

Developing Applications with Microsoft Works

Introduction

Developing applications on the Tandy 600 can be a challenge. The primary storage device is RAM-based, and therefore is extremely fragile in the face of run-away code. In addition, the operating system does not have offer an interactive 'DEBUG'-like facility. These factors make application development on the Tandy 600 difficult.

Microsoft Works solves this dilemma by allowing the application developer to create and maintain programs on a MS-DOS based computer. Tools are provided to create application- and ROM-image files. A simulator of the operating system, Hand-Held Operating System (HHOS) allows the developer to test their code prior to installation on the Tandy 600. In conjunction with Microsoft's SYMDEB, interactive 'debugging' may be done from within the simulator.

Requirements

Microsoft Works requires that the developer has the following:

- Microsoft Works Disk (Vendor Development Services Internal Document # VDS-263901-01)
- Microsoft Macro-Assembler (MASM) version 4.0.
- MS-DOS based computer with 512K RAM (not compatible with Tandy 2000).

Developing Applications

In order to develop applications, follow the guidelines set forth in the Tandy 600 Programmer's Reference Guide and BIOS Specifications, pages 111-116. Convert the '.EXE' file created by MASM with the EXECNV utility. The resultant '.HHX' may be then transported to the Tandy 600 through RS-232C connections. 'TELCOM', provided with the Tandy 600, provides XMODEM capabilities. The resultant '.HHX' may also be tested on the HHOS simulator. This involves creating a ROM image file.

Developing ROMS

The Tandy 600 hardware is designed to support a secondary ROM or EEPROM. During boot-up procedure, if the ROM is determined to have been installed, HHOS reads the contents of the ROM, and places the names of the application(s) found therein on the menu bar. The BLDROM utility creates the ROM image in a format acceptable to HHOS. BLDROM determines with applications to place on the ROM image by reading the file DEBUG.GEN. As provided, DEBUG.GEN is set up to create a ROM image file containing overlays of HHOS, Microsoft Word, and the Pop-Up Calculator. To have BLDROM create a ROM image file with your application(s), merely add the names of these to the DEBUG.GEN file. For example:

```
word.!30 word.!30  
dump600.hhx dump600.!92  
dir600.hhx dir600.!93
```

Note: If you are creating a ROM image file intended to be burned into the secondary ROM, do not have BLDROM copy the HHOS overlays, Word, or the Calculator. These files are present and available in the primary ROM of the Tandy 600.

Using The Simulator

The premise behind the Simulator is to provide an environment similar to HHOS, operating under MS-DOS. To do this, first develop your application with MASM. Convert the '.EXE' file to '.HHX' format. Create a ROM image file, with the operating system overlays, Word, the Calculator, and however many of your applications. Then, type:

```
HHSIM DEBUG.PRM
```

The top portion of the screen clears, and the simulation of the Tandy 600 begins. Note that the simulator does not emulate the BIOS calls for RS-232C. When referencing the Tandy 600's disk drive, the simulator will access drive A:. If you require referencing the Tandy 600's disk drive, use BLDROM to make a ROM image file of the data or programs files found on the disk drive, and copy that file, DEBUG.ROM, to drive A: prior to running the simulator.

The file, DEBUG.PRM, contains parameters about the simulation. The number of rows and columns on the screen can be altered, the name of the ROM image file may be specified, as well as the RAM file. The RAM file has the contents of the directory system. The RAMSIZ variable allows the user to specify the size of the ROM or EEPROM being used.

In running the simulator, note the the F1 key is used as the 'Label' key. Alternate-F10 exits the simulator back to MS-DOS.

Interactive Debugging

In order to use SYMDEB in conjunction with the simulator, first create a '.MAP' file of your application with MASM. MAPSYM converts this file to '.SYM' format. Then, type:

```
HHDEBUG {name}.SYM
```

And follow the instructions outlined in the Microsoft Works documentation.

Application Structure Overview

Tandy 600 applications files have a fixed header structure, which is used by HHOS to determine the amount of memory required by the program. Information is in the header to indicate the size of the data region, stack, and code. As designed, all strings, data tables, etc. intended to be in the data segment are contained in the code segment of the application. During load time, HHOS allocates a data segment in memory, and moves these elements away from the code segment. Stack is set up in the data segment; the user specifies where the stack pointer offset will be. So, immediately following the loading of an application, code and data segments may look as follows:

Code Segment

```
+-----+
| Header |
|        |
| Code   |
|        |
|-----|
| data tables, strings, |
| etc., TO BE MOVED TO |
| Data Segment from    |
| Code Segment.        |
|                       |
+-----+
```

Data Segment (AMI)

```
+-----+
| Stack area |
|            |
|=====  
| Data tables, strings, |
| etc., THAT WAS MOVED  |
| TO Data Segment from  |
| Code Segment.         |
|                        |
| extra space for uninit |
| tables, etc.          |
|                       |
+-----+
```

Keep in mind that this situation is changeable by your application following load procedures. You may change memory allocation to fit the application.

Here is a sample header:

```

;      sample header
dw      5f10h      ; aplchk
dw      9000h      ; amicod
db      0          ; ldrld; 8086 native code
db      'V'        ; amivis; AMI visible from menu
db      '.'        ; amityp; owned files are
                        AMI's; '-' for data
db      'WRK'      ; amiext; extension for AMI
                        files
dw      begin      ; aplip; set IP to origin of
                        code
dw      stack      ; aplsp; set SP to top of stack
                        area in AMI
dw      cs_length  ; aplsiz; size of code segment
dw      data_start ; datpos; beginning data area
                        WITHIN CS segment
dw      data_final ; datloc; beginning data WITHIN
                        AMI; to be...
dw      data_length ; datlen; ...moved from CS
                        segment, for length of data
```

The Data Segment, referred to as an AMI, is an actual data file created by the application. The file is named 'WORK.', with the extension indicated by the 'AMIEXT' field in the header. It is extremely important to assign unique extensions for your applications. HHOS determines ownership of files from the extension when displaying the menu bar.

If your AMI file exists the subsequent time your application is run, the previous values within the AMI are maintained. That is, the previous contents of your data and stack segment are intact. This is important to note, as HHOS is designed to allow the user to leave an application for another, and return back to the original intact (e.g., the Pop-Up Calculator).

What controls whether your AMI file is still resident as file is how you exit from your application. Should you wish to exit the application with AMI intact, your code segment should read:

```
mov     al,0                ; exit application with AMI
                           ; intact
mov     ah,Exit             ; ah = HH05 function code to
                           ; exit
int     42h                 ; call HH05
```

In order to have the AMI killed on exit, change the first line to read:

```
mov     al,-1               ; exit application, killing AMI
```

When exiting your application to call another (say, the Pop-Up Calculator), use the first method shown above. To exit your application back to the menu bar, use the second.

When writing your application, it is important to note that at any time, the user may press a function key to either exit your application, or call up another. The keyboard routines must be designed to check for the proper scan codes, and act upon them immediately. The sample programs shown below illustrate how to go about this.

Sample Programs

The two sample programs illustrate programming style and structure for the Tandy 600. The first demonstrates how to re-use the AMI file; when the application is running, it prompts the user for a string of characters. When run again, the original string entered is displayed. The AMI file is then killed.

The second program illustrates keyboard scanning techniques, as well as file management. These programs demonstrate different ways to handle data placement in the code segment.

M600F.ASM

page 50,132
title Sample Tandy 600 Program

```
; Sample program prompts the user to type information
; (ConStringInput emulated). Text is displayed back to the
; screen; system date and time are displayed. If AMI present
; from last time, the previous contents are displayed first. Also
; note the keyboard handler that takes care of user forking off
; to the pop-up calculator, etc.
; os_redraw: We'd have to buffer what is on screen at fork-time,
; to restore previous contents on exit. Not attempted, but working
; at the LCD level looks the path to follow...
;
; Written 4-10-86. (M600F.ASM)
```

; HHOS Functions

GetTime	equ	2ch	; get system time
GetDate	equ	2ah	; get system date
ConStringOutput	equ	09h	; display string to console
Exit	equ	4ch	; exit to dos
GetChar	equ	08h	; get char from keyboard w/o echo
PutChar	equ	02h	; write char to screen

; LCD Text Interrupt Functions

ClearScreen	equ	01h	; clear screen on lcd
GetCursor	equ	03h	; get cursor position
SetCursor	equ	02h	; set cursor position
DosInt	macro	function	
	mov	ah,function	; ah = function number
	int	42h	; go call dos
	endm		
LcdInt	macro	function	
	mov	ah,function	; ah = function number

```

int      51h                ; call BIOS
endm

code      SEGMENT public ?CODE?
          ASSUME cs:code,ds:code

; This is the program header
begin:
    dw      5f10h            ; application check
    dw      9004h            ; our application as code
    db      0                ; 8086 native code
    db      'V'              ; app is always visible
    db      '.'              ; owned files are AMIs
    db      'M6F'            ; extension (MUST BE UNIQUE!)
    dw      start            ; beginning of IP
    dw      0100h            ; place to put SP in AMI
    dw      aplsiz           ; size of application
    dw      begin_of_data    ; beginning of data
    dw      0102h            ; place to keep data
    dw      data_length      ; length of data area

start:
    LcdInt   ClearScreen     ; clear the screen

resume:
    mov      [cod],cs        ; save code segment
    mov      [dod],ds        ; save data segment
    mov      [eod],es        ; save extra segment
    mov      [sod],ss        ; save stack segment
    mov      ax,sp           ; stack stack offset
    mov      [spd],ax        ; save stack offset

; figure out where things are in memory

    mov      ax,[cod]        ; get code segment
    mov      si,offset cod$   ; si -> place to keep answer
    call     bin_hex          ; convert binary to ascii hex
    mov      ax,[dod]        ; get data segment
    mov      si,offset dod$   ; si -> place to put answer
    call     bin_hex          ; convert binary to ascii hex

```



```

mov     ax,[eod]                ; get extra segment
mov     si,offset eod$          ; si -> place to put answer
call    bin_hex                 ; do conversion
mov     ax,[sod]                ; get stack segment
mov     si,offset sod$          ; si -> answer
call    bin_hex                 ; do conversion
mov     ax,[spd]                ; get stack offset
mov     si,offset spd$          ; si -> answer
call    bin_hex                 ; do conversion

mov     dx,offset seg$          ; post answer to screen
DosInt  ConStringOutput         ; thusly

; post a "howdy" to the screen

mov     dx,offset hello$        ; ds:dx -> "hello"
DosInt  ConStringOutput         ; out to screen we go

; prompt users to type something in

mov     si,offset buffer$       ; ds:si -> user buffer
call    ConStringInput          ; await keyboard entry

; post a report to the users

push    si                      ; save this for a minute
mov     dx,offset report$       ; display header information
DosInt  ConStringOutput         ; post to screen

; prepare to display user's input to screen

pop     dx                      ; get offset back
inc     dx                      ; jump over OUR length byte
mov     si,dx                   ; move that into si
inc     dx                      ; jump over THEIR length byte
mov     bx,byte ptr [si]        ; get length of entered buffer
xor     bh,bh                   ; clean this up
mov     byte ptr [si+bx+1], '$' ; stash an end-of-string
DosInt  ConStringOutput         ; and display string

```

```

; get system date

DosInt  GetDate                                ; get system date

mov     si,offset date$                        ; si -> ascii buffer for date
mov     al,dh                                  ; convert month first
call    convert_single                         ; convert 2-digit ascii
mov     al,dl                                  ; convert day next
call    convert_single                         ; convert 2-digit ascii
call    convert_double                         ; convert year in cx, 4-digit ascii

; get system time

DosInt  GetTime                                ; get system time

mov     si,offset time$                        ; si -> ascii buffer for time
mov     al,ch                                  ; handle hour
call    convert_single                         ; do conversion
mov     al,cl                                  ; handle minutes
call    convert_single                         ; do conversion
mov     al,dh                                  ; handle seconds
call    convert_single                         ; do conversion
mov     al,dl                                  ; handle hundreds of seconds
call    convert_single                         ; do conversion

; post answer to screen

mov     dx,offset system_time$                 ; dx -> the time is
DosInt  ConStringOutput                         ; send to screen

; await keystroke and exit

DosInt  GetChar                                ; await character
mov     al,0                                    ; no error; but allow resumption
exit_ality equ     $-1
DosInt  Exit                                    ; exit to DOS

; we get here from second invocation

mov     al,-1                                  ; post a minus one

```

```
mov     byte ptr cs:[exit_ality],al      ; save into exit routine

LcdInt  ClearScreen                     ; clear screen
mov     dx,offset second_time$         ; ds:dx -> second time around
DosInt  ConStringOutput                 ; post that
mov     dx,offset report$               ;
DosInt  ConStringOutput                 ;
mov     dx,offset buffer$+2             ;
DosInt  ConStringOutput                 ; out it goes!
jmp     resume                          ; resume code

;      Convert binary to decimal ascii

convert_single:
push    ax                             ; save registers
push    bx                             ;
push    cx                             ;
push    dx                             ;

xor     ah,ah                           ; clean out msb
jmp     convert_short                  ; do short conversion

convert_double:
push    ax                             ; save registers
push    bx                             ;
push    cx                             ;
push    dx                             ;

xchg    ax,cx                          ; swap cx with ax

mov     cx,1000                         ; see if we've any thousands
call    decimal_ascii                  ; go call procedure
mov     cx,100                          ; see if we've any hundreds
call    decimal_ascii                  ; go call procedure

convert_short:
mov     cx,10                           ; see if we've any tens
call    decimal_ascii                  ; go call procedure
mov     cx,1                             ; see if we've any ones
call    decimal_ascii                  ; go do it
inc     si                              ; skip over
```

```

        pop     dx             ; restore registers
        pop     cx             ;
        pop     bx             ;
        pop     ax             ;
        ret                  ; to caller

;      do the actual base-of-ten determination

decimal_ascii:
        xor     bl,bl         ; clean register
aloop:
        sub     ax,cx         ; see if cx is in ax
        jc      add_it        ; branch out if negative
        inc     bl            ; bump counter
        jmp     aloop         ; go do again
add_it:
        add     ax,cx         ; restore the one too far
        add     bl,30h        ; make it ascii
        mov     byte ptr [si],bl ; stuff answer it
        inc     si            ; bump pointer
        ret                  ; to caller

;      buffered keyboard entry

ConStringInput:
        push    si            ; save for a spell
        mov     cx,byte ptr [si] ; get legimate count
        mov     ch,0          ; this is counter
        inc     si            ; bump si to point to result$
        push    si            ; save for a spell
ConLoop:
        DosInt  GetChar        ; get a character
        cmp     ax,6700h       ; do we have a <quit>?
        jz      os_quit        ; yes; get out
        cmp     ax,5e00h       ; do we have a control-f1?
        jz      os_suspend     ; yes; get out
        cmp     ax,0c100h      ; do we have a <suspend>?
        jz      os_suspend     ; yes; get out
        cmp     ax,0c200h      ; do we have a <redraw>?
        jz      os_redraw      ; yes; get out

```

```

cmp      al,13          ; is it a carriage return?
jz       ConExit        ; yes, so exit routine
cmp      al,8           ; is it a backspace?
jz       ConBack        ; yes, so handle
cmp      cl,0           ; is cx = 0?
jz       ConFull        ; yes, so handle
mov      dl,al          ; move character into dl
DosInt   PutChar        ; post to screen
inc      si             ; increment pointer
mov      byte ptr [si],al ; save character
dec      cl             ; decrement cl
inc      ch             ; and increment bl
jmp      ConLoop        ; go and do it again

ConBack:
cmp      ch,0           ; have we backspaced all way
jz       ConFull        ; yes, so beep user
LcdInt   GetCursor      ; get cursor position
dec      dh             ; move back column
push     dx             ; save position
LcdInt   SetCursor      ; set cursor pos
mov      dl,' '         ; display a space
DosInt   PutChar        ; post a character
pop      dx             ; get cusor pos back
LcdInt   SetCursor      ; set cursor pos
dec      si             ; bump pointer back
inc      cl             ; increment cl
dec      ch             ; dec ch
jmp      ConLoop        ; go do again

ConFull:
mov      dl,7           ; beep
DosInt   PutChar        ; post to screen
jmp      ConLoop        ; go do again

ConExit:
pop      si             ; get back old ptr
mov      byte ptr [si],ch ; save character count
pop      si             ; get original ptr
ret                ; to caller

js_suspend:
mov      al,0           ; exit w/ return

```


Tandy Vendor Development Internal Document
Number: VDS-263901-02

```

        loop    bin_loop                ; go do again
        ret                                ; to caller

;      strings and things
;      DS that gets moved into ami

begin_of_data    equ    $
offset_data      equ    102h

cod              equ    $-begin_of_data+offset_data
                dw      0                ; code segment
dod              equ    $-begin_of_data+offset_data
                dw      0                ; data segment
eod              equ    $-begin_of_data+offset_data
                dw      0                ; extra segment
sod              equ    $-begin_of_data+offset_data
                dw      0                ; stack segment
spd              equ    $-begin_of_data+offset_data
                dw      0                ; stack offset

hello$          equ    $-begin_of_data+offset_data
                db      'This is a sample program. Type something'
                db      13,10,'$'
back_again$     equ    $-begin_of_data+offset_data
                db      'This is a <Re-Draw> Request',13,10,'$'
second_time$    equ    $-begin_of_data+offset_data
                db      'This is the second time around!!',13,10
                db      'The last time around, you typed:',13,10,'$'
buffer$         equ    $-begin_of_data+offset_data
                db      20
                db      0
                db      21 dup(?)
report$         equ    $-begin_of_data+offset_data
                db      13,10,'You have typed:',13,10,'$'
system_time$    equ    $-begin_of_data+offset_data
                db      13,10,'System Date is: '
date$           equ    $-begin_of_data+offset_data
                db      '00-00-0000; Time is: '
time$           equ    $-begin_of_data+offset_data
                db      '00:00:00.00',13,10

```

```

seg$      db      'Press a Key to Exit','$'
          equ      $-begin_of_data+offset_data
          db      13,10,'CS has '
cod$      equ      $-begin_of_data+offset_data
          db      '0000. DS has '
dod$      equ      $-begin_of_data+offset_data
          db      '0000. ES has '
eod$      equ      $-begin_of_data+offset_data
          db      '0000. SS has '
sod$      equ      $-begin_of_data+offset_data
          db      '0000. SP has '
spd$      equ      $-begin_of_data+offset_data
          db      '0000.',13,10,'$'
end_of_data equ      $
end_of_world equ      $
aplsiz     equ      end_of_world-begin
data_length equ      end_of_data-begin_of_data
code      ends
          end      start

```

DIR600.ASM

```
page      55,132
title     'Dir 600 - Sample Directory display program'
;
; Dir 600 - program that reads the directory for specified
; wildcard.
;
; Version 1.0, 7-15-86
;
cr          equ      0dh          ; carriage return
lf          equ      0ah          ; line feed
;
; DOS Function equates
;
PutChar     equ      02h          ; write char to screen
GetChar     equ      08h          ; get char from keyboard w/o echo
ConStringOutput equ      09h      ; display string to console
Exit        equ      4ch          ; exit to DOS
FindFirst   equ      4eh          ; find first matching file
FindNext    equ      4fh          ; find next matching file
;
; LCD Text Interrupt Functions
;
ClearScreen equ      01h          ; clear screen on lcd
SetCursor   equ      02h          ; set cursor position
GetCursor   equ      03h          ; get cursor position
;
DosInt      macro      function
mov         ah,function          ; ah = function number
int         42h                  ; go call dos
endm
;
LcdInt      macro      function
mov         ah,function          ; ah = function number
int         51h                  ; call BIOS
endm
;
Code        SEGMENT public 'CODE'
ASSUME cs:code,ds:code
```

```

; This is the program header
begin:
    dw      5f10h          ; application check
    dw      9007h          ; our application has code 9007h
    db      0              ; 8086 native code
    db      'V'            ; app is always visible
    db      '.'            ; owned files are AMIs
    db      'DIR'          ; extension (must be unique!!)
    dw      start          ; beginning of IP
    dw      Data_Length+0100h ; place to put SP in AMI
    dw      aplsiz          ; size of application
    dw      Begin_Of_Data   ; beginning of data
    dw      Begin_Of_Data   ; place to keep data
    dw      Data_Length     ; length of data area

dump          proc      far

; data area

Begin_Of_Data equ      $          ;beginning of data area

FileSpec$     db      12,0
               db      13 dup (?)

Hello$        db      'Dir 600 - Tandy 600 Directory Test Suite.',cr,lf
               db      '7-15-86; Version 01.00.',cr,lf
               db      cr,lf,'Enter File to List: $'

End_Road$     db      'Press ANY Key to Exit'
Cr_Lf$        db      cr,lf,'$'

Report$       db      'Files Matching Wildcard:',cr,lf,'$'

; error messages

Dir_Error$    db      cr,lf,'Dir 600: Error in DIR command',cr,lf,'$'

Data_Length   equ      $-Begin_Of_Data          ; length of data area

```


beginning of code area

Start:

screen

```

      LcdInt  ClearScreen          ; clear the screen

      mov     dx,offset Hello$     ;
      DosInt  ConStringOutput      ; post a hello message to

      mov     si,offset FileSpec$-2 ; ds:si -> ASCIIZ filename
      call    ConStringInput       ; do the read
      mov     al,byte ptr [si+1]   ; zero read?
      or      al,al               ; (they only pressed ENTER)
      jz      Start               ; if so, try again

      inc     si                   ; bump counter
      inc     si                   ; up 2
      xor     ah,ah                ; clean off msb
      add     ax,si                ; ax = offset end of string
      mov     si,ax                ; move resultant into si
      mov     byte ptr [si],0h     ; make ASCIIZ

      mov     cx,66h               ; look for all files
      mov     dx,offset FileSpec$  ; ds:dx -> file mask
      DosInt  FindFirst            ; do first scan
      jc      Error                ; if carry, error

      LcdInt  ClearScreen          ; clear the screen
      push    dx                   ; for a minute
      mov     dx,offset Report$    ; dx -> header
      DosInt  ConStringOutput      ; post that
      pop     dx                   ; restore dx

      mov     ax,9                 ; lets add offset 9
      add     ax,dx                 ; to the brew
      mov     si,ax                 ; move that to si
      push    ds                   ; save ds for a while
      push    cx                   ; save our segment
      pop     ds                   ; into ds

```

Display_Find:

```

                                mov     cx,21                ; attempt to read max 21
bytes
Post_Loop:
                                mov     al,byte ptr [si]      ; grab the byte
                                or      al,al                ; is it a zero?
                                jz      Post_Out              ; yes, so boogie
                                mov     dl,al                ; move char to dl
                                DosInt   PutChar              ; post char
                                inc     si                    ; si -> next char
                                loop    Post_Loop             ; go through again

Post_Out:
                                pop     ds                    ; restore ds
                                mov     dx,offset Cr_Lf$     ; newline
                                DosInt   ConStringOutput      ; post to screen
                                DosInt   FindNext              ; go find the next file
                                jnc     Display_Find          ; and post it
                                cmp     ax,2                  ; see if we've an error
                                jz      Clean_Exit            ; (besides "out of files")

;      handle error in directory

Error:
                                mov     dx,offset Dir_Error$  ; dx -> error
                                DosInt   ConStringOutput      ; post it

Clean_Exit:
                                mov     dx,offset End_Road$    ; display "goodbye"
                                DosInt   ConStringOutput      ;
                                call     Getc                  ; wait for a character

                                mov     al,-1                  ; exit w/o return
                                DosInt   Exit                  ; go home

dump
                                endp

ConStringInput proc near
                                push    si                    ; save for a spell
                                mov     cx,byte ptr [si]      ; get legimate count
                                mov     ch,0                  ; this is counter
                                inc     si                      ; bump si to point to
result$

```

```

ConLoop:      push    si                ; save for a spell

              call    Getc              ; get a character
              cmp     al,13             ; is it a carriage return?
              jz      ConExit           ; yes, so exit routine
              cmp     al,8              ; is it a backspace?
              jz      ConBack           ; yes, so handle
              cmp     cl,0              ; is cx = 0?
              jz      ConFull           ; yes, so handle
              mov     dl,al             ; move character into dl
              dosint   PutChar           ; post to screen
              inc     si                ; increment pointer
              mov     byte ptr [si],al  ; save character
              dec     cl                ; decrement cl
              inc     ch                ; and increment bl
              jmp     ConLoop           ; go and do it again

ConBack:      cmp     ch,0              ; have we backspaced all
way          jz      ConFull           ; yes, so beep user
              lcdint  GetCursor         ; get cursor position
              dec     dh                ; move back column
              push    dx                ; save position
              lcdint  SetCursor         ; set cursor pos
              mov     dl,' '            ; display a space
              dosint  PutChar           ; post a character
              pop     dx                ; get cursor pos back
              lcdint  SetCursor         ; set cursor pos
              dec     si                ; bump pointer back
              inc     cl                ; increment cl
              dec     ch                ; dec ch
              jmp     ConLoop           ; go do again

ConFull:      mov     dl,7              ; beep
              dosint  PutChar           ; post to screen
              jmp     ConLoop           ; go do again

ConExit:      pop     si                ; get back old ptr
              mov     byte ptr [si],ch  ; save character count
              pop     si                ; get original ptr

```

```

ConStringInput    ret                ; to caller
                  endo

Getc              proc near
DosInt    GetChar                ; get a character
cmp       ax,6700h              ; do we have a <quit>?
jz        os_quit               ; yes; get out
cmp       ax,5e00h              ; do we have a control-f1?
jz        os_suspend           ; yes; get out
cmp       ax,0c100h             ; do we have a <suspend>?
jz        os_suspend           ; yes; get out
cmp       ax,0c200h             ; do we have a <redraw>?
jz        os_redraw            ; yes; get out
ret                                ; else return

os_suspend:
mov       al,0                  ; exit w/ return clause
jmp       os_out               ; get out of Dodge

os_quit:
mov       al,-1                 ; exit w/o return

os_out:
DosInt    Exit                  ; 'bye

os_redraw:
ret                                ; null routine...

Getc              endo

Aplsiz          equ      $-begin

Code            ends

end            Dump

```