

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

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

Студент гр. 9382

Субботин М.О.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

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. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа **.СOM**, 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 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.

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.

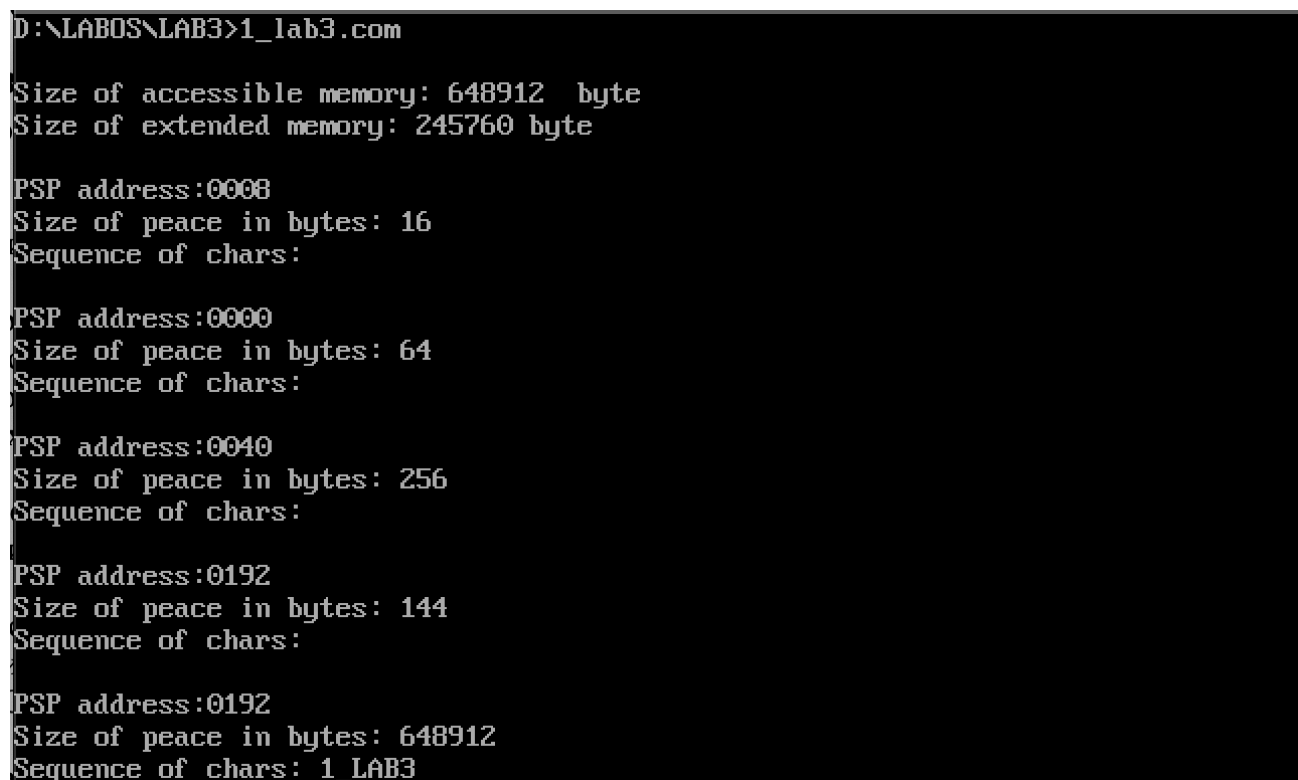
Шаг 3. Измените программу еще раз таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти функцией 48H прерывания 21H. Повторите эксперимент, запустив модифицированную программу. Сравните выходные данные с результатами, полученными на предыдущих шагах. Сохраните результаты,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

Ход выполнения:

Результаты выполнения программы на первом шаге:



```
D:\LABOS\LAB3>1_lab3.com

Size of accessible memory: 648912 byte
Size of extended memory: 245760 byte

PSP address:0008
Size of piece in bytes: 16
Sequence of chars:

PSP address:0000
Size of piece in bytes: 64
Sequence of chars:

PSP address:0040
Size of piece in bytes: 256
Sequence of chars:

PSP address:0192
Size of piece in bytes: 144
Sequence of chars:

PSP address:0192
Size of piece in bytes: 648912
Sequence of chars: 1_LAB3
```

Рисунок 1. Результат первой программы.

Затем программа была модифицирована и теперь она умеет очищать неиспользуемую память.

```

D:\LABOS\LAB3>lab3_2.com

Size of accessible memory: 648912  byte
Size of extended memory: 245760 byte
PSP address: 0008
Size of peace in bytes: 16
Sequence of chars:
PSP address: 0000
Size of peace in bytes: 64
Sequence of chars:
PSP address: 0040
Size of peace in bytes: 256
Sequence of chars:
PSP address: 0192
Size of peace in bytes: 144
Sequence of chars:
PSP address: 0192
Size of peace in bytes: 880
Sequence of chars: LAB3_2
PSP address: 0000
Size of peace in bytes: 648016
Sequence of chars: &:E@w@.
D:\LABOS\LAB3>

```

Рисунок 2. Результат выполнения второй программы.

Следующая программа после освобождения памяти выделяет блок памяти 64кб.

```

Size of accessible memory: 896      byte
Size of extended memory: 245760 byte
PSP address: 0008
Size of peace in bytes: 16
Sequence of chars:
PSP address: 0000
Size of peace in bytes: 64
Sequence of chars:
PSP address: 0040
Size of peace in bytes: 256
Sequence of chars:
PSP address: 0192
Size of peace in bytes: 144
Sequence of chars:
PSP address: 0192
Size of peace in bytes: 896
Sequence of chars: LAB3_3
PSP address: 0192
Size of peace in bytes: 65536
Sequence of chars: LAB3_3
PSP address: 0000
Size of peace in bytes: 582448
Sequence of chars: ght (C)

```

Рисунок 3. Результат выполнения третьей программы.

По результатам третьей программы можно заметить, что появился еще один блок размером 64кб.

В четвертой программе, после выделения памяти происходит проверка, а также после ее очистки. Проверка заключается в вызове функции 4ah прерывания 21h. Выделение памяти проходит нормально, а после очистки выбрасывается ошибка, т.к. не осталось неиспользуемой памяти.

```
D:\LABOS\LAB3>lab3_4.com
Function 4ah of dispatcher 21h correct
Function 4ah of dispatcher 21h failed, error code: 0900

Size of accessible memory: 648912 byte
Size of extended memory: 245760 byte
PSP address: 0008
Size of piece in bytes: 16
Sequence of chars:
PSP address: 0000
Size of piece in bytes: 64
Sequence of chars:
PSP address: 0040
Size of piece in bytes: 256
Sequence of chars:
PSP address: 0192
Size of piece in bytes: 144
Sequence of chars:
PSP address: 0192
Size of piece in bytes: 648912
Sequence of chars: LAB3_4
```

Рисунок 4. Результат выполнения четвертой программы.

Ответы на контрольные вопросы.

1. Что означает “доступный объем памяти”?

Это такой объем оперативной памяти, который занимает и использует программа.

2. Где МСВ блок Вашей программы в списке?

В 1-ой программе МСВ блок программы находится в конце списка.

В 2-ой программе МСВ блок программы находится на предпоследнем месте, т.к. на последнем месте находится блок с высвобожденной неиспользуемой памятью.

В 3-ей программе МСВ блок программы смещается еще на одну позицию вверх по сравнению со второй программой, т.к. сначала освобождается неиспользуемая память и после от нее выделяется часть под программу.

В 4-ой программе МСВ блок программы находится последним.

3. Какой размер памяти занимает программа в каждом случае?

В 1-ой программе всю свободную память (648912 байт)

В 2-ой программе необходимый объем в 880 байт.

В 3-ей программе необходимый объем и дополнительно выделенные 64Кб, т.е. $896 + 64 \cdot 1024$ байт

В 4-й программе всю свободную память (648912 байт)

Выводы.

Были изучены способы организации памяти в ОС, управления динамическими разделами. Также были исследованы структуры данных и работа функций управления памятью ядра операционной системы.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММ

Lab3_1.asm:

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
; DATA
ACCESSIBLE_MEMORY DB 0DH,0AH,'SIZE OF ACCESSIBLE MEMORY:
$'
EXTENDED_MEMORY DB 0DH,0AH,'SIZE OF EXTENDED MEMORY:
$'
STR_BYTE DB ' BYTE $'
ADDRESS_PSP DB 0DH,0AH,0DH,0AH,'PSP ADDRESS:      ',0DH,0AH,$'
STR_SIZE DB 'SIZE OF PEACE IN BYTES:      ',0DH,0AH,$'
SEQUENCE DB 'SEQUENCE OF CHARS: $'

; PROCEDURES
;-----
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 ШЕСТ. 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, 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 10С/С, SI - АДРЕС ПОЛЯ МЛАДШЕЙ ЦИФРЫ
    PUSH CX
    PUSH DX
    XOR AH,AH
    XOR DX,DX
    MOV CX,10
LOOP_BD:
    DIV CX
    OR DL,30H
    MOV [SI],DL
    DEC SI
    XOR DX,DX
    CMP AX,10
    JAE LOOP_BD
    CMP AL,00H
    JE END_L
    OR AL,30H
    MOV [SI],AL
END_L:
    POP DX
    POP CX
    RET
BYTE_TO_DEC ENDP

```



```

;-----
WRITE_STRING PROC NEAR
    PUSH AX
    MOV AH,09H
    INT 21H
    POP AX
    RET
WRITE_STRING ENDP

;-----

WRITE_MEMORY_SIZE PROC
;SAVE REGISTERS
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI

        MOV BX,10H
        MUL BX
        MOV BX,0AH
        XOR CX,CX

DIVISION:
    DIV BX
    PUSH DX
    INC CX
    XOR DX,DX
    CMP AX,0H
    JNZ DIVISION

WRITE_SYMBOL:
    POP DX
    OR DL,30H
    MOV [SI], DL
    INC SI
    LOOP WRITE_SYMBOL

    POP SI
    POP DX
    POP CX
    POP BX
    POP AX

```

```
RET
WRITE_MEMORY_SIZE ENDP
```

```
;-----
```

```
MCB PROC
;SAVE REGISTERS
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH SI
```

```
MOV AH,52H
INT 21H
MOV AX,ES:[BX-2]
MOV ES,AX
XOR CX,CX
```

```
INC CX
MCB_PARAGRAPH:
PUSH CX
```

```
XOR AH,AH
MOV AL,ES:[0]
PUSH AX
MOV AX,ES:[1]
LEA DI, ADDRESS_PSP
ADD DI, 19
CALL WRD_TO_HEX
LEA DX, ADDRESS_PSP
CALL WRITE_STRING
```

```
MOV AX,ES:[3]
LEA SI,STR_SIZE
ADD SI, 24
CALL WRITE_MEMORY_SIZE
LEA DX,STR_SIZE
CALL WRITE_STRING
```

```
XOR DX, DX
LEA DX , SEQUENCE
CALL WRITE_STRING
```

```

MOV CX,8
XOR DI,DI

WRITE_CHAR:
MOV DL,ES:[DI+8]
MOV AH,02H
INT 21H
INC DI
LOOP WRITE_CHAR

MOV AX,ES:[3]
MOV BX,ES
ADD BX,AX
INC BX
MOV ES,BX
POP AX
POP CX
INC CX
CMP AL,5AH
JE THE_END
CMP AL,4DH
JNE THE_END
JMP MCB_PARAGRAPH

THE_END:
POP SI
POP DX
POP CX
POP BX
POP AX
RET
MCB ENDP

; CODE
BEGIN:
;ACCESSIBLE MEMORY
MOV AH,4AH
MOV BX,0FFFFH
INT 21H
MOV AX,BX
LEA SI, ACCESSIBLE_MEMORY
ADD SI, 29 ;OFFSET FOR A NUMBER
CALL WRITE_MEMORY_SIZE
LEA DX, ACCESSIBLE_MEMORY

```

CALL WRITE_STRING

LEA DX,STR_BYTE
CALL WRITE_STRING

; EXTENDED MEMORY
MOV AL,30H
OUT 70H,AL
IN AL,71H
MOV BL,AL
MOV AL,31H
OUT 70H,AL
IN AL,71H

MOV BH,AL
MOV AX,BX
LEA SI,EXTENDED_MEMORY
ADD SI, 27
CALL WRITE_MEMORY_SIZE
LEA DX,EXTENDED_MEMORY
CALL WRITE_STRING

LEA DX,STR_BYTE
CALL WRITE_STRING

;MCB STAFF
CALL MCB

XOR AL,AL
MOV AH,4CH
INT 21H
TESTPC ENDS
END START

Lab3_2.asm:

TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
; DATA
ACCESSIBLE_MEMORY DB 0DH,0AH,'SIZE OF ACCESSIBLE MEMORY:
\$'

```

EXTENDED_MEMORY DB 0DH,0AH,'SIZE OF EXTENDED MEMORY:
$'
STR_BYTE DB ' BYTE $'
ADDRESS_PSP DB 0DH,0AH,'PSP ADDRESS:      ',0DH,0AH,'$'
STR_SIZE DB 'SIZE OF PEACE IN BYTES:      ',0DH,0AH,'$'
SEQUENCE DB 'SEQUENCE OF CHARS: $'

; PROCEDURES
;-----
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 ШЕСТ. 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, 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 10С/С, SI - АДРЕС ПОЛЯ МЛАДШЕЙ ЦИФРЫ
PUSH CX
PUSH DX
XOR AH,AH
XOR DX,DX
MOV CX,10
LOOP_BD:
DIV CX
OR DL,30H
MOV [SI],DL
DEC SI
XOR DX,DX
CMP AX,10
JAE LOOP_BD
CMP AL,00H
JE END_L
OR AL,30H
MOV [SI],AL
END_L:
POP DX
POP CX
RET
BYTE_TO_DEC ENDP
;-----
WRITE_STRING PROC NEAR
PUSH AX
MOV AH,09H
INT 21H
POP AX
RET
WRITE_STRING ENDP
;-----

WRITE_MEMORY_SIZE PROC
;SAVE REGISTERS

```

```
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH SI
```

```
    MOV BX,10H
    MUL BX
    MOV BX,0AH
    XOR CX,CX
```

DIVISION:

```
    DIV BX
    PUSH DX
    INC CX
    XOR DX,DX
    CMP AX,0H
    JNZ DIVISION
```

WRITE_SYMBOL:

```
    POP DX
    OR DL,30H
    MOV [SI], DL
    INC SI
    LOOP WRITE_SYMBOL
```

```
POP SI
POP DX
POP CX
POP BX
POP AX
RET
```

WRITE_MEMORY_SIZE ENDP

;-----

MCB PROC

;SAVE REGISTERS

```
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
```

```

MOV AH,52H
INT 21H
MOV AX,ES:[BX-2]
MOV ES,AX
XOR CX,CX

    INC CX
MCB_PARAGRAPH:
    PUSH CX

        XOR AH,AH
        MOV AL,ES:[0]
        PUSH AX
        MOV AX,ES:[1]
        LEA DI, ADDRESS_PSP
        ADD DI, 19
        CALL WRD_TO_HEX
        LEA DX, ADDRESS_PSP
        CALL WRITE_STRING

        MOV AX,ES:[3]
        LEA SI,STR_SIZE
        ADD SI, 24
        CALL WRITE_MEMORY_SIZE
        LEA DX,STR_SIZE
        CALL WRITE_STRING

        XOR DX, DX
        LEA DX , SEQUENCE
        CALL WRITE_STRING

        MOV CX,8
        XOR DI,DI

WRITE_CHAR:
    MOV DL,ES:[DI+8]
    MOV AH,02H
    INT 21H
    INC DI
    LOOP WRITE_CHAR

    MOV AX,ES:[3]
    MOV BX,ES
    ADD BX,AX

```



```
INC BX
MOV ES,BX
POP AX
POP CX
INC CX
CMP AL,5AH
JE THE_END
CMP AL,4DH
JNE THE_END
JMP MCB_PARAGRAPH
```

```
THE_END:
POP SI
POP DX
POP CX
POP BX
POP AX
RET
MCB ENDP
```

```
FREE_UNUSED_MEMORY PROC
```

```
PUSH AX
PUSH BX
PUSH CX
PUSH DX
```

```
LEA AX, VERY_END
MOV BX,10H
XOR DX,DX
DIV BX
INC AX
MOV BX,AX
ADD BX,5H
MOV AL,0
MOV AH,4AH
INT 21H
```

```
POP DX
POP CX
POP BX
POP AX
```

```
RET
FREE_UNUSED_MEMORY ENDP
```

```

; CODE
BEGIN:
;ACCESSIBLE MEMORY
    MOV AH,4AH
    MOV BX,0FFFFH
    INT 21H
MOV AX,BX
LEA SI, ACCESSIBLE_MEMORY
ADD SI, 29 ;OFFSET FOR A NUMBER
CALL WRITE_MEMORY_SIZE
LEA DX, ACCESSIBLE_MEMORY
CALL WRITE_STRING

LEA DX,STR_BYTE
CALL WRITE_STRING

CALL FREE_UNUSED_MEMORY

; EXTENDED MEMORY
MOV AL,30H
OUT 70H,AL
IN AL,71H
MOV BL,AL
MOV AL,31H
OUT 70H,AL
IN AL,71H

    MOV BH,AL
    MOV AX,BX
    LEA SI,EXTENDED_MEMORY
    ADD SI, 27
    CALL WRITE_MEMORY_SIZE
    LEA DX,EXTENDED_MEMORY
    CALL WRITE_STRING

    LEA DX,STR_BYTE
    CALL WRITE_STRING

;MCB STAFF
CALL MCB

XOR AL,AL

```

```

MOV AH,4CH
INT 21H
VERY_END:
TESTPC ENDS
END START

```

Lab3_3.asm:

```

TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
; DATA
ACCESSIBLE_MEMORY DB 0DH,0AH,'SIZE OF ACCESSIBLE MEMORY:
$'
EXTENDED_MEMORY DB 0DH,0AH,'SIZE OF EXTENDED MEMORY:
$'
STR_BYTE DB ' BYTE $'
ADDRESS_PSP DB 0DH,0AH,'PSP ADDRESS:      ',0DH,0AH,'$'
STR_SIZE DB 'SIZE OF PEACE IN BYTES:      ',0DH,0AH,'$'
SEQUENCE DB 'SEQUENCE OF CHARS: $'

; PROCEDURES
;-----
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 ШЕСТ. 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, 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 10С/С, SI - АДРЕС ПОЛЯ МЛАДШЕЙ ЦИФРЫ
    PUSH CX
    PUSH DX
    XOR AH,AH
    XOR DX,DX
    MOV CX,10
LOOP_BD:
    DIV CX
    OR DL,30H
    MOV [SI],DL
    DEC SI
    XOR DX,DX
    CMP AX,10
    JAE LOOP_BD
    CMP AL,00H
    JE END_L
    OR AL,30H
    MOV [SI],AL
END_L:
    POP DX
    POP CX

```

```

    RET
BYTE_TO_DEC ENDP
;-----
WRITE_STRING PROC NEAR
    PUSH AX
    MOV AH,09H
    INT 21H
    POP AX
    RET
WRITE_STRING ENDP

;-----

WRITE_MEMORY_SIZE PROC
;SAVE REGISTERS
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI

    MOV BX,10H
    MUL BX
    MOV BX,0AH
    XOR CX,CX

DIVISION:
    DIV BX
    PUSH DX
    INC CX
    XOR DX,DX
    CMP AX,0H
    JNZ DIVISION

WRITE_SYMBOL:
    POP DX
    OR DL,30H
    MOV [SI], DL
    INC SI
    LOOP WRITE_SYMBOL

    POP SI
    POP DX
    POP CX

```

```

    POP BX
    POP AX
    RET
WRITE_MEMORY_SIZE ENDP

```

```

;-----

```

```

MCB PROC
;SAVE REGISTERS
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI

```

```

    MOV AH,52H
    INT 21H
    MOV AX,ES:[BX-2]
    MOV ES,AX
    XOR CX,CX

```

```

        INC CX
MCB_PARAGRAPH:
    PUSH CX

```

```

    XOR AH,AH
    MOV AL,ES:[0]
    PUSH AX
    MOV AX,ES:[1]
    LEA DI, ADDRESS_PSP
    ADD DI, 19
    CALL WRD_TO_HEX
    LEA DX, ADDRESS_PSP
    CALL WRITE_STRING

```

```

    MOV AX,ES:[3]
    LEA SI,STR_SIZE
    ADD SI, 24
    CALL WRITE_MEMORY_SIZE
    LEA DX,STR_SIZE
    CALL WRITE_STRING

```

```

    XOR DX, DX
    LEA DX , SEQUENCE

```

```

CALL WRITE_STRING

MOV CX,8
XOR DI,DI

WRITE_CHAR:
    MOV DL,ES:[DI+8]
    MOV AH,02H
    INT 21H
    INC DI
    LOOP WRITE_CHAR

    MOV AX,ES:[3]
    MOV BX,ES
    ADD BX,AX
    INC BX
    MOV ES,BX
    POP AX
    POP CX
    INC CX
    CMP AL,5AH
    JE THE_END
    CMP AL,4DH
    JNE THE_END
    JMP MCB_PARAGRAPH

THE_END:
    POP SI
    POP DX
    POP CX
    POP BX
    POP AX
    RET
MCB ENDP

FREE_UNUSED_MEMORY PROC
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX

    LEA AX, VERY_END
    MOV BX,10H
    XOR DX,DX

```

```

    DIV BX
    INC AX
    MOV BX,AX
    ADD BX,5H
    MOV AL,0
    MOV AH,4AH
    INT 21H

    POP DX
    POP CX
    POP BX
    POP AX

    RET
FREE_UNUSED_MEMORY ENDP

GET_EXTRA_MEMORY PROC NEAR
    PUSH AX
    PUSH BX
    PUSH DX

    MOV BX,1000H
    MOV AH,48H
    INT 21H

    POP DX
    POP BX
    POP AX
    RET
GET_EXTRA_MEMORY ENDP

; CODE
BEGIN:
    CALL FREE_UNUSED_MEMORY
    CALL GET_EXTRA_MEMORY

;ACCESSIBLE MEMORY
    MOV AH,4AH
    MOV BX,0FFFFH
    INT 21H
    MOV AX,BX
    LEA SI, ACCESSIBLE_MEMORY
    ADD SI, 29 ;OFFSET FOR A NUMBER

```



```
CALL WRITE_MEMORY_SIZE
LEA DX, ACCESSIBLE_MEMORY
CALL WRITE_STRING
```

```
LEA DX,STR_BYTE
CALL WRITE_STRING
```

```
; EXTENDED MEMORY
MOV AL,30H
OUT 70H,AL
IN AL,71H
MOV BL,AL
MOV AL,31H
OUT 70H,AL
IN AL,71H
```

```
    MOV BH,AL
    MOV AX,BX
    LEA SI,EXTENDED_MEMORY
    ADD SI, 27
    CALL WRITE_MEMORY_SIZE
    LEA DX,EXTENDED_MEMORY
    CALL WRITE_STRING
```

```
    LEA DX,STR_BYTE
    CALL WRITE_STRING
```

```
;MCB STAFF
CALL MCB
```

```
XOR AL,AL
MOV AH,4CH
INT 21H
VERY_END:
TESTPC ENDS
END START
```

Lab3_4.asm:

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

```

; Data
ACCESSIBLE_MEMORY db 0DH,0AH,'Size of accessible memory:    '$'
EXTENDED_MEMORY db 0DH,0AH,'Size of extended memory:    '$'
STR_BYTE db ' byte '$'
ADDRESS_PSP db 0DH,0AH,'PSP address:    ',0DH,0AH,'$'
STR_SIZE db 'Size of peace in bytes:    ',0DH,0AH,'$'
SEQUENCE db 'Sequence of chars: '$'
CHECK_MEMORY_BAD db 'Function 4ah of dispatcher 21h failed, error code:    ',
0AH, 0DH, '$'
CHECK_MEMORY_GOOD db 'Function 4ah of dispatcher 21h correct', 0DH, 0AH,
'$'

```

```

; Procedures

```

```

;-----

```

```

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 шест. 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, 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 10с/с, SI - адрес поля младшей цифры
    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
    ret
BYTE_TO_DEC ENDP
;-----
WRITE_STRING PROC near
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
WRITE_STRING ENDP

```

;-----

WRITE_MEMORY_SIZE PROC

;save registers

push ax

push bx

push cx

push dx

push si

mov bx,10h

mul bx

mov bx,0ah

xor cx,cx

division:

div bx

push dx

inc cx

xor dx,dx

cmp ax,0h

jnz division

write_symbol:

pop dx

or dl,30h

mov [si], dl

inc si

loop write_symbol

pop si

pop dx

pop cx

pop bx

pop ax

ret

WRITE_MEMORY_SIZE ENDP

;-----

MCB PROC

;save registers

push ax

```
push bx
push cx
push dx
push si
```

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

```
inc cx
MCB_PARAGRAPH:
push cx
```

```
xor ah,ah
mov al,es:[0]
push ax
mov ax,es:[1]
lea di, ADDRESS_PSP
add di, 19
call WRD_TO_HEX
lea dx, ADDRESS_PSP
call WRITE_STRING
```

```
mov ax,es:[3]
lea si,STR_SIZE
add si, 24
call WRITE_MEMORY_SIZE
lea dx,STR_SIZE
call WRITE_STRING
```

```
xor dx, dx
lea dx , SEQUENCE
call WRITE_STRING
```

```
mov cx,8
xor di,di
```

```
write_char:
mov dl,es:[di+8]
mov ah,02h
int 21h
inc di
```

```

        loop write_char

        mov ax,es:[3]
        mov bx,es
        add bx,ax
        inc bx
        mov es,bx
        pop ax
        pop cx
        inc cx
        cmp al,5Ah
        je the_end
        cmp al,4Dh
        jne the_end
        jmp MCB_PARAGRAPH

the_end:
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        ret
MCB ENDP

FREE_UNUSED_MEMORY PROC
        push ax
        push bx
        push cx
        push dx

        lea ax, very_end
        mov bx,10h
        xor dx,dx
        div bx
        inc ax
        mov bx,ax
        add bx,5h
        mov al,0
        mov ah,4AH
        int 21h

        pop dx
        pop cx

```

```

    pop bx
    pop ax

    ret
FREE_UNUSED_MEMORY ENDP

CHECK_MEMORY PROC near

    jae wrong
    mov dx, offset CHECK_MEMORY_GOOD
    mov ah, 09h
    int 21h
    jmp end_CHECK_MEMORY

wrong:

    mov di, offset CHECK_MEMORY_GOOD - 4
    call WRD_TO_HEX

    mov dx, offset CHECK_MEMORY_BAD
    mov ah, 09h
    int 21h

end_CHECK_MEMORY:

    ret
CHECK_MEMORY ENDP

GET_EXTRA_MEMORY PROC near
    push ax
    push bx
    push dx

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

    pop dx
    pop bx
    pop ax
    ret
GET_EXTRA_MEMORY ENDP

```

```

; Code
BEGIN:
    call GET_EXTRA_MEMORY
    call CHECK_MEMORY
    call FREE_UNUSED_MEMORY
    call CHECK_MEMORY

;accessible memory
    mov ah,4ah
    mov bx,0ffffh
    int 21h
    mov ax,bx
    lea si, ACCESSIBLE_MEMORY
    add si, 29 ;offset for a number
    call WRITE_MEMORY_SIZE
    lea dx, ACCESSIBLE_MEMORY
    call WRITE_STRING

    lea dx,STR_BYTE
    call WRITE_STRING

; extended memory
    mov AL,30h
    out 70h,AL
    in AL,71h
    mov BL,AL
    mov AL,31h
    out 70h,AL
    in AL,71h

    mov bh,al
    mov ax,bx
    lea si,EXTENDED_MEMORY
    add si, 27
    call WRITE_MEMORY_SIZE
    lea dx,EXTENDED_MEMORY
    call WRITE_STRING

    lea dx,STR_BYTE
    call WRITE_STRING

;MCB staff
    call MCB

```



```
xor AL,AL
mov AH,4Ch
int 21H
very_end:
TESTPC ENDS
END START
```