",0DH,0AH,'$'
    MCB_info DB
"MCB#  ,adress:    ,owner:    ,size:         ,SD/SC:        ",0DH,0AH,'$'


MAIN:

    call print_available_memory
    call print_extended_memory
    call print_all_MCB
    xor AL,AL
    mov AH, 4ch
    int 21h

print PROC NEAR
    push AX
    mov AH,09h
    int 21h
    pop AX
ret
print ENDP



CLEAR_MCB_INFO PROC NEAR; SI - начало строки
    push SI
    push AX
    push BX

    mov BX,0
clear_loop:
    mov AL,[SI]
    cmp AL,36
    je clear_loop_end
    cmp AL,58
    je clear_active_mode
    cmp AL,35
    je clear_active_mode
    cmp AL,44
    je clear_passive_mode
    cmp AL, 0dh
    je clear_passive_mode
    jmp clear_continue
clear_active_mode:
```

```
    mov BX,1
    inc SI
    jmp clear_loop

clear_passive_mode:
    mov BX,0
    inc SI
    jmp clear_loop

clear_continue:
    cmp BX,1
    je change
    inc SI
    jmp clear_loop

change:
    mov AL,32
    mov [SI],AL
    inc SI
    jmp clear_loop

clear_loop_end:
    pop BX
    pop AX
    pop SI
    ret
CLEAR_MCB_INFO ENDP


print_one_MCB PROC NEAR ;ES-адрес начала блока, CX- счетчик
    push AX
    push SI
    push DI
    push CX
    push DX


    mov AL,CL
    mov SI,offset MCB_info
    add SI,4
    call byte_to_dec
    mov AX,ES
    mov DI,offset MCB_info
    add DI,17
    call wrd_to_hex
    mov AX, ES:[1]
    mov DI,offset MCB_info
    add DI,28
    call wrd_to_hex
    mov AX,ES:[3]
    mov DI,offset MCB_info
    add DI,43
    call paragraphs_to_bytes
    mov DI,8
    mov CX,8
    mov SI,offset MCB_info
    add SI,51
bytes_loop:
```

```
    mov AL,ES:[DI]
    mov [SI],AL
    inc DI
    inc SI
    loop bytes_loop

    mov DX, offset MCB_info
    call print
    mov SI,offset MCB_info
    call CLEAR_MCB_INFO
    pop DX
    pop CX
    pop DI
    pop SI
    pop AX
    ret
print_one_MCB ENDP


print_all_MCB PROC NEAR
    push AX
    push BX
    push CX
    push ES

    mov AH,52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES,AX
    mov CX,1
print_all_loop:
    call print_one_MCB
    cmp BYTE PTR ES:[0], 05ah
    je end_loop
    mov AX,ES
    mov BX, ES:[3]
    add AX, BX
    inc AX
    mov ES,AX
    inc CX
    jmp print_all_loop

end_loop:
    pop ES
    pop CX
    pop BX
    pop AX
    ret
print_all_MCB ENDP

print_extended_memory PROC NEAR
    push AX
    push BX
    push DX
    push DI

    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 DI,offset extended_memory
        add DI,35
        call kilobytes_to_bytes
        mov DX,offset extended_memory
        call print

        pop DI
        pop DX
        pop BX
        pop AX
        ret
    print_extended_memory ENDP


    kilobytes_to_bytes PROC NEAR; AX - количество килобайт, DI - адрес
последнего символа результата
        push AX
        push BX
        push DX
        push CX
        push DI

        mov DX,0

        mov BX,1024
        mul BX
        mov BX,10
    kilobytes_cycle:
        cmp DX,0
        jne check_passed
        cmp AX,10
        ja check_passed
        add AX,48
        mov [DI],AL
        jmp kilobytes_end

    check_passed:
        call division_32
        add CX,48
        mov [DI],CL
        dec DI
       jmp kilobytes_cycle

    kilobytes_end:

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

```
kilobytes_to_bytes ENDP

paragraphs_to_bytes PROC NEAR ; AX-число параграфов, DI-арес
последнего символа,куда надо записать результат
        push BX
        push DX
        push AX
        push DI

        mov DX,0
        mov BX, 16
        mul BX
        mov BX,10
    paragraphs_start:
        cmp DX,0
        jne paragraphs_skip_check
        cmp AX,10
        jb paragraphs_end
    paragraphs_skip_check:
        div BX
        add DX,48
        mov [DI],DL
        dec DI
        mov DX,0
        jmp paragraphs_start
    paragraphs_end:
        add AX,48
        mov [DI],AL

        pop DI
        pop AX
        pop DX
        pop BX
        ret
    paragraphs_to_bytes ENDP


    division_32 PROC NEAR
        push DI
        push BX

        jmp division_begin
        HIGHT DW 0
        LOW  DW  0
        DIVIDER DW 0
        REMAINDER_HIGHT DW 0
        REMAINDER_LOW DW 0
        TEMP_LOW DW 0

    division_begin:

        mov [HIGHT],DX
        mov [LOW],AX
        mov [DIVIDER],BX
        mov AX,0
        mov BX,0
        mov DX,0
        mov DI,0
```

```
    mov CX,0

    mov DI,DIVIDER
    mov AX,HIGHT
    DIV DI
    mov REMAINDER_HIGHT, DX
    mov HIGHT,AX

    mov AX,0FFFFh
    mov DX,0
    DIV DI
    mov BX,DX
    mov CX,REMAINDER_HIGHT
    mul CX ; В AX-нижний байт, В DX-остаток
    mov DX,0
    cmp CX,0
    je skip_division
start:
    call Module
    loop start

skip_division:
    mov BX,REMAINDER_HIGHT
    call Module
    mov REMAINDER_LOW,DX
    mov TEMP_LOW,AX

    mov AX,LOW
    mov DX,0
    DIV DI
    mov BX,REMAINDER_LOW
    call Module
    add AX,TEMP_LOW
    adc HIGHT,0000
    mov LOW,AX
    mov REMAINDER_LOW,DX

    mov AX,[LOW]
    mov DX,[HIGHT]
    mov CX,[REMAINDER_LOW]

    pop BX
    pop DI
    ret
division_32 ENDP



Module PROC NEAR
    add DX,BX
    cmp DX,DI
    jb finish
    sub DX,DI
    inc AX
finish:
    ret
Module ENDP
```

```
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
    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




    print_available_memory PROC NEAR
        push AX
        push BX
        push DI
        push DX

        mov AH,4ah
        mov BX, 0ffffh
        int 21h
        mov AX,BX
        mov DI, offset available_memory
        add DI,33
        call paragraphs_to_bytes
        mov DX,offset available_memory
        call print

        pop DX
        pop DI
        pop BX
        pop AX
        ret
    print_available_memory ENDP
    CODE ENDS
    END programm_start
```

Название файла: lab3_1.asm

```
    CODE SEGMENT
    ORG 100h
    ASSUME CS:CODE, DS:CODE,CS:CODE,ES:NOTHING

        programm_start: jmp MAIN
        available_memory DB "Amount of available memory:
",0DH,0AH,'$'
        extended_memory DB "Extended memory size:
",0DH,0AH,'$'
        MCB_info DB
"MCB#  ,adress:    ,owner:    ,size:         ,SD/SC:        ",0DH,0AH,'$'


    MAIN:
```

```
    call print_available_memory
    call print_extended_memory
    call free_memory
    call print_all_MCB
    xor AL,AL
    mov AH, 4ch
    int 21h

print PROC NEAR
    push AX
    mov AH,09h
    int 21h
    pop AX
ret
print ENDP


free_memory PROC NEAR
    push AX
    push BX
    push DX

    mov AX,offset programm_end
    mov DX,0
    mov BX,16
    div BX
    cmp DX,0
    je skip_increment
    inc AX

skip_increment:
    mov BX,AX
    mov AH,4ah
    int 21h

    pop DX
    pop BX
    pop AX
    ret

free_memory ENDP



CLEAR_MCB_INFO PROC NEAR; SI - начало строки
    push SI
    push AX
    push BX

    mov BX,0
clear_loop:
    mov AL,[SI]
    cmp AL,36
    je clear_loop_end
    cmp AL,58
    je clear_active_mode
    cmp AL,35
```

```
    je clear_active_mode
    cmp AL,44
    je clear_passive_mode
    cmp AL, 0dh
    je clear_passive_mode
    jmp clear_continue
clear_active_mode:
    mov BX,1
    inc SI
    jmp clear_loop

clear_passive_mode:
    mov BX,0
    inc SI
    jmp clear_loop

clear_continue:
    cmp BX,1
    je change
    inc SI
    jmp clear_loop

change:
    mov AL,32
    mov [SI],AL
    inc SI
    jmp clear_loop

clear_loop_end:
    pop BX
    pop AX
    pop SI
    ret
CLEAR_MCB_INFO ENDP


print_one_MCB PROC NEAR ;ES-адрес начала блока, CX- счетчик
    push AX
    push SI
    push DI
    push CX
    push DX


    mov AL,CL
    mov SI,offset MCB_info
    add SI,4
    call byte_to_dec
    mov AX,ES
    mov DI,offset MCB_info
    add DI,17
    call wrd_to_hex
    mov AX, ES:[1]
    mov DI,offset MCB_info
    add DI,28
    call wrd_to_hex
    mov AX,ES:[3]
    mov DI,offset MCB_info
```

```
    add DI,43
    call paragraphs_to_bytes
    mov DI,8
    mov CX,8
    mov SI,offset MCB_info
    add SI,51
bytes_loop:
    mov AL,ES:[DI]
    mov [SI],AL
    inc DI
    inc SI
    loop bytes_loop

    mov DX, offset MCB_info
    call print
    mov SI,offset MCB_info
    call CLEAR_MCB_INFO
    pop DX
    pop CX
    pop DI
    pop SI
    pop AX
    ret
print_one_MCB ENDP


print_all_MCB PROC NEAR
    push AX
    push BX
    push CX
    push ES

    mov AH,52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES,AX
    mov CX,1
print_all_loop:
    call print_one_MCB
    cmp BYTE PTR ES:[0], 05ah
    je end_loop
    mov AX,ES
    mov BX, ES:[3]
    add AX, BX
    inc AX
    mov ES,AX
    inc CX
    jmp print_all_loop

end_loop:
    pop ES
    pop CX
    pop BX
    pop AX
    ret
print_all_MCB ENDP

print_extended_memory PROC NEAR
```

```
        push AX
        push BX
        push DX
        push DI

        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 DI,offset extended_memory
        add DI,35
        call kilobytes_to_bytes
        mov DX,offset extended_memory
        call print

        pop DI
        pop DX
        pop BX
        pop AX
        ret
    print_extended_memory ENDP

    ;-------------------------------------
    kilobytes_to_bytes PROC NEAR; AX - количество килобайт, DI - адрес
последнего символа результата
        push AX
        push BX
        push DX
        push CX
        push DI

        mov DX,0

        mov BX,1024
        mul BX
        mov BX,10
    kilobytes_cycle:
        cmp DX,0
        jne check_passed
        cmp AX,10
        ja check_passed
        add AX,48
        mov [DI],AL
        jmp kilobytes_end

    check_passed:
        call DIVISION_32
        add CX,48
        mov [DI],CL
        dec DI
       jmp kilobytes_cycle

    kilobytes_end:
```

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

    paragraphs_to_bytes PROC NEAR ; AX-число параграфов, DI-арес
последнего символа,куда надо записать результат
        push BX
        push DX
        push AX
        push DI

        mov DX,0
        mov BX, 16
        mul BX
        mov BX,10
    paragraphs_start:
        cmp DX,0
        jne paragraphs_skip_check
        cmp AX,10
        jb paragraphs_end
    paragraphs_skip_check:
        div BX
        add DX,48
        mov [DI],DL
        dec DI
        mov DX,0
        jmp paragraphs_start
    paragraphs_end:
        add AX,48
        mov [DI],AL

        pop DI
        pop AX
        pop DX
        pop BX
        ret
    paragraphs_to_bytes ENDP


    division_32 PROC NEAR ;AX - младший байт, DX - старший байт,BX -
делитель.Результат: AX - младший байт, DX - старший байт, CX - остаток
        push DI
        push BX

        jmp division_begin
        HIGHT DW 0
        LOW  DW  0
        DIVIDER DW 0
        REMAINDER_HIGHT DW 0
        REMAINDER_LOW DW 0
        TEMP_LOW DW 0

    division_begin:
```

```
    mov [HIGHT],DX
    mov [LOW],AX
    mov [DIVIDER],BX
    mov AX,0
    mov BX,0
    mov DX,0
    mov DI,0
    mov CX,0

    mov DI,DIVIDER
    mov AX,HIGHT
    DIV DI
    mov REMAINDER_HIGHT, DX
    mov HIGHT,AX

    mov AX,0FFFFh
    mov DX,0
    DIV DI
    mov BX,DX
    mov CX,REMAINDER_HIGHT
    mul CX ; В AX-нижний байт, В DX-остаток
    mov DX,0
    cmp CX,0
    je skip_division
start:
    call Module
    loop start

skip_division:
    mov BX,REMAINDER_HIGHT
    call Module
    mov REMAINDER_LOW,DX
    mov TEMP_LOW,AX

    mov AX,LOW
    mov DX,0
    DIV DI
    mov BX,REMAINDER_LOW
    call Module
    add AX,TEMP_LOW
    adc HIGHT,0000
    mov LOW,AX
    mov REMAINDER_LOW,DX

    mov AX,[LOW]
    mov DX,[HIGHT]
    mov CX,[REMAINDER_LOW]

    pop BX
    pop DI
    ret
division_32 ENDP



Module PROC NEAR
    add DX,BX
```

```
    cmp DX,DI
    jb finish
    sub DX,DI
    inc AX
finish:
    ret
Module ENDP


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
    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



print_available_memory PROC NEAR
    push AX
    push BX
    push DI
    push DX

    mov AH,4ah
    mov BX, 0ffffh
    int 21h
    mov AX,BX
    mov DI, offset available_memory
    add DI,33
    call paragraphs_to_bytes
    mov DX,offset available_memory
    call print

    pop DX
    pop DI
    pop BX
    pop AX
    ret
print_available_memory ENDP

programm_end:

CODE ENDS
END programm_start
```

Название файла: lab3_2.asm

```
CODE SEGMENT
ORG 100h
ASSUME CS:CODE, DS:CODE,CS:CODE,ES:NOTHING
```

```
        programm_start: jmp MAIN
        avalible_memory DB "Amount of available memory:
",0DH,0AH,'$'
        extended_memory DB "Extended memory size:
",0DH,0AH,'$'
        MCB_info DB
"MCB#  ,adress:    ,owner:    ,size:         ,SD/SC:        ",0DH,0AH,'$'
        memory_request_error DB "Memory allocation error.",0DH,0AH,'$'


    MAIN:

        call print_avalible_memory
        call print_extended_memory
        call free_memory
        call request_memory
        call print_all_MCB
        xor AL,AL
        mov AH, 4ch
        int 21h

    print PROC NEAR
        push AX
        mov AH,09h
        int 21h
        pop AX
    ret
    print ENDP


    free_memory PROC NEAR
        push AX
        push BX
        push DX

        mov AX,offset programm_end
        mov DX,0
        mov BX,16
        div BX
        cmp DX,0
        je skip_increment
        inc AX

    skip_increment:
        mov BX,AX
        mov AH,4ah
        int 21h

        pop DX
        pop BX
        pop AX
        ret

    free_memory ENDP


    request_memory PROC NEAR
```

```
    push AX
    push BX

    mov AH,48h
    mov BX,1000h
    int 21h
    jnc end_request
    mov DX,offset memory_request_error
    call print
end_request:

    pop BX
    pop AX
    ret
request_memory ENDP



CLEAR_MCB_INFO PROC NEAR; SI - начало строки
    push SI
    push AX
    push BX

    mov BX,0
clear_loop:
    mov AL,[SI]
    cmp AL,36
    je clear_loop_end
    cmp AL,58
    je clear_active_mode
    cmp AL,35
    je clear_active_mode
    cmp AL,44
    je clear_passive_mode
    cmp AL, 0dh
    je clear_passive_mode
    jmp clear_continue
clear_active_mode:
    mov BX,1
    inc SI
    jmp clear_loop

clear_passive_mode:
    mov BX,0
    inc SI
    jmp clear_loop

clear_continue:
    cmp BX,1
    je change
    inc SI
    jmp clear_loop

change:
    mov AL,32
    mov [SI],AL
    inc SI
```

```
    jmp clear_loop

clear_loop_end:
    pop BX
    pop AX
    pop SI
    ret
CLEAR_MCB_INFO ENDP


print_one_MCB PROC NEAR ;ES-адрес начала блока, CX- счетчик
    push AX
    push SI
    push DI
    push CX
    push DX


    mov AL,CL
    mov SI,offset MCB_info
    add SI,4
    call byte_to_dec
    mov AX,ES
    mov DI,offset MCB_info
    add DI,17
    call wrd_to_hex
    mov AX, ES:[1]
    mov DI,offset MCB_info
    add DI,28
    call wrd_to_hex
    mov AX,ES:[3]
    mov DI,offset MCB_info
    add DI,43
    call paragraphs_to_bytes
    mov DI,8
    mov CX,8
    mov SI,offset MCB_info
    add SI,51
bytes_loop:
    mov AL,ES:[DI]
    mov [SI],AL
    inc DI
    inc SI
    loop bytes_loop

    mov DX, offset MCB_info
    call print
    mov SI,offset MCB_info
    call CLEAR_MCB_INFO
    pop DX
    pop CX
    pop DI
    pop SI
    pop AX
    ret
print_one_MCB ENDP
```

```
print_all_MCB PROC NEAR
    push AX
    push BX
    push CX
    push ES

    mov AH,52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES,AX
    mov CX,1
print_all_loop:
    call print_one_MCB
    cmp BYTE PTR ES:[0], 05ah
    je end_loop
    mov AX,ES
    mov BX, ES:[3]
    add AX, BX
    inc AX
    mov ES,AX
    inc CX
    jmp print_all_loop

end_loop:
    pop ES
    pop CX
    pop BX
    pop AX
    ret
print_all_MCB ENDP

print_extended_memory PROC NEAR
    push AX
    push BX
    push DX
    push DI

    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 DI,offset extended_memory
    add DI,35
    call kilobytes_to_bytes
    mov DX,offset extended_memory
    call print

    pop DI
    pop DX
    pop BX
    pop AX
    ret
print_extended_memory ENDP
```

```
;-------------------------------------
kilobytes_to_bytes PROC NEAR; AX - количество килобайт, DI - адрес последнего символа результата
    push AX
    push BX
    push DX
    push CX
    push DI

    mov DX,0

    mov BX,1024
    mul BX
    mov BX,10
kilobytes_cycle:
    cmp DX,0
    jne check_passed
    cmp AX,10
    ja check_passed
    add AX,48
    mov [DI],AL
    jmp kilobytes_end

check_passed:
    call DIVISION_32
    add CX,48
    mov [DI],CL
    dec DI
   jmp kilobytes_cycle

kilobytes_end:

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

paragraphs_to_bytes PROC NEAR ; AX-число параграфов, DI-арес последнего символа,куда надо записать результат
    push BX
    push DX
    push AX
    push DI

    mov DX,0
    mov BX, 16
    mul BX
    mov BX,10
paragraphs_start:
    cmp DX,0
    jne paragraphs_skip_check
    cmp AX,10
    jb paragraphs_end
paragraphs_skip_check:
```

```
        div BX
        add DX,48
        mov [DI],DL
        dec DI
        mov DX,0
        jmp paragraphs_start
    paragraphs_end:
        add AX,48
        mov [DI],AL

        pop DI
        pop AX
        pop DX
        pop BX
        ret
    paragraphs_to_bytes ENDP


    DIVISION_32 PROC NEAR ;AX - младший байт, DX - старший байт,BX -
делитель.Результат: AX - младший байт, DX - старший байт, CX - остаток
        push DI
        push BX

        jmp division_begin
        HIGHT DW 0
        LOW  DW  0
        DIVIDER DW 0
        REMAINDER_HIGHT DW 0
        REMAINDER_LOW DW 0
        TEMP_LOW DW 0

    division_begin:

        mov [HIGHT],DX
        mov [LOW],AX
        mov [DIVIDER],BX
        mov AX,0
        mov BX,0
        mov DX,0
        mov DI,0
        mov CX,0

        mov DI,DIVIDER
        mov AX,HIGHT
        DIV DI
        mov REMAINDER_HIGHT, DX
        mov HIGHT,AX

        mov AX,0FFFFh
        mov DX,0
        DIV DI
        mov BX,DX
        mov CX,REMAINDER_HIGHT
        mul CX ; В AX-нижний байт, В DX-остаток
        mov DX,0
        cmp CX,0
        je skip_division
    start:
```

```
    call Module
    loop start

skip_division:
    mov BX,REMAINDER_HIGHT
    call Module
    mov REMAINDER_LOW,DX
    mov TEMP_LOW,AX

    mov AX,LOW
    mov DX,0
    DIV DI
    mov BX,REMAINDER_LOW
    call Module
    add AX,TEMP_LOW
    adc HIGHT,0000
    mov LOW,AX
    mov REMAINDER_LOW,DX

    mov AX,[LOW]
    mov DX,[HIGHT]
    mov CX,[REMAINDER_LOW]

    pop BX
    pop DI
    ret
DIVISION_32 ENDP



Module PROC NEAR
    add DX,BX
    cmp DX,DI
    jb finish
    sub DX,DI
    inc AX
finish:
    ret
Module ENDP


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
    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



;-----------------------------------
print_avalible_memory PROC NEAR
    push AX
    push BX
    push DI
```

```
        push DX

        mov AH,4ah
        mov BX, 0ffffh
        int 21h
        mov AX,BX
        mov DI, offset avalible_memory
        add DI,33
        call paragraphs_to_bytes
        mov DX,offset avalible_memory
        call print

        pop DX
        pop DI
        pop BX
        pop AX
        ret
    print_avalible_memory ENDP

    programm_end:

    CODE ENDS
    END programm_start
```

Название файла: lab3_2.asm

```
    CODE SEGMENT
    ORG 100h
    ASSUME CS:CODE, DS:CODE,CS:CODE,ES:NOTHING

        programm_start: jmp MAIN
        avalible_memory DB "Amount of available memory:
",0DH,0AH,'$'
        extended_memory DB "Extended memory size:
",0DH,0AH,'$'
        MCB_info DB
"MCB#  ,adress:    ,owner:    ,size:         ,SD/SC:        ",0DH,0AH,'$'
        memory_request_error DB "Memory allocation error.",0DH,0AH,'$'


    MAIN:

        call print_avalible_memory
        call print_extended_memory
        call request_memory
        call free_memory
        call print_all_MCB
        xor AL,AL
        mov AH, 4ch
        int 21h

    print PROC NEAR
        push AX
        mov AH,09h
        int 21h
        pop AX
```

```
ret
print ENDP


free_memory PROC NEAR
    push AX
    push BX
    push DX

    mov AX,offset programm_end
    mov DX,0
    mov BX,16
    div BX
    cmp DX,0
    je skip_increment
    inc AX

skip_increment:
    mov BX,AX
    mov AH,4ah
    int 21h

    pop DX
    pop BX
    pop AX
    ret

free_memory ENDP


request_memory PROC NEAR
    push AX
    push BX

    mov AH,48h
    mov BX,1000h
    int 21h
    jnc end_request
    mov DX,offset memory_request_error
    call print
end_request:

    pop BX
    pop AX
    ret
request_memory ENDP




CLEAR_MCB_INFO PROC NEAR; SI - начало строки
    push SI
    push AX
    push BX

    mov BX,0
clear_loop:
    mov AL,[SI]
```

```
    cmp AL,36
    je clear_loop_end
    cmp AL,58
    je clear_active_mode
    cmp AL,35
    je clear_active_mode
    cmp AL,44
    je clear_passive_mode
    cmp AL, 0dh
    je clear_passive_mode
    jmp clear_continue
clear_active_mode:
    mov BX,1
    inc SI
    jmp clear_loop

clear_passive_mode:
    mov BX,0
    inc SI
    jmp clear_loop

clear_continue:
    cmp BX,1
    je change
    inc SI
    jmp clear_loop

change:
    mov AL,32
    mov [SI],AL
    inc SI
    jmp clear_loop

clear_loop_end:
    pop BX
    pop AX
    pop SI
    ret
CLEAR_MCB_INFO ENDP


print_one_MCB PROC NEAR ;ES-адрес начала блока, CX- счетчик
    push AX
    push SI
    push DI
    push CX
    push DX


    mov AL,CL
    mov SI,offset MCB_info
    add SI,4
    call byte_to_dec
    mov AX,ES
    mov DI,offset MCB_info
    add DI,17
    call wrd_to_hex
    mov AX, ES:[1]
```

```
    mov DI,offset MCB_info
    add DI,28
    call wrd_to_hex
    mov AX,ES:[3]
    mov DI,offset MCB_info
    add DI,43
    call paragraphs_to_bytes
    mov DI,8
    mov CX,8
    mov SI,offset MCB_info
    add SI,51
bytes_loop:
    mov AL,ES:[DI]
    mov [SI],AL
    inc DI
    inc SI
    loop bytes_loop

    mov DX, offset MCB_info
    call print
    mov SI,offset MCB_info
    call CLEAR_MCB_INFO
    pop DX
    pop CX
    pop DI
    pop SI
    pop AX
    ret
print_one_MCB ENDP


print_all_MCB PROC NEAR
    push AX
    push BX
    push CX
    push ES

    mov AH,52h
    int 21h
    mov AX, ES:[BX-2]
    mov ES,AX
    mov CX,1
print_all_loop:
    call print_one_MCB
    cmp BYTE PTR ES:[0], 05ah
    je end_loop
    mov AX,ES
    mov BX, ES:[3]
    add AX, BX
    inc AX
    mov ES,AX
    inc CX
    jmp print_all_loop

end_loop:
    pop ES
    pop CX
    pop BX
```

```
        pop AX
        ret
    print_all_MCB ENDP

    print_extended_memory PROC NEAR
        push AX
        push BX
        push DX
        push DI

        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 DI,offset extended_memory
        add DI,35
        call kilobytes_to_bytes
        mov DX,offset extended_memory
        call print

        pop DI
        pop DX
        pop BX
        pop AX
        ret
    print_extended_memory ENDP

    ;-------------------------------------
    kilobytes_to_bytes PROC NEAR; AX - количество килобайт, DI - адрес
последнего символа результата
        push AX
        push BX
        push DX
        push CX
        push DI

        mov DX,0

        mov BX,1024
        mul BX
        mov BX,10
    kilobytes_cycle:
        cmp DX,0
        jne check_passed
        cmp AX,10
        ja check_passed
        add AX,48
        mov [DI],AL
        jmp kilobytes_end

    check_passed:
        call DIVISION_32
        add CX,48
```

```
        mov [DI],CL
        dec DI
       jmp kilobytes_cycle

    kilobytes_end:

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

    paragraphs_to_bytes PROC NEAR ; AX-число параграфов, DI-арес
последнего символа,куда надо записать результат
        push BX
        push DX
        push AX
        push DI

        mov DX,0
        mov BX, 16
        mul BX
        mov BX,10
    paragraphs_start:
        cmp DX,0
        jne paragraphs_skip_check
        cmp AX,10
        jb paragraphs_end
    paragraphs_skip_check:
        div BX
        add DX,48
        mov [DI],DL
        dec DI
        mov DX,0
        jmp paragraphs_start
    paragraphs_end:
        add AX,48
        mov [DI],AL

        pop DI
        pop AX
        pop DX
        pop BX
        ret
    paragraphs_to_bytes ENDP


    DIVISION_32 PROC NEAR ;AX - младший байт, DX - старший байт,BX -
делитель.Результат: AX - младший байт, DX - старший байт, CX - остаток
        push DI
        push BX

        jmp division_begin
        HIGHT DW 0
        LOW  DW  0
        DIVIDER DW 0
```

```
    REMAINDER_HIGHT DW 0
    REMAINDER_LOW DW 0
    TEMP_LOW DW 0

division_begin:

    mov [HIGHT],DX
    mov [LOW],AX
    mov [DIVIDER],BX
    mov AX,0
    mov BX,0
    mov DX,0
    mov DI,0
    mov CX,0

    mov DI,DIVIDER
    mov AX,HIGHT
    DIV DI
    mov REMAINDER_HIGHT, DX
    mov HIGHT,AX

    mov AX,0FFFFh
    mov DX,0
    DIV DI
    mov BX,DX
    mov CX,REMAINDER_HIGHT
    mul CX ; В AX-нижний байт, В DX-остаток
    mov DX,0
    cmp CX,0
    je skip_division
start:
    call Module
    loop start

skip_division:
    mov BX,REMAINDER_HIGHT
    call Module
    mov REMAINDER_LOW,DX
    mov TEMP_LOW,AX

    mov AX,LOW
    mov DX,0
    DIV DI
    mov BX,REMAINDER_LOW
    call Module
    add AX,TEMP_LOW
    adc HIGHT,0000
    mov LOW,AX
    mov REMAINDER_LOW,DX

    mov AX,[LOW]
    mov DX,[HIGHT]
    mov CX,[REMAINDER_LOW]

    pop BX
    pop DI
    ret
DIVISION_32 ENDP
```

```
Module PROC NEAR
    add DX,BX
    cmp DX,DI
    jb finish
    sub DX,DI
    inc AX
finish:
    ret
Module ENDP


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
```

```
    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



print_avalible_memory PROC NEAR
    push AX
    push BX
    push DI
    push DX

    mov AH,4ah
    mov BX, 0ffffh
    int 21h
    mov AX,BX
    mov DI, offset avalible_memory
    add DI,33
    call paragraphs_to_bytes
    mov DX,offset avalible_memory
    call print

    pop DX
    pop DI
    pop BX
    pop AX
    ret
print_avalible_memory ENDP

programm_end:

CODE ENDS
END programm_start
```