

```

;*****;
;*                               V H E R C                               *;
;*-----*
;*   Task           : Makes a basic function available for           *;
;*                   : access to the Hercules Graphics Card.           *;
;*-----*
;*   Info           : All functions partition the screen display       *;
;*                   : into columns 0-79 and lines 0-24 (text mode) *;
;*                   : & columns 0-719 and lines 0-347 (graphic mode)*;
;*-----*
;*   Author          : Michael Tischer                                  *;
;*   Developed on     : 08/11/89                                         *;
;*   Last update      : 03/02/92                                         *;
;*-----*
;*   Assembly        : MASM VHERC;                                       *;
;*                   : LINK VHERC;                                       *;
;*-----*
;*   Call            : VHERC                                             *;
;*****;

```

```

;== Constants =====

```

```

CONTROL_REG  = 03B8h           ;Control register port address
ADDRESS_6845 = 03B4h           ;6845 address register
DATA_6845     = 03B5h           ;6845 data register
CONFIG_REG    = 03BFh           ;Configuration register
VIO_SEG       = 0B000h         ;Video RAM segment address
CUR_START     = 10              ;Reg. # for CRTC: Start cursor line
CUR_END       = 11              ;Reg. # for CRTC: End cursor line
CURPOS_HI     = 14              ;Reg. # for CRTC: Cursor pos hi byte
CURPOS_LO     = 15              ;Reg. # for CRTC: Cursor pos lo byte

```

```

DELAY          = 20000                      ;Count for delay loop

;== Macros =====

setmode    macro modus                      ;Set control register

            mov  dx,CONTROL_REG             ;Screen control register address
            mov  al,modus                   ;Put new mode in AL register
            out  dx,al                      ;Send mode to control register

            endm

setvk      macro                            ;Write value to CRTC registers
            ;Input: AL = register number
            ;      AH = Value for register

            mov  dx,ADDRESS_6845           ;Index register address
            out  dx,ax                     ;Display register number and new value

            endm

;== Stack =====

stack      segment para stack              ;Definition of stack segment

            dw 256 dup (?)                 ;Stack is 256 words in size

stack      ends                            ;End of stack segment

;== Data =====

data       segment para 'DATA'             ;Define data segment

```

;;= Data needed for demo program =====

```
initm      db 13,10,"VHERC (c) 1987 by Michael Tischer",13,10,13,10
           db "This demonstration program runs only with "
           db " a HERCULES",13,10,"graphics card. If your PC "
           db "has another type of display card, ",13,10
           db "please input an >s< to stop the "
           db " program.",13,10,"Otherwise please press any "
           db "key to start the ",13,10
           db "program ...",13,10,"$"
```

```
str1       db 1,17,16,2,7,0
```

```
str2       db 2,16,17,1,7,0
```

```
domes      db 13,10
           db "This program creates a short graphic demo ",13,10
           db "and a text demo. Pressing a key during the",13,10
           db "demo ends the program.",13,10
           db "Press a key to start the program...",13,10,"$"
```

;;= Table of line offset addresses =====

```
lines      dw 0*160,1*160,2*160 ;Beginning addresses of the lines as
           dw 3*160,4*160,5*160 ;offset addresses in video RAM
           dw 6*160,7*160,8*160
           dw 9*160,10*160,11*160,12*160,13*160,14*160,15*160,16*160
           dw 17*160,18*160,19*160,20*160,21*160,22*160,23*160,24*160
```

```
grafikt    db 35h, 2Dh, 2Eh, 07h, 5Bh, 02h ;Register values for the
           db 57h, 57h, 02h, 03h, 00h, 00h ;graphic mode
```

```

textt      db 61h, 50h, 52h, 0Fh, 19h, 06h  ;Register values for the
           db 19h, 19h, 02h, 0Dh, 0Bh, 0ch  ;text mode

data       ends                               ;End of data segment

;== Code segment =====

code       segment para 'CODE'               ;Definition of the code segment
           org 100h

           assume cs:code, ds:data, es:data, ss:stack

;== this is only the Demo-Program =====

demo       proc far

           mov  ax,data                      ;Get segment address of data segment
           mov  ds,ax                       ;Load into DS
           mov  es,ax                       ;and ES

           ;-- Opening msg., wait for input -----

           mov  ah,9                        ;Output function number for string
           mov  dx,offset initm             ;address of the message
           int  21h                         ;Call DOS interrupt

           xor  ah,ah                       ;Get function number for key
           int  16h                         ;Call BIOS keyboard interrupt
           cmp  al,"s"                      ;Was <s> entered?
           je   ende                        ;YES--> End program
           cmp  al,"S"                      ;Was <S> entered?

```

```

        jne  startdemo          ;NO --> Start demo

ende:    mov  ax,4C00h          ;Function number - end program
        int  21h               ;Call DOS interrupt 21(h)

startdemo label near
        mov  ah,9              ;Output function number for string
        mov  dx,offset domes    ;address of the message
        int  21h               ;Call DOS interrupt

        xor  ah,ah             ;Get function number for key
        int  16h               ;Call BIOS keyboard interrupt

        ;-- Initialize graphic mode -----

        mov  al,11b            ;Graphic and page 2 possible
        call config            ;Configure
        xor  bp,bp             ;Access display page 0
        call grafik            ;Switch to graphic mode
        xor  al,al
        call cgr               ;Erase graphic page 0
        xor  bx,bx             ;Begin in the upper left
        xor  dx,dx             ;Display corner
        mov  ax,347            ;Vertical pixels
        mov  cx,719            ;Horizontal pixels
gr1:     push cx                ;Push horizontal pixels on stack
        mov  cx,ax             ;Vertical pixels in counter
        push ax                ;Push vertical pixels on stack
gr2:     call spix              ;Set pixel
        inc  dx                ;Increment line
        loop gr2               ;Draw line
        pop  ax                ;Get vert. pixels from stack

```

```

sub ax,3 ;next line 3 pixels less
pop cx ;Get horiz. pixels from stack
push cx ;Store horizontal pixels
gr3: push ax ;Push vertical pixels on stack
call spix ;Set pixel
inc bx ;Increment column
loop gr3 ;Draw line
pop ax ;Get vertical pixels from stack
pop cx ;Get horizontal pixels from stack
sub cx,6 ;Next line 6 pixels less
push cx ;Record horizontal pixels
mov cx,ax ;Vertical pixels in counter
push ax ;Note vertical pixels on stack
gr4: call spix ;Set pixel
dec dx ;Decrement line
loop gr4 ;Draw line
pop ax ;Get vertical pixels from stack
sub ax,3 ;Next line 3 pixels less
pop cx ;Get horizontal pixels from stack
push cx ;Record horizontal pixels
push ax ;Record vertical pixels on stack
gr5: call spix ;Set pixel
dec bx ;Increment column
loop gr5 ;Draw line
pop ax ;Get vertical pixels from stack
pop cx ;Get horizontal pixels from stack
sub cx,6 ;Next line 6 pixels less
cmp ax,5 ;Is the vertical line longer than 5
ja gr1 ;YES --> continue

xor ah,ah ;Wait for function nr. for key
int 16h ;Call BIOS keyboard interrupt

```

```

;-- Initialize text mode -----
call text                ;Switch on text mode
mov  cx,0d00h            ;Switch on full cursor
call cdef
call cls                 ;Clear screen

;-- Display strings in display page 0 -----
xor  bx,bx               ;Start in upper left display corner
call calo                ;Convert to offset address
mov  si,offset str1      ;Offset address of string1
mov  cx,16*25            ;The string is 5 characters long
demo1: call print        ;Output string
      loop demo1

;-- Display strings in display page 1 -----
inc  bp                 ;Process display page 1
xor  bx,bx              ;Start in the upper left corner
call calo               ;Convert to offset address
mov  si,offset str2     ;Offset address of string1
mov  cx,16*25           ;string is 5 characters long
demo2: call print        ;Output string
      loop demo2

demo3: setmode 10001000b ;Display text page 1

;-- short Pause -----
mov  cx,DELAY           ;Load counter
pause: loop pause       ;Count to 65,536

```

```

        setmode 00001000b          ;Display page 0

;-- short pause -----
mov     cx,DELAY                   ;Load counter
pause1: loop pause1                ;Count to 65,536

mov     ah,1                       ;Test function nr. for key
int     16h                       ;Call BIOS-keyboard-Interrupt
je      demo3                      ;No key --> continue

xor     ah,ah                      ;Get function number for key
int     16h                       ;Call BIOS-keyboard-Interrupt

mov     bp,0                       ;Display page 1
call    cls                       ;Clear screen
mov     cx,0D0ch                   ;Restore normal cursor
call    cdef
call    cls                       ;Clear screen
jmp     ende                      ;End program

demo    endp

;== The actual functions follow =====

;-- CONFIG: configures the HERCULES card -----
;-- Input      : AL : bit 0 = 0 : Only text presentation possible
;--              1 : also graphic presentation possible
;--              bit 1 = 0 : RAM for display page 2 off
;--              1 : RAM for display page 2 on
;-- Output     : none
;-- Register   : AX and DX are changed

```



```

config    proc near

        mov     dx,CONFIG_REG      ;Address of configuration register
        out     dx,al              ;Set new value
        ret                               ;Back to caller

config    endp

;-- TEXT: switches the text presentation on -----
;-- Input   : none
;-- Output  : none
;-- Register : AX and DX are changed

text      proc near

        mov     si,offset textt    ;Offset address of the register-table
        mov     bl,00100000b       ;Display page 0,text mode,blinking
        jmp     short vcprog       ;Program video-controller again

text      endp

;-- GRAFIK: switches on the graphic mode -----
;-- Input   : none
;-- Output  : none
;-- Register : AX and DX are changed

grafik    proc near

        mov     si,offset grafikt  ;Offset address of the register-table
        mov     bl,00000010b       ;Display page 0, graphic mode

```

grafik endp

```
!-- VCPROG: programs the video controller -----
!-- Input   : SI = address of a register-table
!--         BL = value for display-control-register
!-- Output  : none
!-- register : AX, SI, BH, DX and FLAGS are changed
```

vcprog proc near

```
        setmode bl                    ;Bit 3 = 0: display off

        mov  cx,12                    ;12 registers are set
        xor  bh,bh                    ;Start with register 0
vcpl:    lodsb                        ;Get register value from the table
        mov  ah,al                    ;Register value to AH
        mov  al,bh                    ;Number of the register to AL
        setvk                        ;Transmit value to the controller
        inc  bh                       ;Address next register
        loop vcpl                    ;Set additional registers

        or   bl,8                    ;Bit 3 = 1: display on
        setmode bl                    ;Set new mode
        ret                        ;Back to caller
```

vcprog endp

```
!-- cDEF: sets the start and end line of the cursor-----
!-- Input   : cL = start line
!--         cH = end line
!-- Output  : none
!-- register : AX and DX are changed
```

```

cdef      proc near

            mov  al,CUR_START      ;Register 10: start line
            mov  ah,cl             ;Start line to AH
            setvk                ;Transmit to video-controller
            mov  al,CUR_END        ;Register 11: Endline
            mov  ah,ch             ;End line to AH
            setvk                ;Transmit to video-controller
            ret

cdef      endp

;-- SETBLINK : sets the blinking display cursor -----
;-- Input    : DI = offset address of the cursor
;-- Output   : none
;-- register : BX, AX and DX are changed

setblink  proc near

            mov  bx,di             ;Transmit offset to BX
            mov  al,CURPOS_HI      ;Register 15:Hi Byte of cursor offset
            mov  ah,bh             ;HI byte of the offset
            setvk                ;Transmit to video-controller
            mov  al,CURPOS_LO      ;Register 15:Lo-Byte of cursor offset
            mov  ah,bl             ;Lo byte of the offset
            setvk                ;Transmit to CRT
            ret

setblink  endp

;-- GETVK    : reads a byte from one register of the video-controller -

```

```

;-- Input      : AL = number of the register
;-- Output     : AL = content of the register
;-- register   : DX and AL are changed

```

```
getvk      proc near
```

```

    mov  dx,ADDRESS_6845    ;Address of the index register
    out  dx,al              ;Send number of the register
    jmp  $+2                ;Short io pause
    inc  dx                  ;Address of the index register
    in   al,dx              ;Read content to AL
    ret                     ;Back to caller

```

```
getvk      endp
```

```

;-- SCROLLUp: scrolls a window by N lines upward -----
;-- Input   : BL = line upper left
;--         : BH = column upper left
;--         : DL = line lower right
;--         : DH = column lower right
;--         : CL = number of the lines to be scrolled
;--         : BP = number of the display page (0 or 1)
;-- Output  : none
;-- register: only FLAGS are changed
;-- Info    : the display lines released are erased

```

```
scrollup   proc near
```

```

    cld                     ;Increment for string instructions
    push ax                  ;Store all changed registers
    push bx                  ;on the stack
    push di                  ;In this case the sequence

```

```

push si                                ;must be followed !

push bx                                ;These three registers are returned
push cx                                ;from the stack before
push dx                                ;the end of the routine
sub dl,bl                              ;Calculate number of lines
inc dl                                 ;Deduct number
sub dl,cl                              ;of lines to be scrolled
sub dh,bh                              ;Calculate number of columns
inc dh

call calo                              ;Convert upper left in offset
mov si,di                              ;Note address in SI
add bl,cl                              ;First line in scrolled window
call calo                              ;Convert first line in offset
xchg si,di                             ;Exchange SI and DI
push ds                                ;Store segment register
push es                                ;on the stack
mov ax,VIO_SEG                         ;Segment address of the video RAM
mov ds,ax                              ;to DS
mov es,ax                              ;and ES
supl: mov ax,di                        ;Note DI in AX
      mov bx,si                        ;Note SI in BX
      mov cl,dh                        ;Number of columns in counter
      rep movsw                        ;Move a line
      mov di,ax                        ;Restore DI from AX
      mov si,bx                        ;Restore SI from BX
      add di,160                       ;Set next line
      add si,160

      dec dl                           ;Processed all lines ?
      jne supl                         ;NO --> move another line
      pop es                           ;Get segment register from
      pop ds                           ;stack

```

```

pop     dx                ;Get lower right corner
pop     cx                ;Get number of lines
pop     bx                ;Get upper left corner
mov     bl,dl             ;Lower line to BL
sub     bl,cl             ;Deduct number of lines
inc     bl
mov     ah,07h            ;Color : black on white
call    cclear            ;Erase liberated lines

pop     si                ;CX and DX have been brought back
pop     di                ;already
pop     bx
pop     ax

ret                     ;Back to caller

```

```

;-- SCROLLDN: scroll a Window by N lines upwards -----
;-- Input      : BL = line upper left
;--             BH = column upper left
;--             DL = line lower right
;--             DH = column lower right
;--             CL = number of the lines to be scrolled
;--             BP = number of the display page (0 or 1)
;-- Output     : none
;-- register   : only FLAGS are changed
;-- Info      : released lines are deleted

```

```
cld          ;Increment on string instructions
```

```

push ax                ;Secure all changed registers on the
push bx                ;stack
push di                ;In this case the sequence must
push si                ;be followed!

push bx                ;These three registers are
push cx                ;returned from the stack before the
push dx                ;end of the routine

sub dh,bh              ;Calculate number of columns
inc dh
mov al,b1              ;Record line upper left in AL
mov bl,dl              ;Line lower right top lower left
call calo              ;Convert upper left in offset
mov si,di              ;Note address in SI
sub bl,cl              ;Deduct number of chars to scroll
call calo              ;Convert upper left in offset
xchg si,di             ;Exchange SI and DI
sub dl,al              ;Calculate number of lines
inc dl
sub dl,cl              ;Deduct number of lines to scroll
push ds                ;Store segment register on the
push es                ;stack
mov ax,VIO_SEG         ;Segment address of the video RAM
mov ds,ax              ;to DS
mov es,ax              ;and ES
sdn1: mov ax,di         ;Record DI in AX
      mov bx,si         ;Record SI in BX
      mov cl,dh         ;Number of columns in counter
      rep movsw          ;Move a line
      mov di,ax         ;Restore DI from AX

```

```

mov    si,bx                ;Restore SI from BX
sub    di,160               ;Set next line
sub    si,160
dec    dl                   ;All lines processed ?
jne    sdn1                 ;NO --> move another line
pop    es                   ;Get segment register from
pop    ds                   ;stack
pop    dx                   ;Get lower right corner
pop    cx                   ;Get number of lines
pop    bx                   ;Get upper left corner
mov    dl,bl                ;Upper line to DL
add    dl,cl                ;Add number of lines
dec    dl
mov    ah,07h               ;Color : black on white
call   clear                ;Erase liberated lines

pop    si                   ;CX and DX have already
pop    di                   ;been read
pop    bx
pop    ax

ret                          ;Back to caller

```

scrollldn endp

```

;-- cLS: clear the whole screen -----
;-- Input    : BP = number of the display page (0 or 1)
;-- Output   : none
;-- register : only FLAGS are changed

```

cls proc near


```

mov  ah,07h                ;Color is white on black
xor   bx,bx                ;Upper left is (0/0)
mov   dx,4F18h             ;Lower right is (79/24)

    ;-- perform clear -----

cls      endp

;-- CLEAR: fills a designated display area with space character -----
;-- Input      : AH = Attribute/color
;--            BL = line upper left
;--            BH = column upper left
;--            DL = line lower right
;--            DH = column lower right
;--            BP = number of the display page (0 or 1)
;-- Output     : none
;-- register   : only FLAGS are changed

clear    proc near

        cld                ;Increment on string instructions
        push cx             ;Secure all changed
        push dx             ;registers on the stack
        push si
        push di
        push es
        sub  dl,bl          ;Calculate number of lines
        inc  dl
        sub  dh,bh          ;Calculate number of columns
        inc  dh
        call calo           ;Offset address of upper left corner
        mov  cx,VIO_SEG     ;Segment address of the video RAM

```

```

        mov     es,cx                ;to ES
        xor     ch,ch                ;Hi byte of the counter to 0
        mov     al," "              ;Space character
clear1:  mov     si,di                ;Note DI in SI
        mov     cl,dh                ;Number of columns in counter
        rep     stosw                ;Store space character
        mov     di,si                ;Restore DI from SI
        add     di,160                ;Set next line
        dec     dl                    ;All lines processed ?
        jne     clear1              ;NO --> erase another line

        pop     es                    ;Get secured registers
        pop     di                    ;from the stack
        pop     si
        pop     dx
        pop     cx
        ret                          ;Back to caller

clear    endp

```

```

;-- PRINT: outputs a string on the display -----
;-- Input      : AH = attribute/color
;--            DI = offset address of the first character
;--            SI = offset address of the strings to DS
;--            BP = number of the display page (0 or 1)
;-- Output     : DI points behind the last character to be output
;-- register   : AL, DI and FLAGS are changed
;-- Info       : the string must ne terminated with NUL-character.
;--            other control characters are not recognized

print    proc near

```

```

        cld                                ;Increment on string instructions
        push si                            ;SI, DX and ES to the stack
        push es
        push dx
        mov dx,VIO_SEG                    ;First segment address of video RAM
        mov es,dx                          ;to DX and then to ES
        jmp printl                         ;Get first character from string
print0:  stosw                             ;Store attribute and color in V-RAM
printl:  lodsb                             ;Get next character from the string
        or  al,al                          ;Is it NUL
        jne print0                        ;NO --> output

printe:  pop  dx                           ;Get SI, DX and ES from stack again
        pop  es
        pop  si
        ret                               ;Back to caller

print    endp

;-- cALO: converts line and column into offset address -----
;-- Input      : BL = line
;--             BH = column
;--             Bp = number of the display page (0 or 1)
;-- Output     : DI = offset address
;-- register   : DI and FLAGS are changed

calo     proc near

        push ax                            ;Record AX on the stack
        push bx                            ;Record BX on the stack

        shl  bx,1                          ;Column and line times 2

```

```

        mov     al,bh                ;Column to AL
        xor     bh,bh                ;Hi byte
        mov     di,[lines+bx]        ;Get offset address of the line
        xor     ah,ah                ;Hi byte for column offset
        add     di,ax                ;Add lines- and column offset
        or      bp,bp                ;Display page 0?
        je      caloe                ;YES --> address ok

        add     di,8000h              ;Add 32 KB for display page 1

caloe:   pop     bx                    ;Get BX from stack again
        pop     ax                    ;Get AX from the stack again
        ret                                ;Back to caller

calo     endp

;-- CGR: clear the complete graphic screen -----
;-- Input      : BP = number of the display page (0 or 1)
;--            AL = 00(h) : erase all pixels
;--            FF(h) : set all pixels
;-- Output     : none
;-- register   : AH, BX, cX, DI and FLAGS are changed

cgr      proc near

        push    es                    ;Record ES on the stack
        cbw                                ;Expand AL to AH
        xor     di,di                  ;Offset address in video RAM
        mov     bx,VIO_SEG             ;Segment address display page 0
        or      bp,bp                  ;Erase page 1?
        je      cgr1                  ;NO --> erase page 0

```

```

        add    bx,0800h                ;Segment address display page 1

cgr1:    mov    es,bx                  ;Segment address to segment register
        mov    cx,4000h                ;A page is 16K-words
        rep    stosw                  ;Fill page
        pop    es                      ;Get ES from stack
        ret                            ;Back to caller

cgr      endp

;-- SPIX: sets a pixel in the graphic display -----
;-- Input      : BP = number of the display page (0 or 1)
;--            BX = column (0 to 719)
;--            DX = line  (0 to 347)
;-- Output     : none
;-- register   : AX, DI and FLAGS are changed

spix     proc near

        push   es                    ;Store ES on the stack
        push   bx                    ;Store BX on the stack
        push   cx                    ;Store cX on the stack
        push   dx                    ;Store DX on the stack

        xor    di,di                ;Offset address in video RAM
        mov    cx,VIO_SEG            ;Segment address display page 0
        or     bp,bp                ;Access page 1 ?
        je     spixl                ;NO --> access page 0

        mov    cx,0800h              ;Segment address display page 1

spixl:    mov    es,cx                ;Segment address in segment register

```

```

mov ax,dx          ;Move line to AX
shr ax,1           ;Shift line right 2 times
shr ax,1           ;This divides by four
mov cl,90          ;The factor is 90
mul cl             ;Multiply line by 90
and dx,11b         ;AND all bits except for 0 and 1
mov cl,3           ;3 shifts
ror dx,cl          ;Rotate right (* 2000(h))
mov di,bx          ;Column to DI
mov cl,3           ;3 shifts
shr di,cl          ;divide by 8
add di,ax          ;+ 90 * int(line/4)
add di,dx          ;+ 2000(h) * (line mod 4)
mov cl,7           ;Maximum of 7 moves
and bx,7           ;Column mod 8
sub cl,b1          ;7 - column mod 8
mov ah,1           ;Determine bit value of the pixels
shl ah,cl
mov al,es:[di]     ;Get 8 pixels
or al,ah           ;Set pixel
mov es:[di],al     ;Write 8 pixels ;

pop dx             ;Get DX from stack
pop cx             ;Get cX from stack
pop bx             ;Get BX from stack
pop es             ;Get ES from stack
ret                ;Back to caller

```

```
spix      endp
```

```
;;= End =====
```

code

ends

end demo

;End of the code segment