

```

***** S 6 4 3 5 P A . A S M *****
;*
;*-----*
;* Task      : Contains routines for generating sprites in 640x350 pixel graphic mode on an EGA and VGA card.
;*-----*
;* Author    : MICHAEL TISCHER
;* Developed on : 12/08/90
;* Last update  : 01/14/91
;*-----*
;* Assembly   : MASM /mx S6435PA; or TASM -mx S6435PA
;*             ... Link to S6435P.PAS
;*****
;

;== Constants ==
SC_INDEX      = 3c4h      ;Index register for sequencer ctrl.
SC_MAP_MASK   = 2         ;Number of map mask register
SC_MEM_MODE   = 4         ;Number of memory mode register

GC_INDEX      = 3ceh      ;Index register for graphics ctrl.
GC_READ_MAP   = 4         ;Number of read map register
GC_BIT_MASK   = 8         ;Number of bit mask register

PIXX          = 640       ;Horizontal resolution

;== Data segment ==
DATA segment word public
DATA ends

;== Program ==
CODE segment byte public ;Program segment
assume cs:code, ds:data

;-- Public declarations -----
public copybuf2video
public mergeandcopybuf2video
public copyvideo2buf

;-----
;-- CopyBuf2Video: Copies the contents of a rectangular area from the video RAM that were saved by CopyVideo2Buf back to video RAM
;-- Call from TP : CopyBuf2Plane( bufptr : pointer;
;--                               topage : byte;
;--                               tox,
;--                               toy : integer;
;--                               rwidth,
;--                               rheight : byte );
;-- Info : See CopyVideo2Buf

copybuf2video proc near

sfr0 struc ;Structure for stack access
bp0 dw ? ;Gets BP
stofs0 dw ? ;Loc. var.: Start offset in video RAM
ret_adr0 dw ? ;Return address from caller
rheight0 dw ? ;Height of range
rwidth0 dw ? ;Width
toy dw ? ;To Y-coordinate
tox dw ? ;To X-coordinate
topage dw ? ;To page
bufptr0 dd ? ;Pointer to buffer
sfr0 ends

fr equ [ bp - bp0 ] ;Addresses structure elements
bfr equ byte ptr [ bp - bp0 ] ;Addresses stack element as byte

sub sp,2 ;Space for local variables

push bp ;Prepare BP register for
mov bp,sp ;addressing parameters

push ds

cld ;Increment on string instructions

;-- Calculate segment address for access to video RAM -----
mov ax,0A000h ;Move ES to start of TO page
cmp bfr.topage,0 ;Page 0?

```

```

je          cv0                ;Yes --> AL already ok
mov     ax,0A6D6h              ;No --> page 1 of A6D6H
cv0:      mov     es,ax

;-- Calculate offset for target position in page -----
mov     ax,PIXX / 8            ;AX to target position FROM
mul     fr.toy
mov     bx,fr.tox
shr     bx,1
shr     bx,1
shr     bx,1
add     bx,ax
mov     fr.stofs0,bx          ;Store start offset in local variable

lds     si,fr.bufptr0         ;Set DS:SI on the buffer

;-- Load counter for the copying loop -----
mov     dl,bfr.rwidth0        ;DL = Bytes
mov     bx,PIXX / 8           ;BX as offset to next line
sub     bl,dl
xor     ch,ch                  ;High byte of counter is always 0

;-- Set addressable bitplane -----
mov     ah,1                  ;Load plane number as bit mask
mov     al,SC_MAP_MASK        ;Register number to AL
cv1:      mov     dx,SC_INDEX   ;Open access to plane for
out     dx,ax                 ;processing

;-- Copy routine for a bitplane
;-- without checking the background

mov     di,fr.stofs0          ;DI to startoffset
mov     dh,bfr.rheight0       ;DH = Lines
mov     dl,bfr.rwidth0        ;DL = Bytes
cv2:      mov     cl,dl         ;Number of bytes to CL

rep movsb                     ;Copy lines
add     di,bx                  ;DI in next line
dec     dh                     ;Another line?
jne     cv2                    ;Yes ---> Continue

shl     ah,1                   ;Shift to next plane
test    ah,16                  ;All planes processed?
je      cv1                    ;No --> Continue with next plane

mov     ax,(0Fh shl 8)+ SC_MAP_MASK ;Allow access to
mov     dx,SC_INDEX            ;all bitplanes
out     dx,ax

pop     ds                     ;Retrieve DS and BP
pop     bp

add     sp,2                   ;Add SP to local variables
ret     14                     ;On return remove parameters
;from stack
copybuf2video endp

```

```

;-----
;-- MergeAndCopyBuf2Video: A bit mask links the contents of a back-
;--                          ground buffer to the contents of a sprite
;--                          buffer and copies the result to video RAM
;-- Call from TP : MergeAndCopyBuf2Video( spribufptr,
;--                                     hgbufptr,
;--                                     andbufptr : pointer;
;--                                     page      : byte;
;--                                     tox,
;--                                     toy      : integer;
;--                                     rwidth,
;--                                     rheight : byte );
;-- Info          : See CopyVideo2Buf

```

```
mergeandcopybuf2video proc near
```

```

sfr2      struc                ;Structure for stack access
bp2       dw ?                 ;Gets BP
andptr2   dd ?                 ;Local var: Pointer in AND buffer
stofs2    dw ?                 ;Local var: Startoffset in video RAM
ret_adr2  dw ?                 ;Return address from caller
rheight2  dw ?                 ;Height of range

```

```

width2    dw ?           ;Width
toy2      dw ?           ;To Y-coordinate
tox2      dw ?           ;To X-coordinate
topage2   dw ?           ;To page
andbufptr dd ?           ;Pointer to AND buffer
hgbufptr  dd ?           ;Pointer to background buffer
spribufptr dd ?          ;Pointer to sprite buffer
sfr2      ends           ;Ends structure

fr         equ [ bp - bp2 ] ;Addresses structure elements
bfr        equ byte ptr [ bp - bp2 ] ;Addresses stack element as byte

sub    sp,6              ;Space for local variables

push   bp                ;Prepare BP register for
mov    bp,sp             ;addressing parameters

push   ds

cld                      ;Increment on string instructions

;-- Calculate segment address for access to video RAM -----

mov    ax,0A000h         ;Move ES to start of TO page
cmp    bfr.topage2,0      ;Page 0?
je     cm0               ;Yes --> AL already O.K.

mov    ax,0A6D6h         ;No --> page 1 of A6D6H

cm0:    mov    es,ax

;-- Calculate offset for target position in page -----

mov    ax,PIXX / 8        ;AX to target position FROM
mul    fr.toy2
mov    bx,fr.tox2
shr    bx,1
shr    bx,1
shr    bx,1
add    bx,ax
mov    fr.stofs2,bx       ;Store start offset in local variable

;-- Load counter for the copying loop -----

mov    dl,bfr.rwidth2     ;DL = Bytes
mov    bx,PIXX / 8        ;BX as offset to next line
sub    bl,dl
xor    ch,ch              ;High byte of counter is always 0

mov    ax,word ptr fr.andbufptr+2 ;Copy segment address of
mov    word ptr fr.andptr2+2,ax   ;AND pointer to local var.

;-- Set addressable bitplane -----

mov    ah,1              ;Load plane number as bit mask

cm1:    mov    al,SC_MAP_MASK ;Register number to AL
mov    dx,SC_INDEX        ;Open access to plane for
out    dx,ax              ;processing

;-- Copy routine for a bitplane
;-- without checking the background

mov    dx,word ptr fr.andbufptr ;Copy offset address of
mov    word ptr fr.andptr2,dx   ;AND pointer to local var.
mov    di,fr.stofs2            ;DI to start offset
mov    dh,bfr.rheight2         ;DH = Lines
mov    dl,bfr.rwidth2          ;DL = Bytes

cm2:    mov    cl,dl           ;Number of bytes to CL

cm3:    lds    si,fr.hgbufptr   ;Load pointer to background buffer
lodsb                   ;Load byte from background buffer
mov    word ptr fr.hgbufptr,si ;Save incr. offset

lds    si,fr.andptr2         ;Load pointer in AND buffer
and    al,[si]               ;Link background to AND mask
inc    si                    ;Increment offset in AND buffer
mov    word ptr fr.andptr2,si ;and save

lds    si,fr.spribufptr      ;Load pointer to sprite buffer
or     al,[si]               ;OR byte from sprite buffer
inc    si                    ;Increment byte in sprite buffer
mov    word ptr fr.spribufptr,si ;and save

stosb                      ;Write byte to video RAM

```

```

loop    cm3                ;Process next byte

add     di,bx              ;DI in next line
dec     dh                ;Another line?
jne     cm2                ;Yes ---> Continue

shl     ah,1               ;Switch to next plane
test    ah,16              ;All planes processed?
je      cm1                ;No --> Continue with next plane

mov     ax,(0Fh shl 8)+ SC_MAP_MASK ;Allow access to all
mov     dx,SC_INDEX        ;bitplanes
out     dx,ax

pop     ds                 ;Retrieve DS and BP
pop     bp

add     sp,6               ;Add SP to local variables
ret     22                 ;On return remove parameters
                        ;from stack

mergeandcopybuf2video endp

;-----
;-- CopyVideo2Buf: Copies a rectangular range from the video RAM to
;--                a buffer
;-- Call from TP : CopyVideo2Buf( bufptr : pointer;
;--                               frompage: byte;
;--                               fromx,
;--                               fromy   : integer;
;--                               rwidth,
;--                               rheight : byte );
;-- Info          : In this version of the routine, the area to be
;--                copied must begin at a pixel column that can be
;--                divided by eight and extend over a multiple of
;--                eight pixels.
;--                RWIDTH refers to the number of bytes per line
;--                of a bit plane!

copyvideo2buf proc near

sfrl    struc              ;Structure for stack access
bpl     dw ?               ;Gets BP
stofs1  dw ?               ;Local var: Start offset in video RAM
ret_adr1 dw ?              ;Return address from caller
rheight1 dw ?              ;Pixel line height
rwidth1  dw ?              ;Pixel width of area
fromy    dw ?              ;From Y-coordinate
fromx    dw ?              ;From X-coordinate
frompage dw ?              ;From page
bufptr1  dd ?              ;Pointer to buffer
sfrl     ends              ;Ends structure

fr      equ [ bp - bpl ]    ;Addresses structure elements
bfr     equ byte ptr [ bp - bpl ] ;Addresses stack element as byte

sub     sp,2               ;Space for local variables

push    bp                 ;Prepare BP register for
mov     bp,sp              ;addressing parameters

push    ds                 ;Increment on string instructions
cld

;-- Calculate segment address for access to video RAM -----

mov     ax,0A000h          ;Move ES start of FROM page
cmp     bfr.frompage,0      ;Page 0?
je      cc0                 ;Yes --> AL already ok

mov     ax,0A6D6h          ;No --> page 1 of A6D6H

cc0:    mov     ds,ax

;-- Form offset in page to be read -----

mov     ax,PIXX / 8         ;Move AX to target position FROM
mul     fr.fromy
mov     bx,fr.fromx
shr     bx,1
shr     bx,1
shr     bx,1
add     bx,ax
mov     fr.stofs1,bx        ;Store start offset in local variable

les     di,fr.bufptr1       ;Set ES:DI on the buffer

```

```

;-- Load counter for the copy loop -----
mov     dl,bfr.rwidthl    ;DL = Bytes
mov     bx,PIXX / 8       ;BX as offset for next line
sub     bl,dl
xor     ch,ch             ;High byte of counter is always 0

;-- Set addressable bitplane -----

xor     ah,ah             ;Begin with plane #0
mov     al,GC_READ_MAP    ;Register number to AL

cc1:    mov     dx,GC_INDEX ;Load index address of graphic ctrl.
out     dx,ax            ;Load read map register

;-- Copy routine for a bitplane
;-- without checking the background

mov     dh,bfr.rheightl   ;DH = Lines
mov     dl,bfr.rwidthl    ;DL = Bytes
mov     si,fr.stofsl      ;Load start offset to SI

cc2:    mov     cl,dl       ;Number of bytes to CL
rep     movsb             ;Copy lines
add     si,bx             ;SI in next line
dec     dh                ;Another line?
jne     cc2               ;Yes --> Continue

inc     ah                ;Switch to next plane
cmp     ah,4              ;All planes processed?
jne     cc1               ;No --> Continue with next plane

pop     ds                ;Retrieve DS and BP
pop     bp

add     sp,2              ;Add SP to local variables
ret     14                ;On return remove parameters
;from stack

copyvideo2buf endp

;== End =====

CODE     ends             ;End of code segment
end      end               ;End of program

```