```
;=============================================================
===============
;
SCREEN ROUTINES
;=============================================================
===============
;
Peter 'Sig' Hewett
;
- 2016/2017
;-------------------------------------------------------------
---------------
; Routines to draw on or modify the screen
;-------------------------------------------------------------
---------------
;-------------------------------------------------------------
---------------
;
SWAP SCREENS
;-------------------------------------------------------------
---------------
; Exchange the front and backbuffer screens
;-------------------------------------------------------------
---------------
#region "SwapScreens"
SwapScreens
        lda CURRENT_SCREEN + 1              ; load hi byte of
current screen
        cmp #>SCREEN2_MEM
        beq @screen2

        loadPointer CURRENT_SCREEN, SCREEN2_MEM
        loadPointer CURRENT_BUFFER, SCREEN1_MEM
        rts

@screen2
        loadPointer CURRENT_SCREEN, SCREEN1_MEM
        loadPointer CURRENT_BUFFER, SCREEN2_MEM
        rts

#endregion
;=============================================================
===================================
;
FETCH PLAYFIELD LINE ADDRESS
;=============================================================
===================================
```

```
; A helper routine to return the line address for the current
front screen only. A cut back version
; of FetchLineAddress for faster use with sprite/character
collisions it also uses the Y register
; instead of the X as that is tied up in our collision routines
to hold the sprite number
;
; Y = line number
; Returns : ZEROPAGE_POINTER_1 = screen line address
; Modifies A
;-----------------------------------------------------------------
-----------------------------------
#region "FetchPlayfieldLineAddress"
FetchPlayfieldLineAddress
        lda CURRENT_SCREEN + 1          ; load HI byte of curren
screen address
        cmp #>SCREEN1_MEM               ; compare it to the HI
byte of SCREEN1_MEM
        beq @screen1                    ; if it's equal - it's
screen1
                                        ; otherwise it's screen2

        lda SCREEN2_LINE_OFFSET_TABLE_LO,y      ; Use Y to
lookup the address and save it in
        sta ZEROPAGE_POINTER_1                  ;
ZEROPAGE_POINTER_1
        lda SCREEN2_LINE_OFFSET_TABLE_HI,y
        sta ZEROPAGE_POINTER_1 + 1
        rts

@screen1
        lda SCREEN1_LINE_OFFSET_TABLE_LO,y      ; Use Y to
lookup the address and save it in
        sta ZEROPAGE_POINTER_1                  ;
ZEROPAGE_POINTER_1
        lda SCREEN1_LINE_OFFSET_TABLE_HI,y
        sta ZEROPAGE_POINTER_1 + 1
        rts
#endregion
;-----------------------------------------------------------------
----------------
;
FETCH LINE ADDRESS
;-----------------------------------------------------------------
----------------
; A helper routine to return the line address for the correct
screen to draw to
```

```
; Given the screen base in WPARAM1, and the line in X (Y coord)
we test
; the high byte in WPARAM1 and use the correct lookup table to
get the line
; address, returning it in ZEROPAGE_POINTER_1

; An additional 'jump in' point "FetchScreenLineAddress" can be
used that will
; only consider the CURRENT_SCREEN pointer, likewise
"FetchBufferLineAddress"
; will jump in and substitute the current buffer.
;
; WPARAM1 - BASE ADDRESS OF SCREEN (1,2,Score)
; X - Line required
;
; returns ZEROPAGE_POINTER_1
;
; Modifies A
;
;-----------------------------------------------------------------
----------------
#region "FetchLineAddress"
FetchLineAddress

        lda WPARAM1 + 1
        jmp detectScreen

FetchScreenLineAddress
        lda CURRENT_SCREEN + 1
        jmp detectScreen

FetchBufferLineAddress
        lda CURRENT_BUFFER + 1

detectScreen
        cmp #>SCREEN1_MEM
        beq @screen1
        cmp #>SCREEN2_MEM
        beq @screen2
        cmp #>SCORE_SCREEN
        beq @score
                                        ; if none of the above,
it will default to Screen1

@screen1
        lda SCREEN1_LINE_OFFSET_TABLE_LO,x
        sta ZEROPAGE_POINTER_1
```

```
        lda SCREEN1_LINE_OFFSET_TABLE_HI,x
        sta ZEROPAGE_POINTER_1 + 1
        rts
@screen2
        lda SCREEN2_LINE_OFFSET_TABLE_LO,x
        sta ZEROPAGE_POINTER_1
        lda SCREEN2_LINE_OFFSET_TABLE_HI,x
        sta ZEROPAGE_POINTER_1 + 1
        rts
@score
        lda SCORE_LINE_OFFSET_TABLE_LO,x
        sta ZEROPAGE_POINTER_1
        lda SCORE_LINE_OFFSET_TABLE_HI,x
        sta ZEROPAGE_POINTER_1 + 1
        rts

#endRegion
;-------------------------------------------------------------------
-----------------------------------
;
DISPLAY BYTE DATA
;-------------------------------------------------------------------
-----------------------------------
; Displays the data stored in a given byte on the screen as
readable text in hex format (0-F)

; X = screen line                 - Yes, this is a little arse-
backwards (X and Y) but I don't think
; Y = screen column                 addressing modes allow me to
swap them around
; A = byte to display
; MODIFIES : ZEROPAGE_POINTER_1, ZEROPAGE_POINTER_3, PARAM4
;-------------------------------------------------------------------
-----------------------------------
#region "DisplayByte"
DisplayByte

        sta PARAM4                                          ; store
the byte to display in PARAM4

        jsr FetchLineAddress

        lda COLOR_LINE_OFFSET_TABLE_LO,x        ; fetch line
address for color
        sta ZEROPAGE_POINTER_3
        lda COLOR_LINE_OFFSET_TABLE_HI,x
        sta ZEROPAGE_POINTER_3 + 1
```

```
        lda PARAM4                                      ; load
the byte to be displayed
        and #$0F
        clc                                             ; mask
for the lower half (0-F)
        adc #$30                                        ; add
$30 (48) to display character set
                                                        ;
numbers

        clc                                             ; clear
carry flag
        cmp #$3A                                        ; less
than the code for A (10)?
        bcc @writeDigit                                 ; Go to
the next digit

        sec
        sbc #$39                                        ; if so
we set the character code back to
                                                        ;
display A-F ($01 - $0A)

@writeDigit
        iny                                             ;
increment the position on the line
        sta (ZEROPAGE_POINTER_1),y                      ; write
the character code
        lda #COLOR_YELLOW                               ; set
the color to white
        sta (ZEROPAGE_POINTER_3),y                      ; write
the color to color ram

        dey                                             ;
decrement the position on the line
        lda PARAM4                                      ; fetch
the byte to DisplayText
        and #$F0                                        ; mask
for the top 4 bits (00 - F0)
        lsr                                             ; shift
it right to a value of 0-F
        lsr
        lsr
        lsr
        adc #$30                                        ; from
here, it's the same
```

```
        clc
        cmp #$3A                                        ; check
for A-F
        bcc @lastDigit
        sbc #$39

@lastDigit
        sta (ZEROPAGE_POINTER_1),y                      ; write
character and color
        lda #COLOR_YELLOW
        sta (ZEROPAGE_POINTER_3),y

        rts

#endregion
;------------------------------------------------------------------
----------------------------------

;------------------------------------------------------------------
----------------
;
DISPLAY TEXT
;------------------------------------------------------------------
----------------
; Display a line of text                '@' ($00) is the end of
text characters
;                                       '/' ($2f) is the line
break character
;
; ZEROPAGE_POINTER_1 = pointer to text data
; PARAM1 = X
; PARAM2 = Y
; PARAM3 = Color
;
; Modifies ZEROPAGE_POINTER_2 and ZEROPAGE_POINTER_3
;
; Note all text should be in lower case : byte 'hello world@' or
byte 'hello world',0
;------------------------------------------------------------------
----------------
#region "DisplayText"

DisplayText

        ldx PARAM2
        lda SCORE_LINE_OFFSET_TABLE_LO,x
```

```
        sta ZEROPAGE_POINTER_2
        lda SCORE_LINE_OFFSET_TABLE_HI,x
        sta ZEROPAGE_POINTER_2 + 1


        lda COLOR_LINE_OFFSET_TABLE_LO,x          ; Fetch the
address for the line in color ram
        sta ZEROPAGE_POINTER_3
        lda COLOR_LINE_OFFSET_TABLE_HI,x
        sta ZEROPAGE_POINTER_3 + 1


                                          ; add the X offset to
the destination address
        lda ZEROPAGE_POINTER_2
        clc
        adc PARAM1
        sta ZEROPAGE_POINTER_2
        lda ZEROPAGE_POINTER_2 + 1
        adc #0
        sta ZEROPAGE_POINTER_2 + 1
                                          ; Same for color ram
        lda ZEROPAGE_POINTER_3
        clc
        adc PARAM1
        sta ZEROPAGE_POINTER_3
        lda ZEROPAGE_POINTER_3 + 1
        adc #0
        sta ZEROPAGE_POINTER_3 + 1
                                          ; Start the write for
this line
        ldy #0
@inlineLoop
        lda (ZEROPAGE_POINTER_1),y
        cmp #00
        beq @endMarkerReached
        cmp #$2F
        beq @lineBreak
        sta (ZEROPAGE_POINTER_2),y
        lda PARAM3
        sta (ZEROPAGE_POINTER_3),y
        iny
        jmp @inLineLoop

@lineBreak
        iny
        tya
```

```
        clc
        adc ZEROPAGE_POINTER_1
        sta ZEROPAGE_POINTER_1
        lda #0
        adc ZEROPAGE_POINTER_1 + 1
        sta ZEROPAGE_POINTER_1 + 1

        inc PARAM2
        jmp DisplayText

@endMarkerReached
        rts

#endregion
;------------------------------------------------------------------
----------------
;
CLEAR SCREEN
;------------------------------------------------------------------
----------------
; As these routines were originally written for speed and
simplicity. The
; Best way to redo them for multi-screens would be to split them
into seperate
; routines for each screen and one for color. Wpuld probably be
smaller than a
; one routine for all screens approach too.
;
; Clears the screen using a chosen character.
; A = Character/Color to clear the screen with
;
; Modifies X
;------------------------------------------------------------------
----------------
#region "ClearScreen0"

ClearScreen1
        ldx #$00
@clearLoop
        sta SCREEN1_MEM,x
        sta SCREEN1_MEM + 250,x
        sta SCREEN1_MEM + 500,x                 ; Game screen
only goes to 720
        sta SCREEN1_MEM + 750,x
        inx
        cpx #250
        bne @clearLoop
```

```
        rts
#endregion

#region "ClearScreen1"
ClearScreen2
        ldx #$00
@clearLoop
        sta SCREEN2_MEM,x
        sta SCREEN2_MEM + 250,x
        sta SCREEN2_MEM + 500,x                 ; Game screen
only goes to 720
        sta SCREEN2_MEM + 750,x
        inx
        cpx #250
        bne @clearloop

        rts
#endregion

#region "ClearScoreScreen"
ClearScoreScreen
        ldx #$00
@clearLoop
        sta SCORE_SCREEN,x
        sta SCORE_SCREEN + 250,x
        sta SCORE_SCREEN + 500,x
        sta SCORE_SCREEN + 750,x
        inx
        cpx #250
        bne @clearloop

        rts
#endRegion

#region "ClearColorRam"
ClearColorRam
        ldx #$00
@clearLoop
        sta COLOR_MEM,x
        sta COLOR_MEM + 250,x
        sta COLOR_MEM + 500,x
        sta COLOR_MEM + 750,x
        inx
        cpx #250
        bne @clearLoop

        rts
```

```
#endregion

;-----------------------------------------------------------------
;                                                          COPY
TO BUFFER (SLOW)
;-----------------------------------------------------------------
; NOTE : Don't use this for scrolling. Use the unrolled version
in scrolling.asm
; this is just for setup purposes. It takes the current front
screen and copys
; it to the buffer
;-----------------------------------------------------------------
#region CopyToBuffer
CopyToBuffer

@copy_screen1
        ldx #$00
@loop1
        lda SCREEN1_MEM,x
        sta SCREEN2_MEM,x

        lda SCREEN1_MEM + 250,x
        sta SCREEN2_MEM + 250,x

        lda SCREEN1_MEM + 500,x
        sta SCREEN2_MEM + 500,x

        lda SCREEN1_MEM + 750,x                 ; Game screen
only goes to 720
        sta SCREEN2_MEM + 750,x

        inx
        cpx #250
        bne @loop1

        rts

#endregion
;-----------------------------------------------------------------
;                                                          DRAW
VERTICAL LINE
;-----------------------------------------------------------------
```

```
; DrawVLine - draws a vertical line with a specified color and
character.
;             it's not optimized or terribly pretty but it lets
you reuse your
;             PARAM variables to draw another line straight away
;
; PARAM1 = start X
; PARAM2 = start Y
; PARAM3 = end Y
; PARAM4 = character
; PARAM5 = color
;-----------------------------------------------------------------
----------------
#region "DrawVLine"

DrawVLine
        ldx PARAM2                              ; fetch the
start address in X (Y coord)
        ldy PARAM1                              ; setup Y
register for column (X coord)
@loop
        lda SCREEN_LINE_OFFSET_TABLE_LO,x       ; Fetch the
address of the start line and
        sta ZEROPAGE_POINTER_1                  ; store it in
ZEROPAGE_POINTER_1
        sta ZEROPAGE_POINTER_2                  ;
ZEROPAGE_POINTER_2 will hold the color address
        lda SCREEN_LINE_OFFSET_TABLE_HI,x
        sta ZEROPAGE_POINTER_1 + 1

        clc
        adc #>COLOR_DIFF                        ; add the
difference to color memory
        sta ZEROPAGE_POINTER_2 + 1              ;
ZEROPAGE_POINTER_2 now has the correct address

        lda PARAM4                              ; load the
character
        sta (ZEROPAGE_POINTER_1),y              ; write the
character to the screen position
        lda PARAM5                              ; load the color
        sta (ZEROPAGE_POINTER_2),Y              ; write it to
color ram

        inx
        clc                                     ; increment X
```

```
        cpx PARAM3                                  ; check against
the end position
        bcc @loop                                   ; if not equal -
loop back

        rts

#endregion
;-------------------------------------------------------------------
---------------
;                                                                   DRAW
HORIZONTAL LINE
;-------------------------------------------------------------------
---------------
; DrawVLine - draws a horizontal line with a specified color and
character.
;             it's not optimized or terribly pretty but it lets
you reuse your
;             PARAM variables to draw another line straight away
;
; PARAM1 = start X
; PARAM2 = start Y
; PARAM3 = end X
; PARAM4 = character
; PARAM5 = color
;-------------------------------------------------------------------
---------------
#region "DrawHLine"

DrawHLine
        ldx PARAM2                          ; load the Y coordinate
for the lookup tables
        ldy PARAM1                          ; start X coord in Y
register

@loop
        lda SCREEN_LINE_OFFSET_TABLE_LO,x       ; load the
screen line address
        sta ZEROPAGE_POINTER_1
        sta ZEROPAGE_POINTER_2                  ; fetch the low
byte for the color address
        lda SCREEN_LINE_OFFSET_TABLE_HI,x
        sta ZEROPAGE_POINTER_1 + 1

        clc
        adc #>COLOR_DIFF                ; add the difference to
color ram
```

```
        sta ZEROPAGE_POINTER_2 + 1

        lda PARAM4
        sta (ZEROPAGE_POINTER_1),y
        lda PARAM5
        sta (ZEROPAGE_POINTER_2),y

        iny                             ; increment y (our X
coordinate
        clc
        cpy PARAM3                      ; compare it to PARAM3 -
end X coordinate
        bcc @loop                       ; if less than, loop
back

        rts
#endregion

; Screen Line Offset Tables
; Query a line with lda (POINTER TO TABLE),x (where x holds the
line number)
; and it will return the screen address for that line

; C64 PRG STUDIO has a lack of expression support that makes
creating some tables very problematic
; Be aware that you can only use ONE expression after a defined
constant, no braces, and be sure to
; account for order of precedence.

; For these tables you MUST have the Operator Calc directive set
at the top of your main file
; or have it checked in options or BAD THINGS WILL HAPPEN!! It
basically means that calculations
; will be performed BEFORE giving back the hi/lo byte with '>'
rather than the default of
; hi/lo byte THEN the calculation
SCREEN_LINE_OFFSET_TABLE_LO
SCREEN1_LINE_OFFSET_TABLE_LO
          byte <SCREEN_MEM
          byte <SCREEN_MEM + 40
          byte <SCREEN_MEM + 80
          byte <SCREEN_MEM + 120
          byte <SCREEN_MEM + 160
          byte <SCREEN_MEM + 200
          byte <SCREEN_MEM + 240
          byte <SCREEN_MEM + 280
          byte <SCREEN_MEM + 320
```

```
            byte <SCREEN_MEM + 360
            byte <SCREEN_MEM + 400
            byte <SCREEN_MEM + 440
            byte <SCREEN_MEM + 480
            byte <SCREEN_MEM + 520
            byte <SCREEN_MEM + 560
            byte <SCREEN_MEM + 600
            byte <SCREEN_MEM + 640
            byte <SCREEN_MEM + 680
            byte <SCREEN_MEM + 720
            byte <SCREEN_MEM + 760
            byte <SCREEN_MEM + 800
            byte <SCREEN_MEM + 840
            byte <SCREEN_MEM + 880
            byte <SCREEN_MEM + 920
            byte <SCREEN_MEM + 960

SCREEN_LINE_OFFSET_TABLE_HI
SCREEN1_LINE_OFFSET_TABLE_HI
            byte >SCREEN_MEM
            byte >SCREEN_MEM + 40
            byte >SCREEN_MEM + 80
            byte >SCREEN_MEM + 120
            byte >SCREEN_MEM + 160
            byte >SCREEN_MEM + 200
            byte >SCREEN_MEM + 240
            byte >SCREEN_MEM + 280
            byte >SCREEN_MEM + 320
            byte >SCREEN_MEM + 360
            byte >SCREEN_MEM + 400
            byte >SCREEN_MEM + 440
            byte >SCREEN_MEM + 480
            byte >SCREEN_MEM + 520
            byte >SCREEN_MEM + 560
            byte >SCREEN_MEM + 600
            byte >SCREEN_MEM + 640
            byte >SCREEN_MEM + 680
            byte >SCREEN_MEM + 720
            byte >SCREEN_MEM + 760
            byte >SCREEN_MEM + 800
            byte >SCREEN_MEM + 840
            byte >SCREEN_MEM + 880
            byte >SCREEN_MEM + 920
            byte >SCREEN_MEM + 960

SCREEN2_LINE_OFFSET_TABLE_LO
            byte <SCREEN2_MEM
```

```
            byte <SCREEN2_MEM + 40
            byte <SCREEN2_MEM + 80
            byte <SCREEN2_MEM + 120
            byte <SCREEN2_MEM + 160
            byte <SCREEN2_MEM + 200
            byte <SCREEN2_MEM + 240
            byte <SCREEN2_MEM + 280
            byte <SCREEN2_MEM + 320
            byte <SCREEN2_MEM + 360
            byte <SCREEN2_MEM + 400
            byte <SCREEN2_MEM + 440
            byte <SCREEN2_MEM + 480
            byte <SCREEN2_MEM + 520
            byte <SCREEN2_MEM + 560
            byte <SCREEN2_MEM + 600
            byte <SCREEN2_MEM + 640
            byte <SCREEN2_MEM + 680
            byte <SCREEN2_MEM + 720
            byte <SCREEN2_MEM + 760
            byte <SCREEN2_MEM + 800
            byte <SCREEN2_MEM + 840
            byte <SCREEN2_MEM + 880
            byte <SCREEN2_MEM + 920
            byte <SCREEN2_MEM + 960

SCREEN2_LINE_OFFSET_TABLE_HI
            byte >SCREEN2_MEM
            byte >SCREEN2_MEM + 40
            byte >SCREEN2_MEM + 80
            byte >SCREEN2_MEM + 120
            byte >SCREEN2_MEM + 160
            byte >SCREEN2_MEM + 200
            byte >SCREEN2_MEM + 240
            byte >SCREEN2_MEM + 280
            byte >SCREEN2_MEM + 320
            byte >SCREEN2_MEM + 360
            byte >SCREEN2_MEM + 400
            byte >SCREEN2_MEM + 440
            byte >SCREEN2_MEM + 480
            byte >SCREEN2_MEM + 520
            byte >SCREEN2_MEM + 560
            byte >SCREEN2_MEM + 600
            byte >SCREEN2_MEM + 640
            byte >SCREEN2_MEM + 680
            byte >SCREEN2_MEM + 720
            byte >SCREEN2_MEM + 760
            byte >SCREEN2_MEM + 800
```

```
            byte >SCREEN2_MEM + 840
            byte >SCREEN2_MEM + 880
            byte >SCREEN2_MEM + 920
            byte >SCREEN2_MEM + 960


SCORE_LINE_OFFSET_TABLE_LO
            byte <SCORE_SCREEN
            byte <SCORE_SCREEN + 40
            byte <SCORE_SCREEN + 80
            byte <SCORE_SCREEN + 120
            byte <SCORE_SCREEN + 160
            byte <SCORE_SCREEN + 200
            byte <SCORE_SCREEN + 240
            byte <SCORE_SCREEN + 280
            byte <SCORE_SCREEN + 320
            byte <SCORE_SCREEN + 360
            byte <SCORE_SCREEN + 400
            byte <SCORE_SCREEN + 440
            byte <SCORE_SCREEN + 480
            byte <SCORE_SCREEN + 520
            byte <SCORE_SCREEN + 560
            byte <SCORE_SCREEN + 600
            byte <SCORE_SCREEN + 640
            byte <SCORE_SCREEN + 680
            byte <SCORE_SCREEN + 720
            byte <SCORE_SCREEN + 760
            byte <SCORE_SCREEN + 800
            byte <SCORE_SCREEN + 840
            byte <SCORE_SCREEN + 880
            byte <SCORE_SCREEN + 920
            byte <SCORE_SCREEN + 960

SCORE_LINE_OFFSET_TABLE_HI
            byte >SCORE_SCREEN
            byte >SCORE_SCREEN + 40
            byte >SCORE_SCREEN + 80
            byte >SCORE_SCREEN + 120
            byte >SCORE_SCREEN + 160
            byte >SCORE_SCREEN + 200
            byte >SCORE_SCREEN + 240
            byte >SCORE_SCREEN + 280
            byte >SCORE_SCREEN + 320
            byte >SCORE_SCREEN + 360
            byte >SCORE_SCREEN + 400
            byte >SCORE_SCREEN + 440
            byte >SCORE_SCREEN + 480
```

```
            byte >SCORE_SCREEN + 520
            byte >SCORE_SCREEN + 560
            byte >SCORE_SCREEN + 600
            byte >SCORE_SCREEN + 640
            byte >SCORE_SCREEN + 680
            byte >SCORE_SCREEN + 720
            byte >SCORE_SCREEN + 760
            byte >SCORE_SCREEN + 800
            byte >SCORE_SCREEN + 840
            byte >SCORE_SCREEN + 880
            byte >SCORE_SCREEN + 920
            byte >SCORE_SCREEN + 960

COLOR_LINE_OFFSET_TABLE_LO
            byte <COLOR_MEM
            byte <COLOR_MEM + 40
            byte <COLOR_MEM + 80
            byte <COLOR_MEM + 120
            byte <COLOR_MEM + 160
            byte <COLOR_MEM + 200
            byte <COLOR_MEM + 240
            byte <COLOR_MEM + 280
            byte <COLOR_MEM + 320
            byte <COLOR_MEM + 360
            byte <COLOR_MEM + 400
            byte <COLOR_MEM + 440
            byte <COLOR_MEM + 480
            byte <COLOR_MEM + 520
            byte <COLOR_MEM + 560
            byte <COLOR_MEM + 600
            byte <COLOR_MEM + 640
            byte <COLOR_MEM + 680
            byte <COLOR_MEM + 720
            byte <COLOR_MEM + 760
            byte <COLOR_MEM + 800
            byte <COLOR_MEM + 840
            byte <COLOR_MEM + 880
            byte <COLOR_MEM + 920
            byte <COLOR_MEM + 960

COLOR_LINE_OFFSET_TABLE_HI
            byte >COLOR_MEM
            byte >COLOR_MEM + 40
            byte >COLOR_MEM + 80
            byte >COLOR_MEM + 120
            byte >COLOR_MEM + 160
            byte >COLOR_MEM + 200
```

```
byte >COLOR_MEM + 240
byte >COLOR_MEM + 280
byte >COLOR_MEM + 320
byte >COLOR_MEM + 360
byte >COLOR_MEM + 400
byte >COLOR_MEM + 440
byte >COLOR_MEM + 480
byte >COLOR_MEM + 520
byte >COLOR_MEM + 560
byte >COLOR_MEM + 600
byte >COLOR_MEM + 640
byte >COLOR_MEM + 680
byte >COLOR_MEM + 720
byte >COLOR_MEM + 760
byte >COLOR_MEM + 800
byte >COLOR_MEM + 840
byte >COLOR_MEM + 880
byte >COLOR_MEM + 920
byte >COLOR_MEM + 960
```