+---------------+
¦ ASSEMBLY XVII ¦
+---------------+

Eis o arquivo .ASM com as rotinas para manipulação da memória expandida:

```
IDEAL
MODEL LARGE,PASCAL
LOCALS
JUMPS

GLOBAL  emmGetVersion : PROC
GLOBAL  emmGetPageFrameSegment : PROC
GLOBAL  emmGetAvailablePages : PROC
GLOBAL  emmAllocPages : PROC
GLOBAL  emmFreePages : PROC
GLOBAL  emmMapPage : PROC
GLOBAL  emmGetError : PROC

DATASEG

emmVersion  dw  0
emmError    db  0       ; Nenhum erro ainda... :)

CODESEG

; Obtém a versão do EMM.
; Devolve no formato 0x0X0Y (onde X é versão e Y revisão).
; Protótipo em C:
;   unsigned pascal emmGetVersion(void);
PROC    emmGetVersion
    mov     [emmError],0    ; Inicializa flag de erro...
    mov     ah,46h
    int     67h             ; Invoca o EMM
    or      ah,ah           ; Testa o sucesso da funçäo...
    jz      @@no_error
    mov     [emmError],ah   ; Poe erro no flag...
    mov     ax,-1           ; ... e retorna != 0.
    jmp     @@done
    mov     ah,al           ; Prepara formato da versäo.
    and     ax,111100001111b ; A função 46h do EMM devolve
    mov     [emmVersion],ax  ; no formato BCD... por isso
@@done:                      ; precisamos formatar...
    ret
ENDP

; Função: Obtém o segmento do Page Frame.
; Protótipo em C:
;   unsigned pascal emmGetPageFrameSegment(void);
PROC    emmGetPageFrameSegment
    mov     ah,41h      ; Usa a função 41h do EMM
    int     67h         ; Chama o EMM
    mov     ax,bx       ; Poe o segmento em AX
                        ; Função 41h coloca o segmento do
                        ; "Page Frame" em BX.
    ret
ENDP

; Função: Obtém o número de páginas disponíveis na memória.
; Protótipo em C:
;   unsigned pascal emmGetAvailablePages(void);
; Obs:
```

```
;   Não verifica a ocorrencia de erros... modifique se quiser
PROC    emmGetAvailablePages
    mov     ah,42h
    int     67h     ; Invoca o EMM.
    mov     ax,bx   ; Poe páginas disponiveis em AX.
    ret
ENDP

; Aloca páginas e devolve handle.
; Protótipo em C:
;   int pascal emmGetAvailablePages(unsigned Pages);
; Obs: Devolve -1 se houve erro na alocaçao e seta
;      a variável emmError.
PROC    emmAllocPages
ARG     Pages:WORD
    mov     [emmError],0    ; Inicializa flag de erros...
    mov     bx,[Pages]      ; BX = número de páginas a alocar
    mov     ah,43h
    int     67h             ; Invoca o EMM.
    or      ah,ah           ; Verifica erro do EMM.
    jz      @@no_error
    mov     [emmError],ah   ; Poe erro na variável emmError
    mov     dx,-1
@@no_error:
    mov     ax,dx           ; retorna código de erro.
                            ; ou o handle.
    ret
ENDP

; Libera páginas alocadas.
; Protótipo em C:
;   void pascal emmFreePages(int handle);
; Obs: Não verifica erros... modifique se quiser...
PROC    emmFreePages
ARG     handle:WORD
    mov     dx,[handle]
    mov     ah,45h
    int     67h
    ret
ENDP

; Mapeia uma página no Page Frame.
; Protótipo em C:
;   int pascal emmMapPage(int handle,
;                         unsigned char pfPage,
;                         unsignec PageNbr);
; Onde: handle é o valor devolvido pela função de alocação de
;       páginas.
;       pfPage é o número da página do Page Frame (0 até 3).
;       PageNbr  é o número da página a ser colocada no
;       Page Frame (0 até máximo - 1).
; Devolve -1 se ocorreu erro e seta a variável emmError.
PROC    emmMapPage
ARG     handle:WORD, pfPage:BYTE, PageNbr:WORD
    mov     [emmError],0
    mov     ah,44h
    mov     al,[pfPage]
    mov     bx,[PageNbr]
    mov     dx,[handle]
    int     67h
    or      ah,ah
    jz      @@no_error
    mov     [emmError],ah
    mov     ah,-1
```

```
  @@no_error:
      mov     al,ah
      ret
  ENDP

  ; Retorna com o erro do EMM.
  ; Protótipo:
  ;   int pascal emmGetError(void);
  PROC    emmGetError
      mov     ax,[emmError]
      ret
  ENDP

  END
```

Esta é uma implementação simplificada, mas para nossos propósitos serve muito bem. Algumas considerações: A alocação de memória via EMM não é feita da mesma maneira que a função malloc() de C ou GetMem() do TURBO PASCAL. Não é devolvido nenhum pointer. Isto se torna óbvio a partir do momento que entendemos como funciona o EMM: Toda a manipulação de bancos de memória é feita de forma indireta pelo Page Frame. A função de alocação deve apenas devolver um handle para que possamos manipular as páginas alocadas. Entenda esse handle da mesma forma com que os arquivos são manipulados... Se quisermos usar um banco alocado precisamos informar ao EMM qual dos bancos queremos usar, fazendo isso via o handle devolvido pelo próprio EMM.

Suponha que queiramos alocar 128kb da memória expandida para o nosso programa. Precisamos alocar 8 páginas lógicas (8 * 16k = 128k). Chamariamos a função emmAllocPages() em C da seguinte forma:

```
  #include <conio.h>
  #include <stdlib.h>

  int emm_handle;

  void f(void)
  {
      /* ... */
      if ((emm_handle = emmAllocPages(8)) == -1) {
          cprintf("EMM ERROR #%d\r\n", emmGetError());
          exit(1);
      }
      /* ... */
  }
```

Na função emmAllocPages() optei por devolver -1 para indicar o insucesso da função... Você pode arrumar um esquema diferente para chegar isso (por exemplo, checando a variável emmError após a chamada a função!).

Well... Temos 8 páginas lógicas disponíveis. E agora?... As 8 páginas estão sempre numeradas de 0 até o máximo - 1. No nosso caso teremos as páginas 0 até 7 disponíveis ao nosso programa. Lembre-se que cada uma tem apenas 16k de tamanho e que podem ser arranjadas de qq maneira q vc queira no Page Frame. Vamos usar as 4 páginas iniciais como exemplo... para isso precisamos mapea-las no Page Frame usando a função emmMapPage().

```
void f(void)
{
    int i;

    /* ... */
    for (i = 0; i < 4; i++)
        emmMapPage(emm_handle,i,i);
}
```

Depois deste pequeno loop sabemos que qualquer alteraçäo no conteúdo do Page Frame alterará as páginas que estäo mapeadas nele...:) Simples né? Só nos resta conhecer o endereço inicial do Page Frame:

```
#include <dos.h>

void far *PageFrameAddr;

void f(void)
{
    /* ... */
    PageFrameAddr = MK_FP(emmGetPageFrameSegment(), 0);
    /* ... */
}
```

Ao fim do uso da memória expandida precisamos dealocar o espaço previamente alocado... C e C++ dealocam automaticamente qualquer espaço alocado por malloc(), calloc() e funçöes afins... Näo é o caso de nossas rotinas acima... entäo acostume-se a manter a casa em ordem e usar a função emmFree() quando näo precisar mais das páginas alocadas.

Isso tudo näo funcionará se o EMM näo estiver instalado... No texto anterior mostrei a rotina para determinar a presença do EMM. E, no mesmo texto, apareceu a rotina emm_majorVer(). Eis a rotina abaixo:

```
int emm_majorVer(void)
{ return ((int)emmGetVersion() >> 8); }
```

See ya l8tr