



Tech Info Library

Pascal III: Accessing the extra memory (2 of 5)

Revised: 11/30/84
Security: Everyone

Pascal III: Accessing the extra memory (2 of 5)

```
=====

        .PROC   Allocate,4
FindSeg .EQU    41

SegBase .EQU    Temp1
Bank    .EQU    Temp2
SegNum  .EQU    Temp3
NumPages .EQU   Temp4

        EnterProc RetAddr

        PULL    SegBase
        PULL    Bank
        PULL    SegNum
        PULL    NumPages

        P_A_Word NumPages,Pages

        SOSCall Find_Seg,Param
        BEQ     $1          ; If it worked,great
        SOSCall Find_Seg,Param ; Else try again!
$1      BEQ     $2
        LDA     #0          ; 2 failures ==> return 0 pages
        STA     Pages
        STA     Pages+1

$2      LDA     SPage        ; Send back starting address
        LDY     #1          ; in high byte!!
        STA     (SegBase),Y
        DEY
        TYA
        STA     (SegBase),Y

        A_P_Word Pages,NumPages
        A_P_Byte SBank,Bank
        A_P_Byte Seg, SegNum
```

```

        Return  RetAddr

Param    .BYTE    06
SrchrMde .BYTE    02      ; Can cross more than one boundary
SegId    .BYTE    30      ; First user segment?
Pages    .WORD
SBank    .BYTE
SPage    .BYTE
         .WORD
Seg       .BYTE

RetAddr  .WORD

        .PROC DeAllocate,1
        ; PROCEDURE DeAllocate(SegNum:INTEGER); EXTERNAL;
Release_Seg .EQU    45
        EnterProc    RetAddr
        PULL         SegNum

        LDA          SegNum
        BEQ          Ret      ; Don't release all segs!

        SOSCALL      Release_Seg,Param
Ret      Return      RetAddr

Param    .BYTE    1
SegNum    .WORD      ; Only first byte used by SOS, but PULL
              ; needs 2
RetAddr    .WORD

        .END

```

```

=====
; This file contains an assembly language procedure which moves
; bytes from one bank to another. Use this from Pascal to get
; access to the rest of the bank space in larger machines.
; This can allow you to do SOS Request-Seg calls and use
; that area for data storage for later use or to pass to an
; assembly procedure. This procedure does two slightly difficult
; things :
;
; 1. it avoids holes in bank memory map, and
; 2. it switches banks on both source and destination at the
;    right times.
;
; Pascal declaration is:
;   PROCEDURE FetchBytes(SrcBank:INTEGER;   Source:INTEGER;
;   ;                               DstBank:INTEGER;   Dest:INTEGER;
;   ;                               PageCount:INTEGER;  Count:INTEGER
;   ;                               ); EXTERNAL;
;
; The procedure will copy 256*PageCount+Count bytes from
; Source (a bank pointer to bank pair starting with SrcBank) to

```

```

; Dest (a bank pointer to bank pair starting with DstBank);
; A SrcBank or DstBank of -1 means "in Pascal's bank pair"
; Thus FetchBytes(-1,Atsign(S),-1,Atsign(D),0,C) ==
;   MoveLeft(S,D,C)
;
; RESTRICTIONS:
;   1. Equivalent to a moveleft (ascending transfer)
;   2. Cannot reach top and bottom (system) banks,
;       and "real" addresses must be offset
;       by $2000 to render bank addresses
;
; ALGORITHM:
;   1.  If we're moving  X*65535 +Y*256 + Z bytes, then
;   2.  First move Y pages to clear middle byte of Count
;   3.  Then use middle byte to move X*256 pages,
;   4.  Finally, move Z bytes in last page.
; =====
; MACRO Defs
;   .MACRO PULL
;     PLA
;     STA    %1
;     PLA
;     STA    %1+1
;   .ENDM
;
;   .MACRO PUSH
;     LDA    %1+1
;     PHA
;     LDA    %1
;     PHA
;   .ENDM

```

Apple Tech Notes

Tech Info Library Article Number:640