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 _____ Кривенцова Л.С.

Преподаватель _____ Губкин А.Ф.

Санкт-Петербург

2022

Цель работы.

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

Задание.

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

- количество доступной памяти;

- размер расширенной памяти;

- выводит цепочку блоков управления памятью.

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

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

2. Измените программу таким образом, чтобы она освобождала память, которую она не занимает. Для этого используйте функцию 4Ah прерывания 21h (пример в разделе “Использование функции 4Ah”). Повторите эксперимент, запустив модифицированную программу. Сравните выходные данные с результатами, полученными на предыдущем шаге. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

3. Измените программу еще раз таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти функцией 48h прерывания 21h. Повторите эксперимент, запустив модифицированную программу. Сравните

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

4. Измените первоначальный вариант программы, запросив 64Кб памяти функцией 48h прерывания 21h до освобождения памяти. Обязательно обрабатывайте завершение функций ядра, проверяя флаг CF.

Выполнение работы.

Исходные коды содержатся в файле lb3_part1.asm - lb3_part4.asm.

1. Для выполнения первого шага были написаны следующие процедуры:

- Процедура _AM выводит доступную память (в байтах).
- Процедура _EM для демонстрирует расширенную память (в байтах).
- Процедура _MCB демонстрирует цепочки блоков управления памятью.

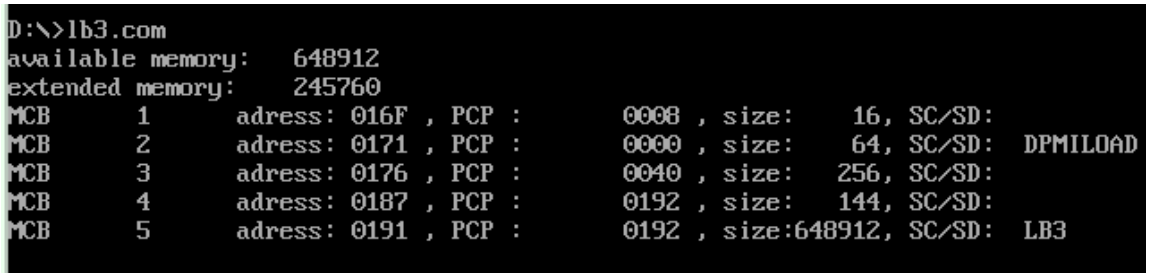

Рис. 1. – Вывод программы на первом шаге.

Здесь программа использует всю доступную память.

2. Для выполнения второго шага реализована процедура memory_free. Она освобождает память, которую не занимает программа.

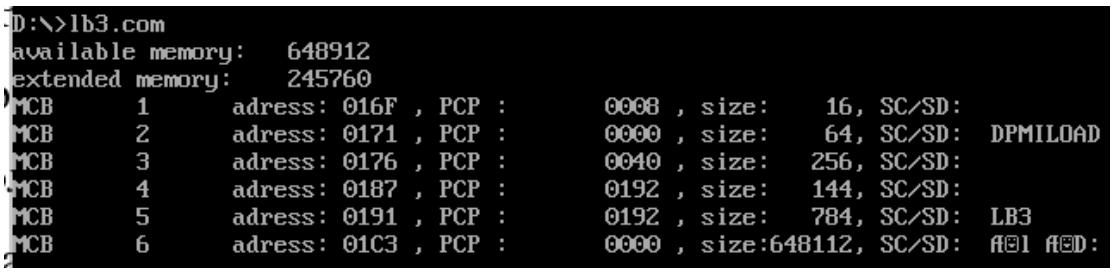

Рис. 2. – Вывод программы на втором шаге.

Теперь программа занимает лишь ту область памяти, что необходима для ее хранения.

3. Для выполнения третьего шага была реализована процедура memory2. Благодаря ей после освобождения памяти, программа запрашивает 64Кб памяти функцией 48h прерывания 21h. Важно, что вызывается она уже после процедуры memory_free.

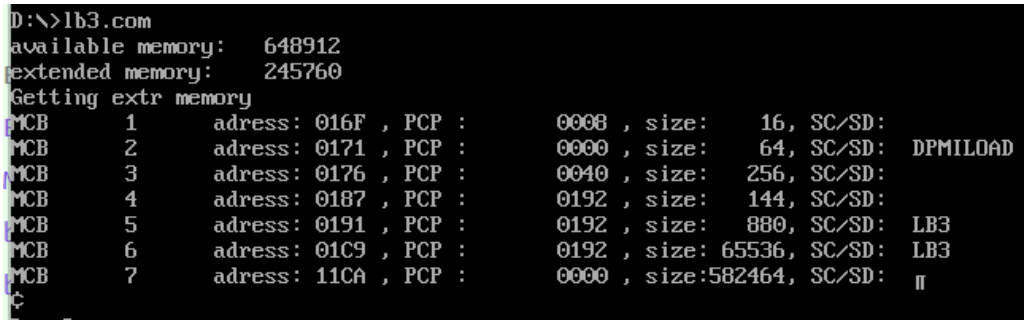

Рис. 3. – Вывод программы на третьем шаге.

Заметно, что на этом шаге программе, после освобождения памяти, выделяется дополнительный блок памяти в размере 64Кб.

4. Для выполнения четвертого шага memory2 вызывается ещё до вызова процедуры memory_free. Таким образом запрос программы 64Кб памяти происходит до освобождения памяти.

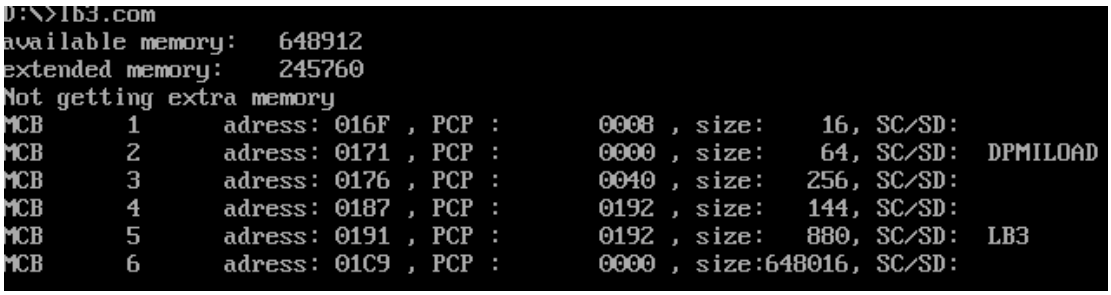

Рис. 4. – Вывод программы на четвертом шаге.

Программа не получает дополнительный блок памяти, т.к. она занимает всю доступную память, а освобождение неиспользованной происходит уже после запроса доп.памяти.

Контрольные вопросы.

1. Что означает “доступный объем памяти”?

Объем памяти, отведенный управляющей программой под модуль.

2. Где MCB блок вашей программы в списке?

Блок (/блоки, т.к. на шаге 3 программа имеет два MCB блока), у которого в графе SC/SD указано наименование программы.

В шаге 1: последний блок в списке.

В шаге 2 и 4: предпоследний блок в списке, так как после него расположена свободная память.

В шаге 3 – два предпоследних блока. Свободная память расположена аналогично шагу 2 и 4.

3. Какой размер памяти занимает программа в каждом случае?

Шаг 1: вся доступная память – 648912 байта.

Шаг 2: 748 байта – столько точно нужно выделить программе.

Шаг 3: 880 + 65536 байта – ровно необходимая для программы память и дополнительные 64 Кбайт.

Шаг 4: 880 байта, т.к. запрос на доп.память не удовлетворён.

Выводы.

Были исследованы структуры данных и работа функций управления памятью ядра операционной системы.

ПРИЛОЖЕНИЕ А.

Исходный код.

lb3_part1.asm:

```
TESTPC SEGMENT
     ASSUME CS:TESTPC, DS:TESTPC
     ORG  100H
START:    jmp BEGIN


     av_mem_print db 'available memory:           ', 0DH, 0AH, '$'
     ext_mem_print db 'extended memory:            ', 0DH, 0AH, '$'
     MCB_print db 'MCB              adress:       , PCP :             ,
size:      , SC/SD:           ', 0DH, '$'

     TETR_TO_HEX PROC
          and AL, 0Fh
          cmp AL, 09
          jbe NEXT
          add AL, 07
     NEXT: add AL, 30h
          ret
     TETR_TO_HEX ENDP

     BYTE_TO_HEX PROC
     ; б а й т  в  AL  п е р е в о д и т с я  в  д в а  с и м в о л а  16-г о  ч и
с л а  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 с т а р ш а я  ц и ф р а
          pop CX           ; в AH м л а д ш а я
          ret
     BYTE_TO_HEX ENDP
```

```
WRD_TO_HEX PROC
      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
      push AX
      push CX
      push DX
      xor AH, AH
      xor DX, DX
      mov CX, 10
loop_bd: div CX
      or DL, 30h
      mov [SI], DL
      dec SI
      xor DX, DX
      cmp AX, 10
      jae loop_bd
      cmp AL, 00h
      je end_1
      or AL, 30h
      mov [SI], AL
end_1:
      pop DX
      pop CX
      pop AX
```

```
      ret
BYTE_TO_DEC ENDP

WORD_TO_DEC PROC
      push AX
      push BX
      push DX
      push CX
      push SI
      mov BX, 10h
      mul BX
      mov BX, 0Ah
segmentation:
      div BX
      or DX, 30h
      mov [SI], DL
      dec SI
      xor DX, DX
      cmp AX, 0h
      jne segmentation
      pop SI
      pop CX
      pop DX
      pop BX
      pop AX
      ret
WORD_TO_DEC ENDP

print PROC
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
print ENDP
print_sym PROC
      push AX
      mov AH, 02h
      int 21h
```

```
      pop AX
      ret
print_sym ENDP

AM PROC
      push AX
      push BX
      push SI
      xor AX, AX
      mov AH, 4Ah
      mov BX, 0FFFFh
      int 21h
      mov AX, BX
      mov SI, offset av_mem_print
      add SI, 25
      call WORD_TO_DEC
      mov DX, offset av_mem_print
      call print
      pop SI
      pop BX
      pop AX
      ret
AM ENDP

EM PROC
      push AX
      push BX
      push SI
      xor AX, AX
      mov AL, 30h
      out 70h, AL
      in AL, 71h
      mov BL, AL
      mov AL, 31h
      out 70h, AL
      in AL, 71h
      mov AH, AL
      mov AL, BL
      mov SI, offset ext_mem_print
```

```
        add SI, 25
        call WORD_TO_DEC
        mov DX, offset ext_mem_print
        call print
        pop SI
        pop BX
        pop AX
        ret
EM ENDP
MCB PROC
        push AX
        push BX
        push CX
        push DX
        push ES
        push SI
        xor AX, AX
        mov AH, 52h
        int 21h
        mov AX, ES:[BX-2]
        mov ES, AX
        mov CL, 1
loop_for_MCB:
        mov AL, CL
        mov SI, offset MCB_print
        add SI, 9
        call BYTE_TO_DEC
        mov AX, ES
        mov DI, offset MCB_print
        add DI, 27
        call WRD_TO_HEX
        mov AX, ES:[01h]
        mov DI, offset MCB_print
        add DI, 46
        call WRD_TO_HEX
        mov AX, ES:[03h]
        add SI, 52
        call WORD_TO_DEC
        mov BX, 8
```

```
      push CX
      mov CX, 7
      add SI, 11
loop_sc_or_sd:
      mov DX, ES:[BX]
      mov DS:[SI], DX
      inc BX
      inc SI
      loop loop_sc_or_sd
      mov DX, offset MCB_print
      call print
      mov AH, ES:[0]
      cmp AH, 5Ah
      je END_of_MCB

      mov BX, ES:[3]
      mov AX, ES
      add AX, BX
      inc AX
      mov ES, AX
      pop CX
      inc CL
      jmp loop_for_MCB
END_of_MCB:
      pop SI
      pop ES
      pop DX
      pop CX
      pop BX
      pop AX
      ret
MCB ENDP
BEGIN:
      call AM
      call EM
      call MCB
      xor AL, AL
      mov AH, 4Ch
      int 21h
```

```
TESTPC ENDS
     END START
```

lb3_part2.asm:

```
TESTPC SEGMENT
     ASSUME CS:TESTPC, DS:TESTPC
     ORG  100H
START:    jmp BEGIN

av_mem_print db 'available memory:          ', 0DH, 0AH, '$'
ext_mem_print db 'extended memory:           ', 0DH, 0AH, '$'
MCB_print db 'MCB              adress:       , PCP :             ,
size:      , SC/SD:          ', 0DH, '$'
TETR_TO_HEX PROC
     and AL, 0Fh
     cmp AL, 09
     jbe NEXT
     add AL, 07
NEXT: add AL, 30h
     ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC
; б а й т  в  AL  п е р е в о д и т с я  в  д в а  с и м в о л а  16-г о  ч и
с л а  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 с т а р ш а я  ц и ф р а
     pop CX           ; в AH м л а д ш а я
     ret
BYTE_TO_HEX ENDP

WRD_TO_HEX PROC
```

```
      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
      push AX
      push CX
      push DX
      xor AH, AH
      xor DX, DX
      mov CX, 10
loop_bd: div CX
      or DL, 30h
      mov [SI], DL
      dec SI
      xor DX, DX
      cmp AX, 10
      jae loop_bd
      cmp AL, 00h
      je end_1
      or AL, 30h
      mov [SI], AL
end_1:
      pop DX
      pop CX
      pop AX
      ret
```

```
BYTE_TO_DEC ENDP

WORD_TO_DEC PROC
      push AX
      push BX
      push DX
      push CX
      push SI
      mov BX, 10h
      mul BX
      mov BX, 0Ah
segmentation:
      div BX
      or DX, 30h
      mov [SI], DL
      dec SI
      xor DX, DX
      cmp AX, 0h
      jne segmentation
      pop SI
      pop CX
      pop DX
      pop BX
      pop AX
      ret
WORD_TO_DEC ENDP

print PROC
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
print ENDP
print_sym PROC
      push AX
      mov AH, 02h
      int 21h
      pop AX
```

```
      ret
print_sym ENDP

AM PROC
      push AX
      push BX
      push SI
      xor AX, AX
      mov AH, 4Ah
      mov BX, 0FFFFh
      int 21h
      mov AX, BX
      mov SI, offset av_mem_print
      add SI, 25
      call WORD_TO_DEC
      mov DX, offset av_mem_print
      call print
      pop SI
      pop BX
      pop AX
      ret
AM ENDP

EM PROC
      push AX
      push BX
      push SI
      xor AX, AX
      mov AL, 30h
      out 70h, AL
      in AL, 71h
      mov BL, AL
      mov AL, 31h
      out 70h, AL
      in AL, 71h
      mov AH, AL
      mov AL, BL
      mov SI, offset ext_mem_print
      add SI, 25
```

```
        call WORD_TO_DEC
        mov DX, offset ext_mem_print
        call print
        pop SI
        pop BX
        pop AX
        ret
EM ENDP
MCB PROC
        push AX
        push BX
        push CX
        push DX
        push ES
        push SI
        xor AX, AX
        mov AH, 52h
        int 21h
        mov AX, ES:[BX-2]
        mov ES, AX
        mov CL, 1
loop_for_MCB:
        mov AL, CL
        mov SI, offset MCB_print
        add SI, 9
        call BYTE_TO_DEC
        mov AX, ES
        mov DI, offset MCB_print
        add DI, 27
        call WRD_TO_HEX
        mov AX, ES:[01h]
        mov DI, offset MCB_print
        add DI, 46
        call WRD_TO_HEX
        mov AX, ES:[03h]
        add SI, 52
        call WORD_TO_DEC
        mov BX, 8
        push CX
```

```
      mov CX, 7
      add SI, 11
loop_sc_or_sd:
      mov DX, ES:[BX]
      mov DS:[SI], DX
      inc BX
      inc SI
      loop loop_sc_or_sd
      mov DX, offset MCB_print
      call print
      mov AH, ES:[0]
      cmp AH, 5Ah
      je END_of_MCB

      mov BX, ES:[3]
      mov AX, ES
      add AX, BX
      inc AX
      mov ES, AX
      pop CX
      inc CL
      jmp loop_for_MCB
END_of_MCB:
      pop SI
      pop ES
      pop DX
      pop CX
      pop BX
      pop AX
      ret
MCB ENDP

memory_free PROC
      push AX
      push BX
      push DX
      lea AX, the_end
      mov BX, 10h
      xor DX, DX
```

```
        div BX
        inc AX
        mov BX, AX
        xor AX, AX
        mov AH, 4Ah
        int 21h
        pop DX
        pop BX
        pop AX
        ret
    memory_free ENDP

    BEGIN:
        call AM
        call memory_free
        call EM
        call MCB
        xor AL, AL
        mov AH, 4Ch
        int 21h
    the_end:
    TESTPC ENDS
        END START
```

lb3_part3.asm:

```
    TESTPC SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC
        ORG  100H
    START:    jmp BEGIN

    av_mem_print db 'available memory:           ', 0DH, 0AH, '$'
    ext_mem_print db 'extended memory:            ', 0DH, 0AH, '$'
    MCB_print db 'MCB               adress:       , PCP :             ,
size:      , SC/SD:           ', 0DH, '$'
    ERROR db 'Not getting extra memory     ', 0DH, 0AH, '$'
    happy_end db 'Getting extr memory', 0DH, 0AH, '$'

    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
; б а й т  в AL  п е р е в о д и т с я  в  д в а  с и м в о л а  16-г о  ч и с л а  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 с т а р ш а я  ц и ф р а
        pop CX           ; в AH м л а д ш а я
        ret
BYTE_TO_HEX ENDP

WRD_TO_HEX PROC
        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
      push AX
      push CX
      push DX
      xor AH, AH
      xor DX, DX
      mov CX, 10
loop_bd: div CX
      or DL, 30h
      mov [SI], DL
      dec SI
      xor DX, DX
      cmp AX, 10
      jae loop_bd
      cmp AL, 00h
      je end_1
      or AL, 30h
      mov [SI], AL
end_1:
      pop DX
      pop CX
      pop AX
      ret
BYTE_TO_DEC ENDP

WORD_TO_DEC PROC
      push AX
      push BX
      push DX
      push CX
      push SI
      mov BX, 10h
      mul BX
      mov BX, 0Ah
segmentation:
      div BX
      or DX, 30h
      mov [SI], DL
      dec SI
```

```
      xor DX, DX
      cmp AX, 0h
      jne segmentation
      pop SI
      pop CX
      pop DX
      pop BX
      pop AX
      ret
WORD_TO_DEC ENDP

print PROC
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
print ENDP
print_sym PROC
      push AX
      mov AH, 02h
      int 21h
      pop AX
      ret
print_sym ENDP

AM PROC
      push AX
      push BX
      push SI
      xor AX, AX
      mov AH, 4Ah
      mov BX, 0FFFFh
      int 21h
      mov AX, BX
      mov SI, offset av_mem_print
      add SI, 25
      call WORD_TO_DEC
      mov DX, offset av_mem_print
```

```
      call print
      pop SI
      pop BX
      pop AX
      ret
AM ENDP

EM PROC
      push AX
      push BX
      push SI
      xor AX, AX
      mov AL, 30h
      out 70h, AL
      in AL, 71h
      mov BL, AL
      mov AL, 31h
      out 70h, AL
      in AL, 71h
      mov AH, AL
      mov AL, BL
      mov SI, offset ext_mem_print
      add SI, 25
      call WORD_TO_DEC
      mov DX, offset ext_mem_print
      call print
      pop SI
      pop BX
      pop AX
      ret
EM ENDP
MCB PROC
      push AX
      push BX
      push CX
      push DX
      push ES
      push SI
      xor AX, AX
```

```
        mov AH, 52h
        int 21h
        mov AX, ES:[BX-2]
        mov ES, AX
        mov CL, 1
loop_for_MCB:
        mov AL, CL
        mov SI, offset MCB_print
        add SI, 9
        call BYTE_TO_DEC
        mov AX, ES
        mov DI, offset MCB_print
        add DI, 27
        call WRD_TO_HEX
        mov AX, ES:[01h]
        mov DI, offset MCB_print
        add DI, 46
        call WRD_TO_HEX
        mov AX, ES:[03h]
        add SI, 52
        call WORD_TO_DEC
        mov BX, 8
        push CX
        mov CX, 7
        add SI, 11
loop_sc_or_sd:
        mov DX, ES:[BX]
        mov DS:[SI], DX
        inc BX
        inc SI
        loop loop_sc_or_sd
        mov DX, offset MCB_print
        call print
        mov AH, ES:[0]
        cmp AH, 5Ah
        je END_of_MCB

        mov BX, ES:[3]
        mov AX, ES
```

```
      add AX, BX
      inc AX
      mov ES, AX
      pop CX
      inc CL
      jmp loop_for_MCB
END_of_MCB:
      pop SI
      pop ES
      pop DX
      pop CX
      pop BX
      pop AX
      ret
MCB ENDP

memory_free PROC
      push AX
      push BX
      push DX
      lea AX, the_end
      mov BX, 10h
      xor DX, DX
      div BX
      inc AX
      mov BX, AX
      xor AX, AX
      mov AH, 4Ah
      int 21h
      pop DX
      pop BX
      pop AX
      ret
memory_free ENDP

memory2 PROC
      push AX
      push BX
      push DX
```

```
      mov BX, 1000h
      xor AX, AX
      mov AH, 48h
      int 21h
      jc for_error
      mov DX, offset happy_end
      call PRINT
      jmp endg

for_error:
      mov DX, offset ERROR
      call print
      jmp endg
endg:
      pop DX
      pop BX
      pop AX
      ret
memory2 ENDP

BEGIN:
      call AM
      call EM
      call memory_free
      call memory2
      call MCB
      xor AL, AL
      mov AH, 4Ch
      int 21h
the_end:
TESTPC ENDS
      END START
```

lb3_part4.asm:

```
TESTPC SEGMENT
      ASSUME CS:TESTPC, DS:TESTPC
      ORG  100H
START:     jmp BEGIN

av_mem_print db 'available memory:           ', 0DH, 0AH, '$'
```

```
      ext_mem_print db 'extended memory:            ', 0DH, 0AH, '$'
      MCB_print db 'MCB              adress:       , PCP :             ,
size:      , SC/SD:           ', 0DH, '$'
      ERROR db 'Not getting extra memory     ', 0DH, 0AH, '$'
      happy_end db 'Getting extr memory', 0DH, 0AH, '$'

      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
      ; б а й т  в  AL  п е р е в о д и т с я  в  д в а  с и м в о л а  16-г о  ч и
с л а  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 с т а р ш а я  ц и ф р а
            pop CX           ; в AH м л а д ш а я
            ret
      BYTE_TO_HEX ENDP

      WRD_TO_HEX PROC
            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
      push AX
      push CX
      push DX
      xor AH, AH
      xor DX, DX
      mov CX, 10
loop_bd: div CX
      or DL, 30h
      mov [SI], DL
      dec SI
      xor DX, DX
      cmp AX, 10
      jae loop_bd
      cmp AL, 00h
      je end_1
      or AL, 30h
      mov [SI], AL
end_1:
      pop DX
      pop CX
      pop AX
      ret
BYTE_TO_DEC ENDP

WORD_TO_DEC PROC
      push AX
      push BX
      push DX
      push CX
      push SI
```

```
      mov BX, 10h
      mul BX
      mov BX, 0Ah
segmentation:
      div BX
      or DX, 30h
      mov [SI], DL
      dec SI
      xor DX, DX
      cmp AX, 0h
      jne segmentation
      pop SI
      pop CX
      pop DX
      pop BX
      pop AX
      ret
WORD_TO_DEC ENDP

print PROC near
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
print ENDP
print_sym PROC near
      push AX
      mov AH, 02h
      int 21h
      pop AX
      ret
print_sym ENDP

AM PROC
      push AX
      push BX
      push SI
      xor AX, AX
```

```
      mov AH, 4Ah
      mov BX, 0FFFFh
      int 21h
      mov AX, BX
      mov SI, offset av_mem_print
      add SI, 25
      call WORD_TO_DEC
      mov DX, offset av_mem_print
      call print
      pop SI
      pop BX
      pop AX
      ret
AM ENDP

EM PROC
      push AX
      push BX
      push SI
      xor AX, AX
      mov AL, 30h
      out 70h, AL
      in AL, 71h
      mov BL, AL
      mov AL, 31h
      out 70h, AL
      in AL, 71h
      mov AH, AL
      mov AL, BL
      mov SI, offset ext_mem_print
      add SI, 25
      call WORD_TO_DEC
      mov DX, offset ext_mem_print
      call print
      pop SI
      pop BX
      pop AX
      ret
EM ENDP
```

```
MCB PROC
      push AX
      push BX
      push CX
      push DX
      push ES
      push SI
      xor AX, AX
      mov AH, 52h
      int 21h
      mov AX, ES:[BX-2]
      mov ES, AX
      mov CL, 1
loop_for_MCB:
      mov AL, CL
      mov SI, offset MCB_print
      add SI, 9
      call BYTE_TO_DEC
      mov AX, ES
      mov DI, offset MCB_print
      add DI, 27
      call WRD_TO_HEX
      mov AX, ES:[01h]
      mov DI, offset MCB_print
      add DI, 46
      call WRD_TO_HEX
      mov AX, ES:[03h]
      add SI, 52
      call WORD_TO_DEC
      mov BX, 8
      push CX
      mov CX, 7
      add SI, 11
loop_sc_or_sd:
      mov DX, ES:[BX]
      mov DS:[SI], DX
      inc BX
      inc SI
      loop loop_sc_or_sd
```

```
      mov DX, offset MCB_print
      call print
      mov AH, ES:[0]
      cmp AH, 5Ah
      je END_of_MCB

      mov BX, ES:[3]
      mov AX, ES
      add AX, BX
      inc AX
      mov ES, AX
      pop CX
      inc CL
      jmp loop_for_MCB
END_of_MCB:
      pop SI
      pop ES
      pop DX
      pop CX
      pop BX
      pop AX
      ret
MCB ENDP

memory_free PROC
      push AX
      push BX
      push DX
      lea AX, the_end
      mov BX, 10h
      xor DX, DX
      div BX
      inc AX
      mov BX, AX
      xor AX, AX
      mov AH, 4Ah
      int 21h
      pop DX
      pop BX
```

```
      pop AX
      ret
memory_free ENDP

memory2 PROC
      push AX
      push BX
      push DX
      mov BX, 1000h
      xor AX, AX
      mov AH, 48h
      int 21h
      jc for_error
      mov DX, offset happy_end
      call PRINT
      jmp endg

for_error:
      mov DX, offset ERROR
      call print
      jmp endg
endg:
      pop DX
      pop BX
      pop AX
      ret
memory2 ENDP

BEGIN:
      call AM
      call EM
      call memory2
      call memory_free
      call MCB
      xor AL, AL
      mov AH, 4Ch
      int 21h
the_end:
TESTPC ENDS
```

```
END START
```