**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 _____ Афанасьев Н. С.

Преподаватели _____ 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 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 .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 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.

**Шаг 2.** Измените программу таким образом, чтобы она освобождала память, которую она не занимает. Для этого используйте функцию 4Ah 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.

**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.**

При работе были использованы/созданы следующие процедуры:

- TETR_TO_HEX, BYTE_TO_HEX, WRD_TO_HEX – процедуры, описанные в шаблоне, для перевода двоичных кодов в символы шестнадцатеричных чисел.
- PRINT – процедура для вывода строки, отступ на которую содержится в DX, на экран, используя функцию 09h прерывания 21h.
- PRINT_MEM – процедура для вывода количества доступной памяти с использованием функции 4Ah прерывания 21h и размера расширенной памяти, взятого из CMOS.
- PRINT_MCB – процедура для вывода цепочек блоков управления памятью (MCB). Первый адрес достаётся из списка списков (“List of Lists”) с использованием функции 52h прерывания 21h.
- FREE_UP_MEM – процедура для освобождения памяти, не используемой программой, с использованием функции 4Ah прерывания 21h.

- REQ_MEM – процедура для запроса 64Кб памяти для программы с помощью функции 48h прерывания 21h. Процедура сообщает об ошибке, если она возникает.

На первом шаге программа запускается без запроса и освобождения памяти (см. рис. 1).

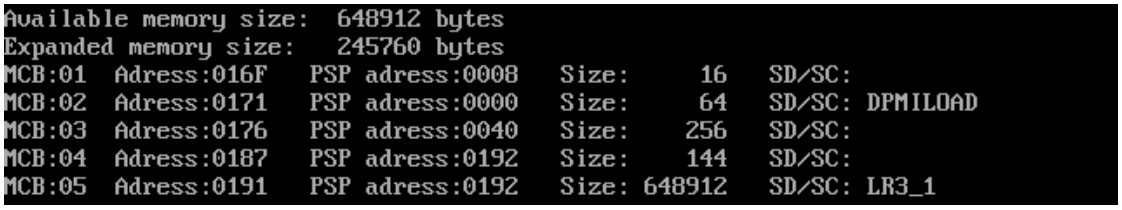


Рисунок 1 – Результат выполнения программы из первого шага

Из результатов можно видеть, что программа занимает блок памяти размером 648912 байт. Также можно заметить и другие участки различных размеров.

На следующем шаге программа освобождает неиспользуемую программой память (см. рис. 2).

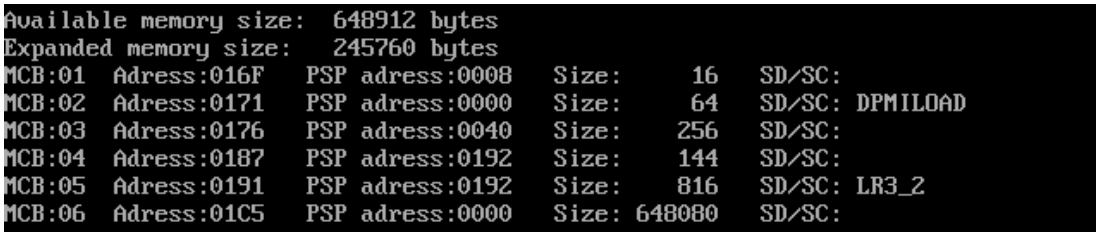


Рисунок 2 – Результат выполнения программы из второго шага

Как можно заметить, блок, в котором расположена программа, стал меньше, а вся неиспользуемая память была выделена в отдельный свободный блок.

На третьем шаге, после освобождения памяти, программа запрашивает дополнительные 64Кб памяти (см. рис. 3).

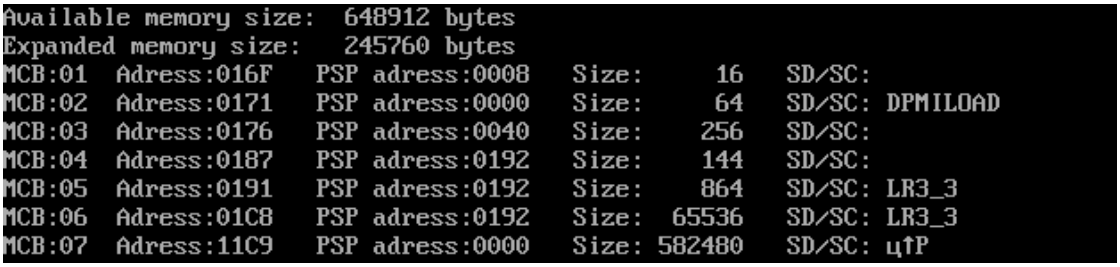

Рисунок 3 – Результат выполнения программы из третьего шага

Можно видеть, что сначала ненужная память была выделена в отдельный блок, а затем программа затребовала 64Кб, которые были выделены из этого блока в новый блок для программы.

На следующем шаге сначала производится запрос памяти и только потом освобождение ненужной памяти (см. рис. 4)

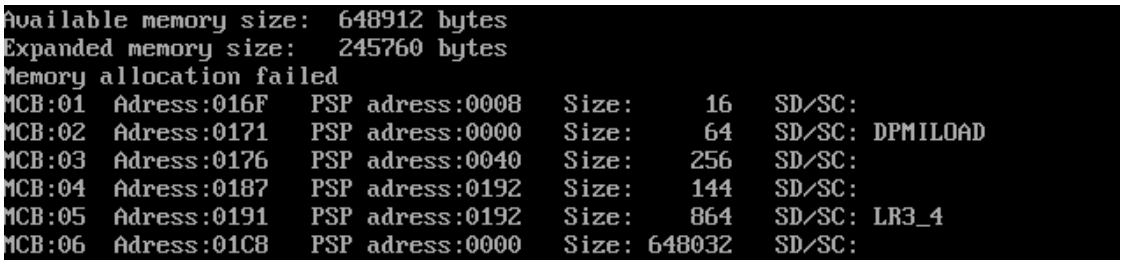


Рисунок 4 – Результат выполнения программы из четвёртого шага

Можно заметить ошибку, которая возникла из-за того, что на начальном этапе отсутствует блок со свободными 64Кб, поэтому выделить их никому не получается. Далее свободный блок создаётся высвобождением памяти из-под программы.

Программный код см. в Приложении А

**Вопросы.**

1) Что означает "доступный объем памяти"?

   – Доступный объём памяти – участок оперативной памяти, который может быть использован программой.

2) Где MCB блок Вашей программы в списке?

   – На каждом шаге MCB блоки программы в графе SD/SC помечены как “LR3_[номер шага]”.

3) Какой размер памяти занимает программа в каждом случае?

   – Размер считается как сумма размеров участков указанных в MCB программы: в первом случае – 648912 байт, во втором – 816 байта, в третьем – 66400 байт, в четвёртом – 864 байта;

**Выводы.**

В результате выполнения лабораторной работы была рассмотрена нестраничная память и способ управления динамическими разделами. Также была исследована организация управления памятью, структуры данных и работа функций управления памятью ядра операционной системы.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: lr3_1.asm

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

      FREE_MEM DB "Available memory size:         bytes",0DH,0AH,'$'
      EXP_MEM  DB "Expanded memory size:          bytes",0DH,0AH,'$'
      MCB    DB "MCB:0   Adress:      PSP adress:      Size:        SD/SC:
$"
      NEWLINE     DB 0DH,0AH,'$'

      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
            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
      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

SIZE_TO_DEC PROC NEAR
      push AX
      push BX
      push DX
      push SI
      add SI, 7
      mov BX,10h
      mul BX
      mov BX,10
      write_loop:
            div BX
            or dl,30h
            mov [SI], dl
            dec SI
            xor DX,DX
            cmp AX,0h
            jnz write_loop
      pop SI
      pop DX
      pop BX
      pop AX
      ret
SIZE_TO_DEC ENDP

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

PRINT_MEM PROC NEAR
      push AX
      push BX
      push DX
      push SI
```

```
      mov AH, 4Ah
      mov BX, 0FFFFh
      int 21h
      mov AX, BX
      mov DX, offset FREE_MEM
      mov SI, DX
      add SI, 22
      call SIZE_TO_DEC
      call PRINT

      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
      mov DX, offset EXP_MEM
      mov SI, DX
      add SI, 22
      call SIZE_TO_DEC
      call PRINT

      pop SI
      pop DX
      pop BX
      pop AX
      ret
PRINT_MEM ENDP

PRINT_MCB PROC NEAR
      push AX
      push BX
      push CX
      push DX
      push DI
      push SI

      mov AH, 52h
      int 21h
      mov AX, ES:[BX-2]
      mov ES, AX
      xor CX,CX

      MCB_block:
            inc CX
            mov AL, CL
            mov DX, offset MCB
            mov SI, DX
            add SI, 5
            call BYTE_TO_DEC

            mov AX, ES
            mov DI, SI
            add DI, 14
```

```
                call WRD_TO_HEX

                mov AX, ES:[1]
                add DI, 21
                call WRD_TO_HEX

                mov AX, ES:[3]
                mov SI, DI
                add SI, 11
                call SIZE_TO_DEC
                call PRINT

                xor DI,DI
                write_char:
                mov DL, ES:[DI+8]
                mov AH, 02h
                int 21h
                inc DI
                cmp DI, 8
                jl write_char
                mov DX, offset NEWLINE
                call PRINT

                mov AL, ES:[0]
                cmp AL, 4Dh
                jne exit
                mov BX, ES
                add BX, ES:[3]
                inc BX
                mov ES, BX
                jmp MCB_block
          exit:
          pop SI
          pop DI
          pop DX
          pop CX
          pop BX
          pop AX
          ret
     PRINT_MCB ENDP

     BEGIN:
          call PRINT_MEM
          call PRINT_MCB
          xor AL,AL
          mov AH, 4Ch
          int 21H
CODE ENDS
END START
```

Название файла: lr3_2.asm

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

     FREE_MEM DB "Available memory size:         bytes",0DH,0AH,'$'
```

```
      EXP_MEM  DB "Expanded memory size:          bytes",0DH,0AH,'$'
      MCB    DB "MCB:0   Adress:      PSP adress:      Size:        SD/SC:
$"
      NEWLINE     DB 0DH,0AH,'$'

      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
            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
            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

SIZE_TO_DEC PROC NEAR
      push AX
      push BX
      push DX
      push SI
      add SI, 7
      mov BX,10h
      mul BX
      mov BX,10
      write_loop:
            div BX
            or dl,30h
            mov [SI], dl
            dec SI
            xor DX,DX
            cmp AX,0h
            jnz write_loop
      pop SI
      pop DX
      pop BX
      pop AX
      ret
SIZE_TO_DEC ENDP

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

PRINT_MEM PROC NEAR
      push AX
      push BX
      push DX
      push SI

      mov AH, 4Ah
      mov BX, 0FFFFh
      int 21h
      mov AX, BX
      mov DX, offset FREE_MEM
      mov SI, DX
      add SI, 22
      call SIZE_TO_DEC
      call PRINT

      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
      mov DX, offset EXP_MEM
      mov SI, DX
      add SI, 22
      call SIZE_TO_DEC
      call PRINT

      pop SI
      pop DX
      pop BX
      pop AX
      ret
PRINT_MEM ENDP

PRINT_MCB PROC NEAR
      push AX
      push BX
      push CX
      push DX
      push DI
      push SI

      mov AH, 52h
      int 21h
      mov AX, ES:[BX-2]
      mov ES, AX
      xor CX,CX

      MCB_block:
            inc CX
            mov AL, CL
            mov DX, offset MCB
            mov SI, DX
            add SI, 5
            call BYTE_TO_DEC

            mov AX, ES
            mov DI, SI
            add DI, 14
            call WRD_TO_HEX

            mov AX, ES:[1]
            add DI, 21
            call WRD_TO_HEX

            mov AX, ES:[3]
            mov SI, DI
            add SI, 11
            call SIZE_TO_DEC
            call PRINT

            xor DI,DI
            write_char:
```

```
                mov DL, ES:[DI+8]
                mov AH, 02h
                int 21h
                inc DI
                cmp DI, 8
                jl write_char
                mov DX, offset NEWLINE
                call PRINT

                mov AL, ES:[0]
                cmp AL, 4Dh
                jne exit
                mov BX, ES
                add BX, ES:[3]
                inc BX
                mov ES, BX
                jmp MCB_block
          exit:
          pop SI
          pop DI
          pop DX
          pop CX
          pop BX
          pop AX
          ret
     PRINT_MCB ENDP

     FREE_UP_MEM PROC NEAR
          push AX
          push BX
          push DX

          mov AX, offset end_address
          mov BX, 10h
          xor DX,DX
          div BX
          add AX, 4
          mov BX, AX
          mov AH, 4Ah
          int 21h

          pop DX
          pop BX
          pop AX
          ret
     FREE_UP_MEM ENDP

     BEGIN:
          call PRINT_MEM
          call FREE_UP_MEM
          call PRINT_MCB
          xor AL,AL
          mov AH, 4Ch
          int 21H
end_address:
CODE ENDS
END START
```

Название файла: lr3_3.asm

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

      FREE_MEM DB "Available memory size:         bytes",0DH,0AH,'$'
      EXP_MEM  DB "Expanded memory size:          bytes",0DH,0AH,'$'
      MCB    DB "MCB:0   Adress:      PSP adress:      Size:        SD/SC:
$"
      MEM_FAIL DB "Memory allocation failed",0DH,0AH,'$'
      NEWLINE     DB 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
            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
            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

SIZE_TO_DEC PROC NEAR
      push AX
      push BX
      push DX
      push SI
      add SI, 7
      mov BX,10h
      mul BX
      mov BX,10
      write_loop:
            div BX
            or dl,30h
            mov [SI], dl
            dec SI
            xor DX,DX
            cmp AX,0h
            jnz write_loop
      pop SI
      pop DX
      pop BX
      pop AX
      ret
SIZE_TO_DEC ENDP

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

PRINT_MEM PROC NEAR
      push AX
      push BX
      push DX
      push SI

      mov AH, 4Ah
      mov BX, 0FFFFh
      int 21h
      mov AX, BX
```

```
      mov DX, offset FREE_MEM
      mov SI, DX
      add SI, 22
      call SIZE_TO_DEC
      call PRINT

      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
      mov DX, offset EXP_MEM
      mov SI, DX
      add SI, 22
      call SIZE_TO_DEC
      call PRINT

      pop SI
      pop DX
      pop BX
      pop AX
      ret
PRINT_MEM ENDP

PRINT_MCB PROC NEAR
      push AX
      push BX
      push CX
      push DX
      push DI
      push SI

      mov AH, 52h
      int 21h
      mov AX, ES:[BX-2]
      mov ES, AX
      xor CX,CX

      MCB_block:
            inc CX
            mov AL, CL
            mov DX, offset MCB
            mov SI, DX
            add SI, 5
            call BYTE_TO_DEC

            mov AX, ES
            mov DI, SI
            add DI, 14
            call WRD_TO_HEX

            mov AX, ES:[1]
            add DI, 21
            call WRD_TO_HEX
```

```
            mov AX, ES:[3]
            mov SI, DI
            add SI, 11
            call SIZE_TO_DEC
            call PRINT

            xor DI,DI
            write_char:
            mov DL, ES:[DI+8]
            mov AH, 02h
            int 21h
            inc DI
            cmp DI, 8
            jl write_char
            mov DX, offset NEWLINE
            call PRINT

            mov AL, ES:[0]
            cmp AL, 4Dh
            jne exit
            mov BX, ES
            add BX, ES:[3]
            inc BX
            mov ES, BX
            jmp MCB_block
      exit:
      pop SI
      pop DI
      pop DX
      pop CX
      pop BX
      pop AX
      ret
PRINT_MCB ENDP

FREE_UP_MEM PROC NEAR
      push AX
      push BX
      push DX

      mov AX, offset end_address
      mov BX, 10h
      xor DX,DX
      div BX
      add AX, 4
      mov BX, AX
      mov AH, 4Ah
      int 21h

      pop DX
      pop BX
      pop AX
      ret
FREE_UP_MEM ENDP

REQ_MEM PROC near
      push AX
```

```
            push BX
            push DX

            mov BX, 1000h
            mov AH, 48h
            int 21h
            jnc exit_
            mov DX, offset MEM_FAIL
            call PRINT

            exit_:
            pop DX
            pop BX
            pop AX
            ret
      REQ_MEM ENDP

      BEGIN:
            call PRINT_MEM
            call FREE_UP_MEM
            call REQ_MEM
            call PRINT_MCB
            xor AL,AL
            mov AH, 4Ch
            int 21H
end_address:
CODE ENDS
END START
```

Название файла: lr3_4.asm

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

      FREE_MEM DB "Available memory size:         bytes",0DH,0AH,'$'
      EXP_MEM  DB "Expanded memory size:          bytes",0DH,0AH,'$'
      MCB    DB "MCB:0   Adress:      PSP adress:      Size:        SD/SC:
$"
      MEM_FAIL DB "Memory allocation failed",0DH,0AH,'$'
      NEWLINE     DB 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
            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
      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

SIZE_TO_DEC PROC NEAR
      push AX
      push BX
      push DX
      push SI
      add SI, 7
      mov BX,10h
      mul BX
      mov BX,10
      write_loop:
            div BX
            or dl,30h
            mov [SI], dl
```

```
            dec SI
            xor DX,DX
            cmp AX,0h
            jnz write_loop
      pop SI
      pop DX
      pop BX
      pop AX
      ret
SIZE_TO_DEC ENDP

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

PRINT_MEM PROC NEAR
      push AX
      push BX
      push DX
      push SI

      mov AH, 4Ah
      mov BX, 0FFFFh
      int 21h
      mov AX, BX
      mov DX, offset FREE_MEM
      mov SI, DX
      add SI, 22
      call SIZE_TO_DEC
      call PRINT

      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
      mov DX, offset EXP_MEM
      mov SI, DX
      add SI, 22
      call SIZE_TO_DEC
      call PRINT

      pop SI
      pop DX
      pop BX
      pop AX
      ret
PRINT_MEM ENDP

PRINT_MCB PROC NEAR
```

```
push AX
push BX
push CX
push DX
push DI
push SI

mov AH, 52h
int 21h
mov AX, ES:[BX-2]
mov ES, AX
xor CX,CX

MCB_block:
     inc CX
     mov AL, CL
     mov DX, offset MCB
     mov SI, DX
     add SI, 5
     call BYTE_TO_DEC

     mov AX, ES
     mov DI, SI
     add DI, 14
     call WRD_TO_HEX

     mov AX, ES:[1]
     add DI, 21
     call WRD_TO_HEX

     mov AX, ES:[3]
     mov SI, DI
     add SI, 11
     call SIZE_TO_DEC
     call PRINT

     xor DI,DI
     write_char:
     mov DL, ES:[DI+8]
     mov AH, 02h
     int 21h
     inc DI
     cmp DI, 8
     jl write_char
     mov DX, offset NEWLINE
     call PRINT

     mov AL, ES:[0]
     cmp AL, 4Dh
     jne exit
     mov BX, ES
     add BX, ES:[3]
     inc BX
     mov ES, BX
     jmp MCB_block
exit:
pop SI
pop DI
```

```
            pop DX
            pop CX
            pop BX
            pop AX
            ret
      PRINT_MCB ENDP

      FREE_UP_MEM PROC NEAR
            push AX
            push BX
            push DX

            mov AX, offset end_address
            mov BX, 10h
            xor DX,DX
            div BX
            add AX, 4
            mov BX, AX
            mov AH, 4Ah
            int 21h

            pop DX
            pop BX
            pop AX
            ret
      FREE_UP_MEM ENDP

      REQ_MEM PROC near
            push AX
            push BX
            push DX

            mov BX, 1000h
            mov AH, 48h
            int 21h
            jnc exit_
            mov DX, offset MEM_FAIL
            call PRINT

            exit_:
            pop DX
            pop BX
            pop AX
            ret
      REQ_MEM ENDP

      BEGIN:
            call PRINT_MEM
            call REQ_MEM
            call FREE_UP_MEM
            call PRINT_MCB
            xor AL,AL
            mov AH, 4Ch
            int 21H
end_address:
CODE ENDS
END START
```