```
;=================================================================
==================================
;
CORE ROUTINES
;=================================================================
==================================
; Core routines for the framework
- Peter 'Sig' Hewett
;
2016
;-----------------------------------------------------------------
----------------------------------
        ;-------------------------------------------------------
----------------------------------
        ;
VBL WAIT
        ;-------------------------------------------------------
----------------------------------
        ; Wait for the raster to reach line $f8 - if it's aleady
there, wait for
        ; the next screen blank. This prevents mistimings if the
code runs too fast
#region "WaitFrame"
WaitFrame
        lda VIC_RASTER_LINE         ; fetch the current raster
line
        cmp #$F8                ; wait here till line #$f8
        beq WaitFrame

@WaitStep2
        lda VIC_RASTER_LINE
        cmp #$F8
        bne @WaitStep2
        rts
#endregion
        ;-------------------------------------------------------
----------------------------------
        ;
UPDATE TIMERS
        ;-------------------------------------------------------
----------------------------------
        ; 2 basic timers - a fast TIMER that is updated every
frame,
        ; and a SLOW_TIMER updated every 16 frames
        ;-------------------------------------------------------
----------------
#region "UpdateTimers"
```

```
UpdateTimers
        inc TIMER                       ; increment TIMER by 1
        lda TIMER
        and #$0F                        ; check if it's equal to
16
        beq @updateSlowTimer            ; if so we update
SLOW_TIMER
        rts

@updateSlowTimer
        inc SLOW_TIMER                  ; increment slow timer
        rts


#endregion
        ;-------------------------------------------------------
-----------------------------------
        ;
READ JOY 2
        ;-------------------------------------------------------
-----------------------------------
        ; Trying this a different way this time.  Rather than
hitting the joystick registers then
        ; doing something every time - The results will be
stored in JOY_X and JOY_Y with values
        ; -1 to 1 , with 0 meaning 'no input' - I should be able
to just add this to a sprite for a
        ; simple move, while still being able to do an easy
check for more complicated movement
        ; later on
        ;-------------------------------------------------------
-----------------------------------
#region "ReadJoystick"

ReadJoystick

        lda #$00                        ; Reset JOY X and Y
variables
        sta JOY_X
        sta JOY_Y

@testUp                                 ; Test for Up pressed
        lda #%00000001                  ; Mask for bit 0
        bit JOY_2                        ; test bit 0 for press
        bne @testDown
        lda #$FF                        ; set JOY_Y to -1 ($FF)
        sta JOY_Y
```

```
        jmp @testLeft                   ; Can't be up AND down

@testDown                               ; Test for Down
        lda #%00000010                  ; Mask for bit 1
        bit JOY_2
        bne @testLeft
        lda #$01                        ; set JOY_Y to 1 ($01)
        sta JOY_Y
@testLeft                               ; Test for Left
        lda #%00000100                  ; Mask for bit 2
        bit JOY_2
        bne @testRight
        lda #$FF
        sta JOY_X
        rts                             ; Can't be left AND
right - no more tests

@testRight                              ; Test for Right
        lda #%00001000                  ; Mask for bit 3
        bit JOY_2
        bne @done
        lda #$01
        sta JOY_X
        rts                             ; no more checks

@done                                  ; Nothing pressed
         rts

#endregion


        ;-------------------------------------------------------
-----------------------------------
        ;
JOYSTICK BUTTON PRESSED
        ;-------------------------------------------------------
-----------------------------------
        ; Notifies the state of the fire button on JOYSTICK 2.
        ; BUTTON_ACTION is set to one on a single press (that is
when the button is released)
        ; BUTTON_PRESSED is set to 1 while the button is held
down.
        ; So either a long press, or a single press can be
accounted for.
        ; TODO I might put a 'press counter' in here to test how
long the button is down for..
```

```
        ;-------------------------------------------------------
-----------------------------------
#region "JoyButton"

JoyButton

        lda #1                                  ; checks for a
previous button action
        cmp BUTTON_ACTION                       ; and clears it
if set
        bne @buttonTest

        lda #0
        sta BUTTON_ACTION

@buttonTest
        lda #$10                                 ; test bit #4
in JOY_2 Register
        bit JOY_2
        bne @buttonNotPressed

        lda #1                                  ; if it's
pressed - save the result
        sta BUTTON_PRESSED                      ; and return -
we want a single press
        rts                                     ; so we need to
wait for the release

@buttonNotPressed

        lda BUTTON_PRESSED                      ; and check to
see if it was pressed first
        bne @buttonAction                       ; if it was we
go and set BUTTON_ACTION
        rts

@buttonAction
        lda #0
        sta BUTTON_PRESSED
        lda #1
        sta BUTTON_ACTION

        rts

#endregion
```

```
        ;-------------------------------------------------------
-----------------------------------
        ;
COPY CHARACTER SET
        ;-------------------------------------------------------
-----------------------------------
        ; Copy the custom character set into the VIC Memory Bank
(2048 bytes)
        ; ZEROPAGE_POINTER_1 = Source
        ; ZEROPAGE_POINTER_2 = Dest
        ;
        ; Returns A,X,Y and PARAM2 intact
        ;-------------------------------------------------------
-----------------------------------
#region "CopyChars"

CopyChars

        saveRegs

        ldx #$00                                ; clear X, Y, A
and PARAM2
        ldy #$00
        lda #$00
        sta PARAM2
@NextLine
        lda (ZEROPAGE_POINTER_1),Y              ; copy from
source to target
        sta (ZEROPAGE_POINTER_2),Y

        inx                                     ; increment x /
y
        iny
        cpx #$08                                ; test for next
character block (8 bytes)
        bne @NextLine                           ; copy next line
        cpy #$00                                ; test for edge
of page (256 wraps back to 0)
        bne @PageBoundryNotReached

        inc ZEROPAGE_POINTER_1 + 1              ; if reached 256
bytes, increment high byte
        inc ZEROPAGE_POINTER_2 + 1              ; of source and
target

@PageBoundryNotReached
```

```
        inc PARAM2                                      ; Only copy 254
characters (to keep irq vectors intact)
        lda PARAM2                                      ; If copying to
F000-FFFF block
        cmp #255
        beq @CopyCharactersDone
        ldx #$00
        jmp @NextLine

@CopyCharactersDone

        restoreRegs

        rts
#endregion

        ;-------------------------------------------------------------
        ;
COPY SPRITE DATA
        ;-------------------------------------------------------------
        ; Copies sprites from ZEROPAGE_POINTER_1 to
ZEROPAGE_POINTER_2
        ; Sprites are copied in sets of 4
        ;-------------------------------------------------------------
;#region "CopySprites"

;CopySprites
;        ldy #$00
;        ldx #$00

;        ;lda #<SPRITE_MEM
;        ;sta ZEROPAGE_POINTER_2
;        ;lda #>SPRITE_MEM
;        ;sta ZEROPAGE_POINTER_2 + 1

;        loadPointer ZEROPAGE_POINTER_2, SPRITE_MEM


;@SpriteLoop
;        lda (ZEROPAGE_POINTER_1),Y
;        sta (ZEROPAGE_POINTER_2),Y
;        iny
;        bne @SpriteLoop
;        inx
```

```
;         inc ZEROPAGE_POINTER_1 + 1
;         inc ZEROPAGE_POINTER_2 + 1
;         cpx #NUMBER_OF_SPRITES_DIV_4
;         bne @SpriteLoop

;         rts

;#endregion

;-------------------------------------------------------------------
-----------------------------------
;
8bit * 16bit = 16bit Multiply
;-------------------------------------------------------------------
-----------------------------------
; (Credit to White Flame for the code this is based on - much
better than my own attempt)
;
; Multiplies num1 * num2 and stores the result in W_PARAM1
;
; num1 LOW      = W_PARAM1
; num1 HIGH     = W_PARAM1 + 1
; num3 8 bit    = PARAM3
;
; Returns
;       WPARAM1 - Low byte / High byte of result
;-------------------------------------------------------------------
-----------------------------------
#region "Multiply16"
Multiply16

        saveRegs                ; Save registers A,X,Y
        lda PARAM3
        pha                     ; Save PARAM3

        lda #$00
        tay                     ; clear Y with 0
;       sty PARAM2              ; uncomment for 8bit * 8bit -
clears the high byte
        beq @enterLoop

@doAdd
        clc                     ; clear carry flag
        adc WPARAM1            ; num1 (Low byte) addition
        tax                     ; save result in X

        tya                     ; transfer high byte to A
```

```
        adc WPARAM1 + 1         ; num1 (high byte) addition
        tay                      ; Save result in Y
        txa                      ; restore low byte to A

@loop
        asl WPARAM1             ; num1 low byte * 2
        rol WPARAM1 + 1

@enterLoop                       ; accumulating multipy entry
point (A = Low, Y = High)
        lsr PARAM3
        bcs @doAdd
        bne @loop

                                 ; At this point A and X contain
Low Byte result
                                 ; Y contains High Byte result

        sta WPARAM1             ; Save results in W_PARAM1 as 16
bit word
        sty WPARAM1 + 1

        pla                      ; restore PARAM3
        sta PARAM3
        restoreRegs              ; restore registers A,X,Y
        rts

#endregion


;=================================================================
==============================
;
QUICK PSUEDORANDOM 8 BIT GEN
;=================================================================
==============================
; Very quick n dirty
;-----------------------------------------------------------------
------------------------------
#region "Random8Bit"
Random8Bit

      lda seed
      asl
      bcc @no_eor
      eor #$cf
@no_eor
```

```
        sta seed
        rts


seed
        byte 0

#endregion
```