

```
;*****;
;*          S 3 2 2 0 C A . A S M          *;
;*-----*
;* Task      : Contains routines for generating sprites in *;
;*            320x200 256 color mode on a VGA card.      *;
;*-----*
;* Author    : Michael Tischer                      *;
;* Developed on : 09/05/90                          *;
;* Last update : 02/17/92                          *;
;*-----*
;* Assembly   : MASM /mx S3220CA; or TASM -mx S3220CA *;
;*            ... Link to S3220C.C                  *;
;*****;
```

```
IGROUP group _text      ;Program segment
DGROUP group const,_bss, _data ;Data segment
      assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP
```

```
CONST segment word public 'CONST';This segment handles all
CONST ends                      ;readable constants
```

```
_BSS segment word public 'BSS' ;This segment handles all uninitial-
_BSS ends                      ;ized static variables
```

```
_DATA segment word public 'DATA' ;This segment handles all initialized
                                ;global and static variables
```

```
_DATA ends
```

```
== 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_GRAPH_MODE  = 5            ;Number of graphic mode register
```

```
VERT_RESCAN    = 3DAh          ;Input status register #1
PIXX           = 320           ;Horizontal resolution
```

```
== Program ==
```

```
_TEXT segment byte public 'CODE' ;Program segment
```

```
-- Public declarations -----
```

```
public _blockmove
```

```
-- BLOCKMOVE: Moves a group of pixels in video RAM -----
-- Declaration : void blockmove( unsigned char frompage, int fromx,
--                               int fromy, unsigned char topage,
--                               int tox, int toy, BYTE pwidth,
--                               byte pheight, byte *bmskp );
```

```
_blockmove proc near
```

```
sframe4 struct          ;Structure for stack access
bp4 dw ?                ;Gets BP
additive dw ?           ;Local variable
restc dw ?
movec dw ?
dataseg dw ?
ret_adr4 dw ?           ;Return address from caller
frompage dw ?           ;From page
fromx dw ?              ;From X-coordinate
fromy dw ?              ;From Y-coordinate
topage dw ?             ;To page
tox dw ?                ;To X-coordinate
toy dw ?                ;To Y-coordinate
pwidth dw ?             ;Pixel width
pheight dw ?            ;Pixel height
bmskp dw ?              ;Pointer to buffer with bit mask
sframe4 ends           ;End structure
```

```
frame equ [ bp - bp4 ] ;Addresses structure elements
```

```

sub    sp,8                ;8 bytes for local variables

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

push   ds
push   si
push   di

mov     frame.dataseg,ds

mov     dx,GC_INDEX        ;Get current write mode and
mov     al,GC_GRAPH_MODE   ;initialize write mode 1
out     dx,al
inc     dx
in      al,dx
push    ax                 ;Push current mode onto stack
and     al,not 3
or      al,1
out     dx,al

mov     al,4               ;Move DS to start of FROM page
mov     cl,byte ptr frame.frompage
mul     cl
or      al,0A0h
xchg    ah,al
mov     ds,ax

mov     al,4               ;Move ES to start of TO page
mov     cl,byte ptr frame.topage
mul     cl
or      al,0A0h
xchg    ah,al
mov     es,ax

mov     ax,PIXX / 4        ;Move SI to FROM starting position
mul     frame.fromy
mov     si,frame.fromx
shr     si,1
shr     si,1
add     si,ax

mov     ax,PIXX / 4        ;Move DI to TO position
mul     frame.toy
mov     di,frame.tox
shr     di,1
shr     di,1
add     di,ax

mov     dh,byte ptr frame.pheight ;DH = Pixel lines
mov     dl,byte ptr frame.pwidth  ;DL = Bytes
shr     dl,1
shr     dl,1

mov     bx,PIXX / 4        ;Move BX as offset to next line
sub     bl,dl
xor     ch,ch              ;High byte of counter is always 0
cmp     frame.bmskp,0      ;No background?
jne     mt2                ;None, use other copy routine

push    dx                 ;Push DX onto stack
mov     dx,SC_INDEX        ;Get bitplane access
mov     ah,0Fh
mov     al,SC_MAP_MASK
out     dx,ax
pop     dx                 ;Pop DX

;-- Copy routine for all four bitplanes,
;-- without checking the background

mt1:    mov     cl,dl        ;Number of bytes to CL

rep     movsb              ;Copy lines
add     di,bx              ;DI and
add     si,bx              ;SI in next line
dec     dh                 ;Still more lines?

```

```

    jne    mt1                ;Yes --> Continue
    jmp    short mtend        ;No --> Get out of routine

;-- Copy routine for individual bitplanes using
;-- the specified bit mask arrays

mt2:    mov     byte ptr frame.restc,dh ;First specify variables
    mov     byte ptr frame.movec,dl ;placed in local variables
    mov     frame.additive,bx        ;on the stack

    mov     al,SC_MAP_MASK    ;Address permanent
    mov     dx,SC_INDEX      ;Map mask register
    out     dx,al
    inc     dx                ;Increment DX on data register

    mov     bx,frame.bmskp    ;BX is pointer to bit mask array
    push    ds
    mov     ds,frame.dataseg
    mov     al,[bx]           ;Load first byte
    xor     ah,ah             ;Start with an even byte
    pop     ds

mt3:    mov     cl,byte ptr frame.movec ;Move number of bytes to CL

mt4:    out     dx,al          ;Set bit mask
    movsb                    ;Copy 4 bytes

    inc     ah                ;Increment odd/even counter
    test    ah,1              ;Odd again?
    jne     mt5               ;Yes --> Move nibble

;-- Get next byte from buffer on every even byte -----

    inc     bx                ;BX to next bit mask byte
    push    ds
    mov     ds,frame.dataseg
    mov     al,[bx]           ;Load next byte
    pop     ds
    loop    mt4                ;Next four latches
    jmp     short mt6

mt5:    shr     al,1           ;Get odd byte bit mask from
    shr     al,1              ;low nibble
    shr     al,1
    shr     al,1
    loop    mt4                ;Next four latches

mt6:    add     di,frame.additive ;Add DI and
    add     si,frame.additive ;SI to next line
    dec     byte ptr frame.restc ;Still more lines?
    jne     mt3                ;Yes --> Continue

mtend:  mov     dx,GC_INDEX    ;Revert to old
    pop     ax                ;write mode
    mov     ah,al
    mov     al,GC_GRAPH_MODE
    out     dx,ax

    pop     di
    pop     si
    pop     ds
    pop     bp

    add     sp,8                ;Clear local variables
    ret

_blockmove endp

;== End =====

_text    ends                ;End of program segment
        end                  ;End of assembler source

```