```
;=================================================================
==================================
;
SPRITE ROUTINES
;=================================================================
==================================
;
Peter 'Sig' Hewett
;
- 2016
; Routines for more advanced handling and manipulation of
sprites
;-----------------------------------------------------------------
---------------------------------

;=================================================================
==================================
;
MOVE SPRITE LEFT
;=================================================================
==================================
; Moves a sprite left one pixel - using the whole screen (with
the X extended bit)
; X = number of hardware sprite to move (0 - 7) - Register is
left intact
;
; NOTE : to move a sprite multiple pixels, you call this
multiple times. One day I might have a crack
;        at doing one for multiple pixels, but at this point I
don't think I could do one that would
;        justify the extra code and provide a performance boost
to make it worthwhile.
;-----------------------------------------------------------------
---------------------------------
; Fixed bug in the strange extended bit behavior.  Flipping the
bit on negative flag sets it on $FF
; but also flips it on every other change from 128 ($80) to 255
($FF) making it flicker.
; Checking instead for 0 corrects this.
;-----------------------------------------------------------------
---------------------------------
#region "MoveSpriteLeft"

MoveSpriteLeft
        lda SPRITE_POS_X,x                      ; First check
for 0 (NOT negative)
```

```
        bne @decNoChange                        ; branch if NOT
0

        dec SPRITE_POS_X,x                      ; Decrement
Sprite X position by 1 (to $FF)

        lda BIT_TABLE,x                         ; fetch the bit
needed to change for this sprite
        eor SPRITE_POS_X_EXTEND                 ; use it as a
mask to flip the correct X extend bit
        sta SPRITE_POS_X_EXTEND                 ; store teh data
then save it in the VIC II register
        sta VIC_SPRITE_X_EXTEND                 ; $D011 - Sprite
extended X bits (one bit per sprite)
        jmp @noChangeInExtendedFlag             ; Jump to saving
the X position

@decNoChange                                    ; Not zero X so
we decrement
        dec SPRITE_POS_X,x
@noChangeInExtendedFlag
        txa                                     ; copy X to the
accumulator (sprite number)
        asl                                     ; shift it left
(multiply by 2)
        tay                                     ; save it in Y
(to calculate the register to save to)

        lda SPRITE_POS_X,x                      ; Load our
variable saved X position
        sta VIC_SPRITE_X_POS,y                  ; save it in
$D000 offset by Y to the correct VIC
                                                ; sprite
register


                                                ; Here we
decrement the Sprite delta - we moved
                                                ; a pixel so the
delta goes down by one
        dec SPRITE_POS_X_DELTA,x
        bmi @resetDelta                         ; test for
change to negative
        rts                                     ; if delta is
still > 0 we're done

@resetDelta
```

```
        lda #$07                                ; if delta falls
below 0
        sta SPRITE_POS_X_DELTA,x                ; reset it to
#$07 - one char
        dec SPRITE_CHAR_POS_X,x                 ; delta has
reset - so decrement character position
        rts

#endregion


;=================================================================
==================================
;
MOVE SPRITE RIGHT
;=================================================================
==================================
; Moves a sprite right one pixel and adjusts the extended X bit
if needed to carry to all the way
; across the screen.
; X = the number of the hardware sprite to move - this register
is left intact
;
; NOTE : to move a sprite multiple pixels, this routine must be
called multiple times
;-----------------------------------------------------------------
----------------------------------
#region "MoveSpriteRight"
MoveSpriteRight
        inc SPRITE_POS_X,x                      ; increase
Sprite X position by 1
;        lda SPRITE_POS_X,x                      ; load the
sprite position
        bne @noChangeInExtendedFlag             ; if not #$00
then no change in x flag

        lda BIT_TABLE,x                         ; get the
correct bit to set for this sprite
        eor SPRITE_POS_X_EXTEND                 ; eor in the
extended bit (toggle it on or off)
        sta SPRITE_POS_X_EXTEND                 ; store the new
flags
        sta VIC_SPRITE_X_EXTEND                 ; set it in the
VIC register

@noChangeInExtendedFlag
```

```
        txa                                     ; transfer the
sprite # to A
        asl                                     ; multiply it by
2
        tay                                     ; transfer the
result to Y

        lda SPRITE_POS_X,x                      ; copy the new
position to our variable
        sta VIC_SPRITE_X_POS,y                  ; update the
correct X position register in the VIC

                                                ; Our X position
is now incremented, so delta also
                                                ; increases by 1
        inc SPRITE_POS_X_DELTA,x
        lda SPRITE_POS_X_DELTA,x
        and #%0111                              ; Mask it to 0-7
;        cmp #$08                                ; if it's
crossed over to 8, we reset it to 0
        beq @reset_delta
        rts                                     ; if it hasn't
we're done
@reset_delta
;        lda #$00
        sta SPRITE_POS_X_DELTA,x                ; reset delta to
0 - this means we've crossed a
        inc SPRITE_CHAR_POS_X,x                 ; a character
boundry, so increase our CHAR position
        rts

#endregion

;=================================================================
====================================
;
MOVE SPRITE UP
;=================================================================
====================================
; Up and down have no special considerations to consider - they
wrap at 255
; X = number of hardware sprite to move
;-----------------------------------------------------------------
-----------------------------------
#region "MoveSpriteUp"

MoveSpriteUp
```

```
        dec SPRITE_POS_Y,x                      ; decrement the
sprite position variable

        txa                                     ; copy the
sprite number to A
        asl                                     ; multiply it by
2
        tay                                     ; transfer it to
Y

        lda SPRITE_POS_Y,x                      ; load the
sprite position for this sprite
        sta VIC_SPRITE_Y_POS,y                  ; send it to the
correct VIC register - $D001 + y

                                                ; Y position has
decreased, so our delta decreases
        dec SPRITE_POS_Y_DELTA,x
        bmi @reset_delta                        ; test to see if
it drops to negative
        rts                                     ; if not we're
done
@reset_delta
        lda #$07                                ; reset the
delta to 0
        sta SPRITE_POS_Y_DELTA,x
        dec SPRITE_CHAR_POS_Y,x                 ; if delta
resets, we've crossed a character border
        rts

#endregion

;=================================================================
====================================
;
MOVE SPRITE DOWN
;=================================================================
====================================
; Much the same
; X = number of hardware sprite to move
;-----------------------------------------------------------------
-----------------------------------
#region "MoveSpriteDown"


MoveSpriteDown
```

```
        inc SPRITE_POS_Y,x                      ; increment the
y pos variable for this sprite
                                                ; sans comments
it looks kinda naked.......
        txa
        asl
        tay

        lda SPRITE_POS_Y,x
        sta VIC_SPRITE_Y_POS,y

        inc SPRITE_POS_Y_DELTA,x
        lda SPRITE_POS_Y_DELTA,x
        cmp #$08
        beq @reset_delta
        rts

@reset_delta
        lda #$00
        sta SPRITE_POS_Y_DELTA,x
        inc SPRITE_CHAR_POS_Y,x
        rts

#endregion

;=================================================================
==================================
;
SPRITE TO CHAR POS
;=================================================================
==================================
; Puts a sprite at the position of character X Y. Calculates the
proper sprite coords from the
; screen memory position then sets it there directly.
; The primary use of this is the inital positioning of any
sprite as it will align it with the
; proper delta set up.
;
; PARAM 1 = Character x pos (column)
; PARAM 2 = Character y pos (row)
; X = sprite number
;-----------------------------------------------------------------
---------------------------------
#region "SpriteToCharPos"

SpriteToCharPos
```

```
        lda BIT_TABLE,x                 ; Lookup the bit for
this sprite number (0-7)
        eor #$ff                        ; flip all bits (invert
the byte %0001 would become %1110)
        and SPRITE_POS_X_EXTEND         ; mask out the X extend
bit for this sprite
        sta SPRITE_POS_X_EXTEND         ; store the result back
- we've erased just this sprites bit
        sta VIC_SPRITE_X_EXTEND         ; store this in the VIC
register for extended X bits

        lda PARAM1                      ; load the X pos in
character coords (the column)
        sta SPRITE_CHAR_POS_X,x         ; store it in the
character X position variable
        cmp #30                         ; if X is less than 30,
no need set the extended bit
        bcc @noExtendedX

        lda BIT_TABLE,x                 ; look up the the bit
for this sprite number
        ora SPRITE_POS_X_EXTEND         ; OR in the X extend
values - we have set the correct bit
        sta SPRITE_POS_X_EXTEND         ; Store the results back
in the X extend variable
        sta VIC_SPRITE_X_EXTEND         ; and the VIC X extend
register

@noExtendedX
                                        ; Setup our Y register
so we transfer X/Y values to the
                                        ; correct VIC register
for this sprite
        txa                             ; first, transfer the
sprite number to A
        asl                             ; multiply it by 2
(shift left)
        tay                             ; then store it in Y
                                        ; (note : see how VIC
sprite pos registers are ordered
                                        ;  to understand why I'm
doing this)

        lda PARAM1                      ; load in the X Char
position
        asl                             ; 3 x shift left =
multiplication by 8
```

```
        asl
        asl
        clc
        adc #24 - SPRITE_DELTA_OFFSET_X ; add the edge of screen
(24) minus the delta offset
                                        ; to the rough center 8
pixels (1 char) of the sprite

        sta SPRITE_POS_X,x              ; save in the correct
sprite pos x variable
        sta VIC_SPRITE_X_POS,y          ; save in the correct
VIC sprite pos register


        lda PARAM2                      ; load in the y char
position (rows)
        sta SPRITE_CHAR_POS_Y,x         ; store it in the
character y pos for this sprite
        asl                             ; 3 x shift left =
multiplication by 8
        asl
        asl
        clc
        adc #50 - SPRITE_DELTA_OFFSET_Y ; add top edge of screen
(50) minus the delta offset
        sta SPRITE_POS_Y,x              ; store in the correct
sprite pos y variable
        sta VIC_SPRITE_Y_POS,y          ; and the correct VIC
sprite pos register

        lda #0
        sta SPRITE_POS_X_DELTA,x        ;set both x and y delta
values to 0 - we are aligned
        sta SPRITE_POS_Y_DELTA,x        ;on a character border
(for the purposes of collisions)
        rts

#endregion

;=================================================================
==================================
;
SHIFT SPRITE DELTA
;=================================================================
==================================
; Shift the sprite delta in a direction (left,right,up,down)
without moving the actual sprite
```

```
;-------------------------------------------------------------
-----------------------------------
; A = direction (same as SCROLL_DIRECTION : 1 = Right / 2 = Left
/ 3 = up / 4 = down)
; X = sprite to shift
;-------------------------------------------------------------
-----------------------------------

#region "ShiftSpriteDelta"

ShiftDeltaRight
        inc SPRITE_POS_X_DELTA,x
        lda SPRITE_POS_X_DELTA,x
        and #%00000111
        sta SPRITE_POS_X_DELTA,x
        rts

ShiftDeltaLeft
        dec SPRITE_POS_X_DELTA,x                ; Decrement the
Delta X for the sprite
        lda SPRITE_POS_X_DELTA,x                ; load Delta
        and #%00000111                           ; adjust it to
a 0-7 value
        sta SPRITE_POS_X_DELTA,x                ; save the value
        rts


#endregion

;=============================================================
====================================
;
SET SPRITE IMAGE
;=============================================================
====================================
; Sets the sprite image for a hardware sprite, and sets up its
pointers for both screens
;-------------------------------------------------------------
-----------------------------------
; A = Sprite image number
; X = Hardware sprite
;
; Leaves registers intact
;-------------------------------------------------------------
-----------------------------------
#region "SetSpriteImage"
SetSpriteImage
```

```
        pha

        clc
        adc #SPRITE_BASE                        ; Sprite image =
image num + base
        sta SPRITE_POINTER_BASE1,x
        sta SPRITE_POINTER_BASE2,x

        pla

        rts

#endregion

;=================================================================
==================================
;
SPRITE ANIMATION ROUTINES
;=================================================================
==================================

;=================================================================
==================================
;
INIT SPRITE ANIM
;-----------------------------------------------------------------
----------------------------------
;
; Setup and initialize a sprites animations
;
; X = Sprite number
; ZERO_PAGE_POINTER_1 = animation list address
;
; Modifies A,Y
;-----------------------------------------------------------------
----------------------------------
#region "InitSpriteAnim"
InitSpriteAnim

        lda #1                          ; Reset Anim counter to
first frame
        sta SPRITE_ANIM_COUNT,x

        txa                             ; copy sprite number to
A
```

```
        asl                             ; multiply it by 2 (so
we can index a word)
        tay                             ; transfer result to Y

        lda ZEROPAGE_POINTER_1          ; Store the address for
the animlist
        sta SPRITE_ANIMATION,y          ; in the correct 'slot'
for this sprite
        lda ZEROPAGE_POINTER_1 + 1
        sta SPRITE_ANIMATION + 1,y

        ldy #0                          ; First byte in the list
is the timer
        lda (ZEROPAGE_POINTER_1),y      ; fetch it
        sta SPRITE_ANIM_TIMER,x         ; store it in this
sprites timer slot


        iny                             ; increment Y to 1 - the
first anim frame
        lda (ZEROPAGE_POINTER_1),y      ; load it

                                        ; right now we have the
sprite image in A
                                        ; and the sprite number
in X, which is exactly what
                                        ; what we need to use
SetSpriteImage
        jsr SetSpriteImage

        cpx #0                          ; if the sprite = 0,
then we are setting the player sprite
        beq @secondSprite               ; the player uses 2
sprites

        rts


@secondSprite
                                        ; We don't need to set
all the info for this sprite
        inx                             ; increment X to the
next hardware sprite

        clc                             ; add one to the current
anim frame number
```

```
        adc #1                          ; because our background
sprite is the next frame.

        jsr SetSpriteImage              ; Now set the sprite
image

        rts

#endRegion
;================================================================
==================================
;
ANIMATE SPRITE
;----------------------------------------------------------------
----------------------------------
; Animate an individual sprite. Taking data from it's Anim List,
updating it's variables
; and updating it's animation frame
;
; It can currently handle loop, play once, and ping-pong
animations
;
; X = sprite number to animate
;
; Modifies :  Y, ZEROPAGE_POINTER_1
;
;----------------------------------------------------------------
----------------------------------
#region "AnimateSprites"
AnimateSprite
        ;-------------------------------------------------------
------------------
        ; First - do we need to animate this sprite at all?
Check the TIMER

        lda SPRITE_ANIM_PLAY                    ; return if anim
paused
        bne @start
        rts
@start

        lda SPRITE_ANIM_TIMER,x                 ; load the anim
timer mask
        bne @timerCheck                         ; is our timer
turned off? (set to 0)

        rts
```

```
@timerCheck
        and #%00001111                          ; mask out the
upper half byte (extra info)
        and TIMER                               ; and the mask
against the timer
        beq @update                             ; if the result
isn't 0 - return
        rts
@update
        ;-------------------------------------------------------
------------------
        ; First we need to fetch this sprites anim list.

        txa                                     ; put the sprite
number in A
        asl                                     ; multiply by 2
(to lookup a word)
        tay                                     ; store the
result in Y

        lda SPRITE_ANIMATION,y                  ; fetch the
address to the sprites Anim List
        sta ZEROPAGE_POINTER_1                  ; and store it
in ZEROPAGE_POINTER_1
        lda SPRITE_ANIMATION + 1,y
        sta ZEROPAGE_POINTER_1 + 1

        ;-------------------------------------------------------
------------------
        ; Next - increment (or decrement for PING PONG) the
animation counter

        lda SPRITE_ANIM_TIMER,x                 ; check for the
MSB in the timer, if it's set
        bpl @incTimer                           ; we count down
instead

        dec SPRITE_ANIM_COUNT,x                 ; decrement anim
counter and continue as normal
        jmp @zerocheck
@incTimer
        inc SPRITE_ANIM_COUNT,x                 ; increment the
counter
@zerocheck
        lda SPRITE_ANIM_COUNT,x                 ; fetch it so we
can do an end test
```

```
        beq @resetPong                          ; it should
NEVER be zero, unless it's a PING_PONG
                                                ; anim returning
back to zero as the first bytes
                                                ; is the anim
timer mask

        tay                                     ; use as an
index to load the current frame from the
        lda (ZEROPAGE_POINTER_1),y              ; animlist


        bpl @setImage                           ; Test to make
sure the frame is positive (0-128)
                                                ; if it isn't,
and the MSB is set, then it's the
                                                ; end byte,
which we use to tell us the type of
                                                ; anim (and what
to do when we reach the end)

        cmp #TYPE_LOOP                          ; Check to see
what type of anim this is
        beq @resetLoop

        cmp #TYPE_PING_PONG
        beq @resetPing
        ;-------------------------------------------------------
---PLAY_ONCE_ANIM
                                                ; PlayOnce Anim
doesn't get reset
        lda #0                                  ; We set the
anim timer to 0, so it never updates
        sta SPRITE_ANIM_TIMER,x
        lda #$FF                                ; We set the
count to -1 ($FF) as something our
        sta SPRITE_ANIM_COUNT,x                 ; code can check
to see if a once only is finished
        rts
        ;-------------------------------------------------------
PING_PONG_ANIM
                                                ; We've reached
the end of the animlist
@resetPing
        lda SPRITE_ANIM_TIMER,x
        eor #%10000000                          ; flip bit 7 in
the timer to start 0 check
```

```
                                                ; and turn on
decrementing
        sta SPRITE_ANIM_TIMER,x                 ; save the new
counter

        lda SPRITE_ANIM_COUNT,x
        sec
        sbc #2                                  ; subtract 2
from the anim counter
        sta SPRITE_ANIM_COUNT,x                 ; and save the
result
        tay
        lda (ZEROPAGE_POINTER_1),y              ; lookup the new
sprite frame
                                                ; A now has the
sprite image
                                                ; X has the
sprite number
                                                ; so we set the
image(s)
        jmp @setImage

@resetPong                                      ; The reverse of
the above
        lda SPRITE_ANIM_TIMER,x
        eor #%10000000                          ; flip bit 7
(MSB) to count forwards
        sta SPRITE_ANIM_TIMER,x                 ; save the new
timer

        lda #2                                  ; skip to frame
2
        sta SPRITE_ANIM_COUNT,x                 ; save the new
counter
        tay
        lda (ZEROPAGE_POINTER_1),y              ; load the new
sprite image
        jmp @setImage
        ;-------------------------------------------------------
------LOOP_ANIM
@resetLoop
        lda #1                                  ; reset the
counter to the first frame
        sta SPRITE_ANIM_COUNT,x
        tay                                     ; now fetch the
first frame
        lda (ZEROPAGE_POINTER_1),y
```

```
@setImage

        ;-------------------------------------------------------
------------
        ; Set the sprite to it's new image - A currently
contains the count
        ; ZEROPAGE_POINTER_1 has the address of the sprites
animlist

                                                ; A = correct
sprite image, X = sprite number,
                                                ; so we're ready
to set our new sprite image
        jsr SetSpriteImage

        cpx #0                                  ; if sprite is 0
, it's the player sprite
        beq @setSecondSprite                    ; so we need to
set a second sprite
        rts

@setSecondSprite
        clc
        adc #1                                  ; add one to the
frame
        inx                                     ; add one to the
sprite number

        jsr SetSpriteImage
        dex

        rts
#endregion
;-------------------------------------------------------------
-----------------------------------
;=============================================================
===================================
;
DATA AND TABLES
;=============================================================
===================================
;-------------------------------------------------------------
-----------------------------------
;
SPRITE POINTER TABLES
```

```
;------------------------------------------------------------
-----------------------------------

                                                ; Lookup tables
for setting Sprite Pointers for the
                                                ; correct
screens

SPRITE_POINTER_BASE1 = SCREEN1_MEM + $3f8
SPRITE_POINTER_BASE2 = SCREEN2_MEM + $3f8


;------------------------------------------------------------
-----------------------------------
;============================================================
===================================
;
SPRITE HANDLING DATA
;============================================================
===================================

SPRITE_IS_ACTIVE
        byte $00,$00,$00,$00,$00,$00,$00,$00
                                                ;
Hardware sprite X position
SPRITE_POS_X
        byte $00,$00,$00,$00,$00,$00,$00,$00
                                                ; Delta
X position (0-7) - within a char
SPRITE_POS_X_DELTA
        byte $00,$00,$00,$00,$00,$00,$00,$00

SPRITE_CHAR_POS_X                               ; Char
pos X - sprite position in character
        byte $00,$00,$00,$00,$00,$00,$00,$00    ; coords
(0-40)
SPRITE_DELTA_TRIM_X
        byte $00,$00,$00,$00,$00,$00,$00,$00    ; Trim
delta for better collisions

SPRITE_POS_X_EXTEND                             ;
extended flag for X positon > 255
        byte $00                                ; bits
0-7 correspond to sprite numbers
```

```
SPRITE_POS_Y                                                    ;
Hardware sprite Y position
        byte $00,$00,$00,$00,$00,$00,$00,$00
SPRITE_POS_Y_DELTA
        byte $00,$00,$00,$00,$00,$00,$00,$00
SPRITE_CHAR_POS_Y
        byte $00,$00,$00,$00,$00,$00,$00,$00

SPRITE_DIRECTION_X
        byte $00,$00,$00,$00,$00,$00,$00,$00    ; Direction of
the sprite (-1 0 1)
SPRITE_DIRECTION_Y
        byte $00,$00,$00,$00,$00,$00,$00,$00

SPRITE_ANIM_TIMER
        byte $00,$00,$00,$00,$00,$00,$00,$00    ; Timing and
playback direction for current anim
SPRITE_ANIM_COUNT
        byte $00,$00,$00,$00,$00,$00,$00,$00    ; Position in
the anim list
SPRITE_ANIM_PLAY
        byte $00,$00,$00,$00,$00,$00,$00,$00    ; Currently
animated or paused? (TO DO - single byte and use BIT_TABLE)

                                                ; Pointer to
current animation table
SPRITE_ANIMATION
        word $0000,$0000,$0000,$0000
        word $0000,$0000,$0000,$0000
;================================================================
==================================
;
SPRITE ANIMATION TABLES
;================================================================
==================================
; Anims are held in a block of data containing info and a list
of anim frames.
; byte 0 = Timing mask:
;          The lower 4 bits contain a mask we will and with the
master timer, if the result is
;          is 0 we will go to the next frame
;          valid values are those that use all bits (1,3,5,7,15)
.
;          Bit 7 is used for the direction of the animation (in
ping pong anims). if we set it
;          we can then use bmi (Branch MInus) to see if we count
backwards instead of forwards
```

```
;
; byte 1 to end = sprite numbers for the animation frames
;
; The last byte both terminates the animation, and also shows
what should be done at the end
; of it.  On a loop we would reset to 0, on a play once anim we
would end, on a ping pong type
; it will start counting backwards
;-----------------------------------------------------------------
-----------------------------------
;-------------------------------------------------------
;                                          ANIMATION TYPES
TYPE_LOOP = $FF
TYPE_PLAY_ONCE = $FE
TYPE_PING_PONG = $FD
;-------------------------------------------------------

ANIM_TEST                                          ; numbers 0 to 9
        byte %00001111
        byte 40,41,42,43,44,45,46,47,48,49
        byte TYPE_PING_PONG

ANIM_PLAYER_WALK_R                                 ; 7 frame
animation walk right
        byte %00000011
        byte 0,2,4,6,8,10,12
        byte TYPE_LOOP

ANIM_PLAYER_WALK_L
        byte %00000011
        byte 14,16,18,20,22,24,26
        byte TYPE_LOOP

ANIM_PLAYER_IDLE                                   ; Player idle
animation
        byte %00000111
        byte 28,28,30,32
        byte TYPE_PING_PONG

ANIM_PLAYER_FALL
        byte %0001111
        byte 34
        byte TYPE_PLAY_ONCE

ANIM_CLIMB_ROPE_UP
        byte %0000111
        byte 36,38
```

```
byte TYPE_LOOP
```