

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. 9383

Камзолов Н.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

Цель работы.

Исследование организации управления памятью. Рассмотрение нестраничной памяти и способов управления динамическими разделами. Исследование структур данных и работа функций управления памятью ядра операционной системы.

Задание.

Шаг 1. Написать и отладить программный модуль .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 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.

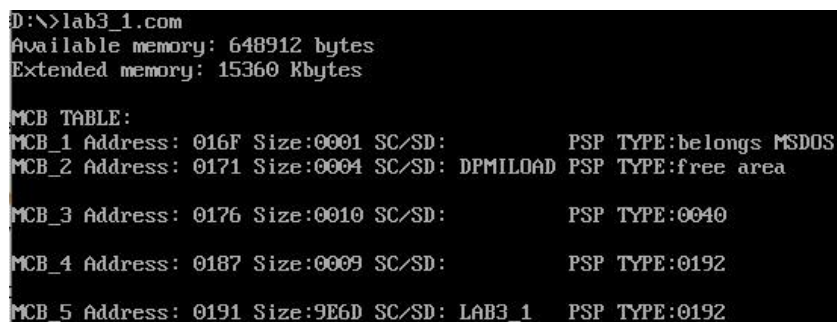
Шаг 4. Измените первоначальный вариант программы, запросив 64Кб памяти функцией 48H прерывания 21H до освобождения памяти. Обязательно обрабатывайте завершение функций ядра, проверяя флаг CF. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

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

Результаты исследования проблем.

Шаг 1. Был написан и отлажен программный модуль типа .COM, который выбирает и распечатывает следующую информацию:

1. Количество доступной памяти в десятичном виде(в байтах).
2. Размер расширенной памяти в десятичном виде(в Кбайтах).
3. Выводит цепочку блоков управления памятью.



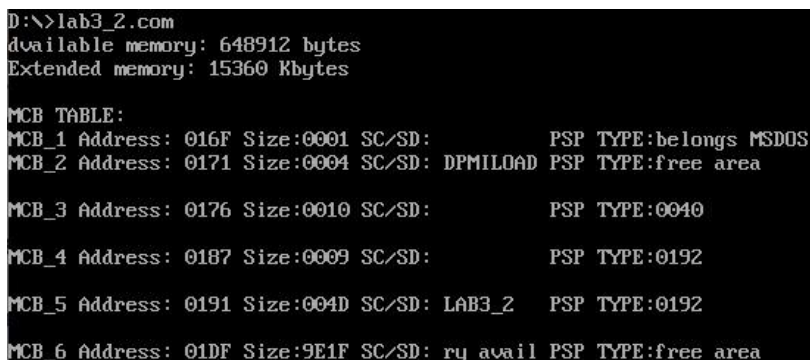
```
D:\>lab3_1.com
Available memory: 648912 bytes
Extended memory: 15360 Kbytes

MCB TABLE:
MCB_1 Address: 016F Size:0001 SC/SD:      PSP TYPE:belongs MSDOS
MCB_2 Address: 0171 Size:0004 SC/SD: DPMILOAD PSP TYPE:free area

MCB_3 Address: 0176 Size:0010 SC/SD:      PSP TYPE:0040
MCB_4 Address: 0187 Size:0009 SC/SD:      PSP TYPE:0192
MCB_5 Address: 0191 Size:9E6D SC/SD: LAB3_1  PSP TYPE:0192
```

Рисунок 1 – Вывод .COM модуля по итогам выполнения первого шага.

Шаг 2. Программа была изменена таким образом, чтобы она освобождала память, которую она не занимает.



```
D:\>lab3_2.com
Available memory: 648912 bytes
Extended memory: 15360 Kbytes

MCB TABLE:
MCB_1 Address: 016F Size:0001 SC/SD:      PSP TYPE:belongs MSDOS
MCB_2 Address: 0171 Size:0004 SC/SD: DPMILOAD PSP TYPE:free area

MCB_3 Address: 0176 Size:0010 SC/SD:      PSP TYPE:0040
MCB_4 Address: 0187 Size:0009 SC/SD:      PSP TYPE:0192
MCB_5 Address: 0191 Size:004D SC/SD: LAB3_2  PSP TYPE:0192
MCB_6 Address: 01DF Size:9E1F SC/SD: ry avail PSP TYPE:free area
```

Рисунок 2 – Вывод .COM модуля по итогам выполнения второго шага.

Шаг 3. Программа была изменена таким образом, чтобы после освобождения памяти, она запрашивала 64Кб памяти.

```
D:\>lab3_3.com
Available memory: 648912 bytes
Extended memory: 15360 Kbytes

MCB TABLE:
MCB_1 Address: 016F Size:0001 SC/SD:      PSP TYPE:belongs MSDOS
MCB_2 Address: 0171 Size:0004 SC/SD: DPMILOAD PSP TYPE:free area

MCB_3 Address: 0176 Size:0010 SC/SD:      PSP TYPE:0040
MCB_4 Address: 0187 Size:0009 SC/SD:      PSP TYPE:0192
MCB_5 Address: 0191 Size:004F SC/SD: LAB3_3  PSP TYPE:0192
MCB_6 Address: 01E1 Size:1000 SC/SD: LAB3_3  PSP TYPE:0192
MCB_7 Address: 11E2 Size:8E1C SC/SD:  2A    PSP TYPE:free area
```

Рисунок 3 – Вывод .COM модуля по итогам выполнения третьего шага.

Шаг 4. Программа была изменена таким образом, чтобы она запрашивала 64Кб памяти до освобождения памяти. Написан обработчик завершения функции ядра, проверяющий флаг CF.

```
D:\>lab3_4.com
Available memory: 648912 bytes
Extended memory: 15360 Kbytes
Cannot allocate memory
MCB TABLE:
MCB_1 Address: 016F Size:0001 SC/SD:      PSP TYPE:belongs MSDOS
MCB_2 Address: 0171 Size:0004 SC/SD: DPMILOAD PSP TYPE:free area

MCB_3 Address: 0176 Size:0010 SC/SD:      PSP TYPE:0040
MCB_4 Address: 0187 Size:0009 SC/SD:      PSP TYPE:0192
MCB_5 Address: 0191 Size:0053 SC/SD: LAB3_4  PSP TYPE:0192
MCB_6 Address: 01E5 Size:9E19 SC/SD: ine not PSP TYPE:free area
```

Рисунок 4 – Вывод .COM модуля по итогам выполнения четвертого шага.

По итогам выполнения работы можно ответить на контрольные вопросы:

1. Что означает «доступный объем памяти»?

Ответ: область оперативной памяти, которая выделяется программе для использования управляющей программой.

2. Где МСВ блок Вашей программы в списке.

Ответ: чтобы ответить на этот вопросы обратимся к скриншотам программы, изображенным на рисунках 1-4(на них изображены те самые МСВ блоки).

Таким образом:

- На первом шаге МСВ блок нашей программы 5-ый в списке.
- На втором шаге МСВ блок нашей программы 5-ый в списке.
- На третьем шаге МСВ блок нашей программы 5-ый и 6-ой в списке, так как мы вручную запросили еще один блок памяти.
- На четвертом шаге МСВ блок нашей программы 5-ый в списке.

3. Какой размер памяти занимает программа в каждом случае?

- На первом шаге программа занимает всю доступную память.
- На втором шаге программа занимает 4D(77) параграфов или 1232 байта.
- На третьем шаге программа занимает 1232 байта + выделенный блок размером 64Кб.
- На четвертом шаге программа занимает 1232 байта, так как программа не смогла выделить дополнительный блок размером 64Кб.

Выводы.

Исследована организация управления памятью. Рассмотрено устройство нестраничной памяти и способов управления динамическими разделами. Исследованы структуры данных и работа функций управления памятью ядра операционной системы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

lab3_1.asm:

```
TESTPC SEGMENT

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

ORG 100H


START: JMP BEGIN


MCB_TABLE_DECLARATION DB 'MCB TABLE: ', 0DH, 0AH, '$'
AVAILABLE_MEMORY_STRING DB 'AVAILABLE MEMORY: $'
EXTENDED_MEMORY_STRING DB 'EXTENDED MEMORY: $'
BYTES_STRING DB ' BYTES$'
KILOBYTES_STRING DB ' KBYTES$'
MCB_SIZE_STRING DB 'SIZE:      $'
FREE_AREA DB 'FREE AREA                $'
OS_XMS_UMB DB 'BELONGS TO OS XMS UMB      $'
EXCLUDED_HIGH DB 'EXCLUDED HIGH MEMORY    $'
BELONGS_MSDOS DB 'BELONGS MSDOS           $'
BUSY_386MAX_UMB DB 'BUSY WITH BLOCK 386MAX UMB$'
BLOCKED_386MAX DB 'BLOCKED 386MAX          $'
BELONGS_386MAX DB 'BELONGS 386MAX          $'
ADDRESS_STRING DB 'ADDRESS:      $'
MCB_SC_SD_STRING DB 'SC/SD: $'
TAB DB ' $'
STRING_FOR_PSP_TYPE DB '                                $'
PSP_TYPE_DECLARATION DB 'PSP TYPE:$'


MCB_NUMBER DB 'MCB_ $'


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 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_L
        OR AL, 30H
        MOV [SI], AL

END_L:
        POP DX
        POP CX
        POP AX
        RET

BYTE_TO_DEC ENDP

PRINT_NEW_LINE PROC NEAR
        PUSH AX
        PUSH DX

        MOV DL, 0DH
        MOV AH, 02H
        INT 21H

        MOV DL, 0AH
        MOV AH, 02H
        INT 21H

        POP DX

```

```

    POP AX
    RET
PRINT_NEW_LINE ENDP

```

```

PRINT_BUF PROC NEAR
    PUSH AX
    MOV AH, 9H
    INT 21H
    POP AX
    RET
PRINT_BUF ENDP

```

```

PRINT_PARAGRAPH PROC NEAR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI

    MOV SI, DX
    MOV BX, 0AH
    XOR CX, CX

```

```

DIVISION_LOOP_AX:
    DIV BX
    PUSH DX
    XOR DX, DX
    INC CX
    CMP AX, 0H
    JNE DIVISION_LOOP_AX

```

```

PRINT_SYMBOL_LOOP:
    POP DX
    ADD DX, 30H ; -Γ'!-® -Γ' 3ĚB«®,    €®¤ БĚ¬Ÿ®«
    MOV AH, 02H
    INT 21H
    LOOP PRINT_SYMBOL_LOOP

```

```

    POP SI
    POP DX
    POP CX
    POP BX
    POP AX
    RET
PRINT_PARAGRAPH ENDP

AVAILABLE_MEMORY PROC NEAR
    PUSH AX
    PUSH BX
    PUSH DX

    MOV DX, OFFSET AVAILABLE_MEMORY_STRING
    CALL PRINT_BUF

    MOV AH, 4AH
    MOV BX, 0FFFFH
    INT 21H
    MOV AX, BX

    XOR DX, DX
    MOV BX, 010H ; ĨΓAΓŸ®ĚĚ ĚŠ Ĩ A JA Д®Ÿ Ÿ Ÿ ©БЛ
    MUL BX

    CALL PRINT_PARAGRAPH
    MOV DX, OFFSET BYTES_STRING
    CALL PRINT_BUF
    CALL PRINT_NEW_LINE

    POP DX
    POP BX
    POP AX
    RET
AVAILABLE_MEMORY ENDP

```

```

EXTENDED_MEMORY PROC NEAR

    PUSH AX

    PUSH BX

    PUSH DX


    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 DX, OFFSET EXTENDED_MEMORY_STRING
    CALL PRINT_BUF


    MOV AX, BX
    XOR DX, DX
    CALL PRINT_PARAGRAPH
    MOV DX, OFFSET KILOBYTES_STRING
    CALL PRINT_BUF
    CALL PRINT_NEW_LINE


    POP DX

    POP BX

    POP AX

    RET

EXTENDED_MEMORY ENDP


MCB_ADDRESS PROC NEAR

    PUSH DX

    PUSH AX

    PUSH ES

    PUSH DI


    MOV AX, ES
    MOV DI, OFFSET ADDRESS_STRING

```

```

    ADD DI, 12
    CALL WRD_TO_HEX

    MOV DX, OFFSET ADDRESS_STRING
    CALL PRINT_BUF
    POP DI
    POP ES
    POP AX
    POP DX
    RET
MCB_ADDRESS ENDP

MCB_PSP_TYPE PROC NEAR
    PUSH AX
    PUSH DX
    PUSH DI
    MOV AX, ES:[1]
    MOV DX, OFFSET TAB
    CALL PRINT_BUF
    MOV DX, OFFSET PSP_TYPE_DECLORATION
    CALL PRINT_BUF

    CMP AX, 0000H
    JE PRINT_1

    CMP AX, 0006H
    JE PRINT_2

    CMP AX, 0007H
    JE PRINT_3

    CMP AX, 0008H
    JE PRINT_4

    CMP AX, 0FFFAH
    JE PRINT_5

    CMP AX, 0FFFDH

```

```

    JE PRINT_6

    CMP AX, 0FFFFEH
    JE PRINT_7

    JMP PRINT_8

PRINT_1:
    MOV DX, OFFSET FREE_AREA
    CALL PRINT_BUF
    JMP EXIT2

PRINT_2:
    MOV DX, OFFSET OS_XMS_UMB
    CALL PRINT_BUF
    JMP EXIT2

PRINT_3:
    MOV DX, OFFSET EXCLUDED_HIGH
    CALL PRINT_BUF
    JMP EXIT2

PRINT_4:
    MOV DX, OFFSET BELONGS_MSDOS
    CALL PRINT_BUF
    JMP EXIT2

PRINT_5:
    MOV DX, OFFSET BUSY_386MAX_UMB
    CALL PRINT_BUF
    JMP EXIT2

PRINT_6:
    MOV DX, OFFSET BLOCKED_386MAX
    CALL PRINT_BUF
    JMP EXIT2

PRINT_7:
    MOV DX, OFFSET BELONGS_386MAX
    CALL PRINT_BUF

PRINT_8:
    MOV DI, OFFSET STRING_FOR_PSP_TYPE
    ADD DI, 3
    CALL WRD_TO_HEX
    MOV DX, OFFSET STRING_FOR_PSP_TYPE

```

```

        CALL PRINT_BUF
EXIT2:
        POP DI
        POP DX
        POP AX
        RET
MCB_PSP_TYPE ENDP

MCB_SIZE PROC NEAR
        PUSH AX
        PUSH DX
        PUSH ES
        PUSH DI

        MOV DX, OFFSET TAB
        CALL PRINT_BUF
        MOV AX, ES:[3]
        MOV DI, OFFSET MCB_SIZE_STRING
        ADD DI, 8
        CALL WRD_TO_HEX
        MOV DX, OFFSET MCB_SIZE_STRING
        CALL PRINT_BUF

        POP DI
        POP ES
        POP DX
        POP AX
        RET
MCB_SIZE ENDP

MCB_SC_SD PROC NEAR
        PUSH DX
        PUSH AX
        PUSH ES
        PUSH DI
        PUSH CX

        MOV DX, OFFSET MCB_SC_SD_STRING

```

```

CALL PRINT_BUF

MOV BX, 8
MOV CX, 08H
SCSD_LOOP:
    MOV DL, ES:[BX]
    MOV AH, 02H
    INT 21H
    INC BX
    LOOP SCSD_LOOP
EXIT3:
    POP CX
    POP DI
    POP ES
    POP AX
    POP DX
    RET
MCB_SC_SD ENDP

MCB_TABLE PROC NEAR
    PUSH AX
    PUSH BX
    PUSH DX
    PUSH CX
    PUSH SI

    CALL PRINT_NEW_LINE
    MOV DX, OFFSET MCB_TABLE_DECLORATION
    CALL PRINT_BUF

    MOV AH, 52H
    INT 21H
    MOV AX, ES:[BX-2]
    MOV ES, AX
    MOV CL, 01H

MCB_LINE_LOOP:

```



```

MOV AL, CL
MOV SI, OFFSET MCB_NUMBER
ADD SI, 4
CALL BYTE_TO_DEC
MOV DX, OFFSET MCB_NUMBER
CALL PRINT_BUF

CALL MCB_ADDRESS
CALL MCB_SIZE
CALL MCB_SC_SD
CALL MCB_PSP_TYPE
CALL PRINT_NEW_LINE

MOV AH, ES:[0]
CMP AH, 5AH
JE EXIT

MOV BX, ES:[3]
INC BX
MOV AX, ES
ADD AX, BX
MOV ES, AX
INC CL
JMP MCB_LINE_LOOP

EXIT:
POP SI
POP CX
POP DX
POP BX
POP AX
RET
MCB_TABLE ENDP

BEGIN:

CALL AVAILABLE_MEMORY

```

```

CALL EXTENDED_MEMORY
CALL MCB_TABLE

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

MCB_TABLE_DECLARATION db 'MCB TABLE: ', 0DH, 0AH, '$'
AVAILABLE_MEMORY_STRING db 'Available memory: $'
EXTENDED_MEMORY_STRING db 'Extended memory: $'
BYTES_STRING db ' bytes$'
KILOBYTES_STRING db ' Kbytes$'
MCB_SIZE_STRING db 'Size:      $'
FREE_AREA db 'free area          $'
OS_XMS_UMB db 'belongs to OS XMS UMB      $'
EXCLUDED_HIGH db 'excluded high memory    $'
BELONGS_MSDOS db 'belongs MSDOS          $'
BUSY_386MAX_UMB db 'busy with block 386MAX UMB$'
BLOCKED_386MAX db 'blocked 386MAX          $'
BELONGS_386MAX db 'belongs 386MAX          $'
ADDRESS_STRING db 'Address:      $'
MCB_SC_SD_STRING db 'SC/SD: $'
TAB db ' $'
STRING_FOR_PSP_TYPE db '                                $'
PSP_TYPE_DECLARATION db 'PSP TYPE:$'

MCB_NUMBER db 'MCB_ $'

```

```

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 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_l
    or al, 30h
    mov [si], al

end_l:
    pop dx
    pop cx
    pop ax
    ret

BYTE_TO_DEC endp

PRINT_NEW_LINE proc near
    push ax
    push dx

    mov dl, 0Dh

```

```

    mov ah, 02h
    int 21h

    mov dl, 0Ah
    mov ah, 02h
    int 21h

    pop dx
    pop ax
    ret
PRINT_NEW_LINE endp

PRINT_BUF proc near
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
PRINT_BUF endp

PRINT_PARAGRAPH proc near
    push ax
    push bx
    push cx
    push dx
    push si

    mov si, dx
    mov bx, 0ah
    xor cx, cx

division_loop_ax:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h

```

```

    jne division_loop_ax

print_symbol_loop:
    pop dx
    add dx, 30h ;нужно не число, а код символа
    mov ah, 02h
    int 21h
    loop print_symbol_loop

    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    ret
PRINT_PARAGRAPH endp

AVAILABLE_MEMORY proc near
    push ax
    push bx
    push dx

    mov dx, offset AVAILABLE_MEMORY_STRING
    call PRINT_BUF

    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx

    xor dx, dx
    mov bx, 010h ;переводим из параграфов в байты
    mul bx

    call PRINT_PARAGRAPH
    mov dx, offset BYTES_STRING
    call PRINT_BUF
    call PRINT_NEW_LINE

```

```

    pop dx
    pop bx
    pop ax
    ret
AVAILABLE_MEMORY endp

EXTENDED_MEMORY proc near
    push ax
    push bx
    push dx

    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 dx, offset EXTENDED_MEMORY_STRING
    call PRINT_BUF

    mov ax, bx
    xor dx, dx
    call PRINT_PARAGRAPH
    mov dx, offset KILOBYTES_STRING
    call PRINT_BUF
    call PRINT_NEW_LINE

    pop dx
    pop bx
    pop ax
    ret
EXTENDED_MEMORY endp

```

```

MCB_ADDRESS proc near
    push dx
    push ax
    push es
    push di

    mov ax, es
    mov di, offset ADDRESS_STRING
    add di, 12
    call WRD_TO_HEX

    mov dx, offset ADDRESS_STRING
    call PRINT_BUF
    pop di
    pop es
    pop ax
    pop dx
    ret
MCB_ADDRESS endp

MCB_PSP_TYPE proc near
    push ax
    push dx
    push di
    mov ax, es:[1]
    mov dx, offset TAB
    call PRINT_BUF
    mov dx, offset PSP_TYPE_DECLARATION
    call PRINT_BUF

    cmp ax, 0000h
    je print_1

    cmp ax, 0006h
    je print_2

    cmp ax, 0007h
    je print_3

```



```

        cmp ax, 0008h
        je print_4

        cmp ax, 0FFFAh
        je print_5

        cmp ax, 0FFFDh
        je print_6

        cmp ax, 0FFFEh
        je print_7

        jmp print_8
print_1:
        mov dx, offset FREE_AREA
        call PRINT_BUF
        jmp exit2
print_2:
        mov dx, offset OS_XMS_UMB
        call PRINT_BUF
        jmp exit2
print_3:
        mov dx, offset EXCLUDED_HIGH
        call PRINT_BUF
        jmp exit2
print_4:
        mov dx, offset BELONGS_MSDOS
        call PRINT_BUF
        jmp exit2
print_5:
        mov dx, offset BUSY_386MAX_UMB
        call PRINT_BUF
        jmp exit2
print_6:
        mov dx, offset BLOCKED_386MAX
        call PRINT_BUF
        jmp exit2

```

```

print_7:
    mov dx, offset BELONGS_386MAX
    call PRINT_BUF
print_8:
    mov di, offset STRING_FOR_PSP_TYPE
    add di, 3
    call WRD_TO_HEX
    mov dx, offset STRING_FOR_PSP_TYPE
    call PRINT_BUF
exit2:
    pop di
    pop dx
    pop ax
    ret
MCB_PSP_TYPE endp

MCB_SIZE proc near
    push ax
    push dx
    push es
    push di

    mov dx, offset TAB
    call PRINT_BUF
    mov ax, es:[3]
    mov di, offset MCB_SIZE_STRING
    add di, 8
    call WRD_TO_HEX
    mov dx, offset MCB_SIZE_STRING
    call PRINT_BUF

    pop di
    pop es
    pop dx
    pop ax
    ret
MCB_SIZE endp

```

```

MCB_SC_SD proc near
    push dx
    push ax
    push es
    push di
    push cx

    mov dx, offset MCB_SC_SD_STRING
    call PRINT_BUF

    mov bx, 8
    mov cx, 08h
scsd_loop:
    mov dl, es:[bx]
    mov ah, 02h
    int 21h
    inc bx
    loop scsd_loop
exit3:
    pop cx
    pop di
    pop es
    pop ax
    pop dx
    ret
MCB_SC_SD endp

MCB_TABLE proc near
    push ax
    push bx
    push dx
    push cx
    push si

    call PRINT_NEW_LINE
    mov dx, offset MCB_TABLE_DECLORATION
    call PRINT_BUF

```

```

    mov ah, 52h
    int 21h
    mov ax, es:[bx-2]
    mov es, ax
    mov cl, 01h

mcb_line_loop:
    mov al, cl
    mov si, offset MCB_NUMBER
    add si, 4
    call BYTE_TO_DEC
    mov dx, offset MCB_NUMBER
    call PRINT_BUF

    call MCB_ADDRESS
    call MCB_SIZE
    call MCB_SC_SD
    call MCB_PSP_TYPE
    call PRINT_NEW_LINE

    mov ah, es:[0]
    cmp ah, 5ah
    je exit

    mov bx, es:[3]
    inc bx
    mov ax, es
    add ax, bx
    mov es, ax
    inc cl
    jmp mcb_line_loop

exit:
    pop si
    pop cx
    pop dx

```

```

        pop bx
        pop ax
        ret
MCB_TABLE endp

FREE_MEMORY proc near
    push ax
    push bx
    push dx

    mov ax, offset end_of_proc
    mov bx, 10h
    xor dx, dx
    div bx
    mov bx, ax
    add bx, dx

    mov ah, 4ah
    int 21h
    pop dx
    pop bx
    pop ax
    ret
FREE_MEMORY endp

begin:
    call AVAILABLE_MEMORY
    call EXTENDED_MEMORY
    call FREE_MEMORY
    call MCB_TABLE

    xor al, al
    mov ah, 4ch
    int 21h
end_of_proc:
TESTPC  ENDS
        END start

```

lab3_3.asm:

```
TESTPC    SEGMENT
            ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
org 100h

start: jmp begin

MCB_TABLE_DECLARATION db 'MCB TABLE: ', 0DH, 0AH, '$'
AVAILABLE_MEMORY_STRING db 'Available memory: $'
EXTENDED_MEMORY_STRING db 'Extended memory: $'
BYTES_STRING db ' bytes$'
KILOBYTES_STRING db ' Kbytes$'
MCB_SIZE_STRING db 'Size:      $'
FREE_AREA db 'free area                $'
OS_XMS_UMB db 'belongs to OS XMS UMB    $'
EXCLUDED_HIGH db 'excluded high memory $'
BELONGS_MSXDOS db 'belongs MSDOS        $'
BUSY_386MAX_UMB db 'busy with block 386MAX UMB$'
BLOCKED_386MAX db 'blocked 386MAX        $'
BELONGS_386MAX db 'belongs 386MAX        $'
ADDRESS_STRING db 'Address:      $'
MCB_SC_SD_STRING db 'SC/SD: $'
TAB db ' $'
STRING_FOR_PSP_TYPE db '                                $'
PSP_TYPE_DECLARATION db 'PSP TYPE:$'

MCB_NUMBER db 'MCB_  $'

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 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_l
        or al, 30h
        mov [si], al

end_l:
        pop dx
        pop cx
        pop ax
        ret

BYTE_TO_DEC endp

PRINT_NEW_LINE proc near
        push ax
        push dx

        mov dl, 0Dh
        mov ah, 02h
        int 21h

        mov dl, 0Ah
        mov ah, 02h
        int 21h

        pop dx
        pop ax
        ret

```



```
PRINT_NEW_LINE endp
```

```
PRINT_BUF proc near
```

```
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
```

```
PRINT_BUF endp
```

```
PRINT_PARAGRAPH proc near
```

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

```
    mov si, dx
    mov bx, 0ah
    xor cx, cx
```

```
division_loop_ax:
```

```
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_ax
```

```
print_symbol_loop:
```

```
    pop dx
    add dx, 30h ;нужно не число, а код символа
    mov ah, 02h
    int 21h
    loop print_symbol_loop
```

```
    pop si
```

```

    pop dx
    pop cx
    pop bx
    pop ax
    ret
PRINT_PARAGRAPH endp

AVAILABLE_MEMORY proc near
    push ax
    push bx
    push dx

    mov dx, offset AVAILABLE_MEMORY_STRING
    call PRINT_BUF

    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx

    xor dx, dx
    mov bx, 010h ;переводим из параграфов в байты
    mul bx

    call PRINT_PARAGRAPH
    mov dx, offset BYTES_STRING
    call PRINT_BUF
    call PRINT_NEW_LINE

    pop dx
    pop bx
    pop ax
    ret
AVAILABLE_MEMORY endp

EXTENDED_MEMORY proc near
    push ax

```

```

    push bx
    push dx

    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 dx, offset EXTENDED_MEMORY_STRING
    call PRINT_BUF

    mov ax, bx
    xor dx, dx
    call PRINT_PARAGRAPH
    mov dx, offset KILOBYTES_STRING
    call PRINT_BUF
    call PRINT_NEW_LINE

    pop dx
    pop bx
    pop ax
    ret
EXTENDED_MEMORY endp

MCB_ADDRESS proc near
    push dx
    push ax
    push es
    push di

    mov ax, es
    mov di, offset ADDRESS_STRING
    add di, 12
    call WRD_TO_HEX

```

```

    mov dx, offset ADDRESS_STRING
    call PRINT_BUF
    pop di
    pop es
    pop ax
    pop dx
    ret
MCB_ADDRESS endp

MCB_PSP_TYPE proc near
    push ax
    push dx
    push di
    mov ax, es:[1]
    mov dx, offset TAB
    call PRINT_BUF
    mov dx, offset PSP_TYPE_DECLARATION
    call PRINT_BUF

    cmp ax, 0000h
    je print_1

    cmp ax, 0006h
    je print_2

    cmp ax, 0007h
    je print_3

    cmp ax, 0008h
    je print_4

    cmp ax, 0FFFAh
    je print_5

    cmp ax, 0FFFDh
    je print_6

```

```

        cmp ax, 0FFFEh
        je print_7

        jmp print_8
print_1:
        mov dx, offset FREE_AREA
        call PRINT_BUF
        jmp exit2
print_2:
        mov dx, offset OS_XMS_UMB
        call PRINT_BUF
        jmp exit2
print_3:
        mov dx, offset EXCLUDED_HIGH
        call PRINT_BUF
        jmp exit2
print_4:
        mov dx, offset BELONGS_MSDOS
        call PRINT_BUF
        jmp exit2
print_5:
        mov dx, offset BUSY_386MAX_UMB
        call PRINT_BUF
        jmp exit2
print_6:
        mov dx, offset BLOCKED_386MAX
        call PRINT_BUF
        jmp exit2
print_7:
        mov dx, offset BELONGS_386MAX
        call PRINT_BUF
print_8:
        mov di, offset STRING_FOR_PSP_TYPE
        add di, 3
        call WRD_TO_HEX
        mov dx, offset STRING_FOR_PSP_TYPE
        call PRINT_BUF
exit2:

```

```

        pop di
        pop dx
        pop ax
        ret
MCB_PSP_TYPE endp

MCB_SIZE proc near
    push ax
    push dx
    push es
    push di

    mov dx, offset TAB
    call PRINT_BUF
    mov ax, es:[3]
    mov di, offset MCB_SIZE_STRING
    add di, 8
    call WRD_TO_HEX
    mov dx, offset MCB_SIZE_STRING
    call PRINT_BUF

    pop di
    pop es
    pop dx
    pop ax
    ret
MCB_SIZE endp

MCB_SC_SD proc near
    push dx
    push ax
    push es
    push di
    push cx

    mov dx, offset MCB_SC_SD_STRING
    call PRINT_BUF

```

```

        mov bx, 8
        mov cx, 08h
scsd_loop:
        mov dl, es:[bx]
        mov ah, 02h
        int 21h
        inc bx
        loop scsd_loop
exit3:
        pop cx
        pop di
        pop es
        pop ax
        pop dx
        ret
MCB_SC_SD endp

MCB_TABLE proc near
        push ax
        push bx
        push dx
        push cx
        push si

        call PRINT_NEW_LINE
        mov dx, offset MCB_TABLE_DECLORATION
        call PRINT_BUF

        mov ah, 52h
        int 21h
        mov ax, es:[bx-2]
        mov es, ax
        mov cl, 01h

mcb_line_loop:
        mov al, cl
        mov si, offset MCB_NUMBER

```

```

    add si, 4
    call BYTE_TO_DEC
    mov dx, offset MCB_NUMBER
    call PRINT_BUF

    call MCB_ADDRESS
    call MCB_SIZE
    call MCB_SC_SD
    call MCB_PSP_TYPE
    call PRINT_NEW_LINE

    mov ah, es:[0]
    cmp ah, 5ah
    je exit

    mov bx, es:[3]
    inc bx
    mov ax, es
    add ax, bx
    mov es, ax
    inc cl
    jmp mcb_line_loop

exit:
    pop si
    pop cx
    pop dx
    pop bx
    pop ax
    ret
MCB_TABLE endp

FREE_MEMORY proc near
    push ax
    push bx
    push dx

```



```

    mov ax, offset end_of_proc
    mov bx, 10h
    xor dx, dx
    div bx
    mov bx, ax
    add bx, dx

    mov ah, 4ah
    int 21h
    pop dx
    pop bx
    pop ax
    ret
FREE_MEMORY endp

```

```

REQUEST_MEMORY proc near
    push ax
    push bx
    push dx

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

    pop dx
    pop bx
    pop ax
    ret
REQUEST_MEMORY endp

```

```

begin:
    call AVAILABLE_MEMORY
    call EXTENDED_MEMORY
    call FREE_MEMORY
    call REQUEST_MEMORY
    call MCB_TABLE

```

```

        xor al, al
        mov ah, 4ch
        int 21h
end_of_proc:
TESTPC  ENDS

        END st

```

lab3_4.asm:

```

TESTPC  SEGMENT

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

org 100h


start: jmp begin


MCB_TABLE_DECLARATION db 'MCB TABLE: ', 0DH, 0AH, '$'
AVAILABLE_MEMORY_STRING db 'Available memory: $'
EXTENDED_MEMORY_STRING db 'Extended memory: $'
BYTES_STRING db ' bytes$'
KILOBYTES_STRING db ' Kbytes$'
MCB_SIZE_STRING db 'Size:      $'
FREE_AREA db 'free area                $'
OS_XMS_UMB db 'belongs to OS XMS UMB      $'
EXCLUDED_HIGH db 'excluded high memory    $'
BELONGS_MSDOS db 'belongs MSDOS           $'
BUSY_386MAX_UMB db 'busy with block 386MAX UMB$'
BLOCKED_386MAX db 'blocked 386MAX          $'
BELONGS_386MAX db 'belongs 386MAX          $'
ADDRESS_STRING db 'Address:      $'
MCB_SC_SD_STRING db 'SC/SD: $'
TAB db ' $'
STRING_FOR_PSP_TYPE db '                                $'
PSP_TYPE_DECLARATION db 'PSP TYPE:$'
REQUEST_MESSAGE db 'Cannot allocate memory$'


MCB_NUMBER db 'MCB_    $'


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 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_l
    or al, 30h
    mov [si], al
```

```
end_l:
```

```
    pop dx
    pop cx
    pop ax
    ret
```

```
BYTE_TO_DEC endp
```

```
PRINT_NEW_LINE proc near
```

```
    push ax
    push dx
```

```
    mov dl, 0Dh
    mov ah, 02h
    int 21h
```

```

    mov dl, 0Ah
    mov ah, 02h
    int 21h

    pop dx
    pop ax
    ret
PRINT_NEW_LINE endp

PRINT_BUF proc near
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
PRINT_BUF endp

PRINT_PARAGRAPH proc near
    push ax
    push bx
    push cx
    push dx
    push si

    mov si, dx
    mov bx, 0ah
    xor cx, cx

division_loop_ax:
    div bx
    push dx
    xor dx, dx
    inc cx
    cmp ax, 0h
    jne division_loop_ax

print_symbol_loop:

```

```

    pop dx
    add dx, 30h ;нужно не число, а код символа
    mov ah, 02h
    int 21h
    loop print_symbol_loop

    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    ret
PRINT_PARAGRAPH endp

AVAILABLE_MEMORY proc near
    push ax
    push bx
    push dx

    mov dx, offset AVAILABLE_MEMORY_STRING
    call PRINT_BUF

    mov ah, 4ah
    mov bx, 0ffffh
    int 21h
    mov ax, bx

    xor dx, dx
    mov bx, 010h ;переводим из параграфов в байты
    mul bx

    call PRINT_PARAGRAPH
    mov dx, offset BYTES_STRING
    call PRINT_BUF
    call PRINT_NEW_LINE

    pop dx

```

```

    pop bx
    pop ax
    ret
AVAILABLE_MEMORY endp

EXTENDED_MEMORY proc near
    push ax
    push bx
    push dx

    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 dx, offset EXTENDED_MEMORY_STRING
    call PRINT_BUF

    mov ax, bx
    xor dx, dx
    call PRINT_PARAGRAPH
    mov dx, offset KILOBYTES_STRING
    call PRINT_BUF
    call PRINT_NEW_LINE

    pop dx
    pop bx
    pop ax
    ret
EXTENDED_MEMORY endp

MCB_ADDRESS proc near
    push dx
    push ax

```

```

    push es
    push di

    mov ax, es
    mov di, offset ADDRESS_STRING
    add di, 12
    call WRD_TO_HEX

    mov dx, offset ADDRESS_STRING
    call PRINT_BUF
    pop di
    pop es
    pop ax
    pop dx
    ret
MCB_ADDRESS endp

MCB_PSP_TYPE proc near
    push ax
    push dx
    push di
    mov ax, es:[1]
    mov dx, offset TAB
    call PRINT_BUF
    mov dx, offset PSP_TYPE_DECLARATION
    call PRINT_BUF

    cmp ax, 0000h
    je print_1

    cmp ax, 0006h
    je print_2

    cmp ax, 0007h
    je print_3

    cmp ax, 0008h
    je print_4

```



```

    cmp ax, 0FFFAh
    je print_5

    cmp ax, 0FFFDh
    je print_6

    cmp ax, 0FFFEh
    je print_7

    jmp print_8
print_1:
    mov dx, offset FREE_AREA
    call PRINT_BUF
    jmp exit2
print_2:
    mov dx, offset OS_XMS_UMB
    call PRINT_BUF
    jmp exit2
print_3:
    mov dx, offset EXCLUDED_HIGH
    call PRINT_BUF
    jmp exit2
print_4:
    mov dx, offset BELONGS_MSDOS
    call PRINT_BUF
    jmp exit2
print_5:
    mov dx, offset BUSY_386MAX_UMB
    call PRINT_BUF
    jmp exit2
print_6:
    mov dx, offset BLOCKED_386MAX
    call PRINT_BUF
    jmp exit2
print_7:
    mov dx, offset BELONGS_386MAX
    call PRINT_BUF

```

```

print_8:
    mov di, offset STRING_FOR_PSP_TYPE
    add di, 3
    call WRD_TO_HEX
    mov dx, offset STRING_FOR_PSP_TYPE
    call PRINT_BUF
exit2:
    pop di
    pop dx
    pop ax
    ret
MCB_PSP_TYPE endp

MCB_SIZE proc near
    push ax
    push dx
    push es
    push di

    mov dx, offset TAB
    call PRINT_BUF
    mov ax, es:[3]
    mov di, offset MCB_SIZE_STRING
    add di, 8
    call WRD_TO_HEX
    mov dx, offset MCB_SIZE_STRING
    call PRINT_BUF

    pop di
    pop es
    pop dx
    pop ax
    ret
MCB_SIZE endp

MCB_SC_SD proc near
    push dx
    push ax

```

```

    push es
    push di
    push cx

    mov dx, offset MCB_SC_SD_STRING
    call PRINT_BUF

    mov bx, 8
    mov cx, 08h
scsd_loop:
    mov dl, es:[bx]
    mov ah, 02h
    int 21h
    inc bx
    loop scsd_loop
exit3:
    pop cx
    pop di
    pop es
    pop ax
    pop dx
    ret
MCB_SC_SD endp

MCB_TABLE proc near
    push ax
    push bx
    push dx
    push cx
    push si

    call PRINT_NEW_LINE
    mov dx, offset MCB_TABLE_DECLORATION
    call PRINT_BUF

    mov ah, 52h
    int 21h

```

```

    mov ax, es:[bx-2]
    mov es, ax
    mov cl, 01h

mcb_line_loop:
    mov al, cl
    mov si, offset MCB_NUMBER
    add si, 4
    call BYTE_TO_DEC
    mov dx, offset MCB_NUMBER
    call PRINT_BUF

    call MCB_ADDRESS
    call MCB_SIZE
    call MCB_SC_SD
    call MCB_PSP_TYPE
    call PRINT_NEW_LINE

    mov ah, es:[0]
    cmp ah, 5ah
    je exit

    mov bx, es:[3]
    inc bx
    mov ax, es
    add ax, bx
    mov es, ax
    inc cl
    jmp mcb_line_loop

exit:
    pop si
    pop cx
    pop dx
    pop bx
    pop ax
    ret

```

```

MCB_TABLE endp

FREE_MEMORY proc near
    push ax
    push bx
    push dx

    mov ax, offset end_of_proc
    mov bx, 10h
    xor dx, dx
    div bx
    mov bx, ax
    add bx, dx

    mov ah, 4ah
    int 21h
    pop dx
    pop bx
    pop ax
    ret
FREE_MEMORY endp

REQUEST_MEMORY proc near
    push ax
    push bx
    push dx

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

    jc memory_allocate_failed
    jmp exit4
memory_allocate_failed:
    mov dx, offset REQUEST_MESSAGE
    call PRINT_BUF
exit4:
    pop dx

```

```

        pop bx
        pop ax
        ret
REQUEST_MEMORY endp

begin:
        call AVAILABLE_MEMORY
        call EXTENDED_MEMORY
        call REQUEST_MEMORY
        call FREE_MEMORY
        call MCB_TABLE

        xor al, al
        mov ah, 4ch
        int 21h
end_of_proc:
TESTPC  ENDS
        END start

```