defs.s

```
BANK_0 = $0
BANK_1 = $4000
BANK_2 = $8000
BANK_3 = $c000

BANK = BANK_2

BUFFER = 10
numer = BUFFER
denom = BUFFER+2
multi = BUFFER+2

summy = BUFFER+4
count = BUFFER+4

LINE1 = 1024
LINE2 = 1064
LINE3 = 1104
LINE4 = 1144

x1 = BUFFER+6
y1 = BUFFER+8
x2 = BUFFER+10
y2 = BUFFER+12

flags = BUFFER+14
delta_X = BUFFER+16
delta_Y = BUFFER+18
tmp_X   = BUFFER+20
tmp_Y   = BUFFER+22

SETLFS     = $ffba
SETNAM     = $ffbd
SAVE       = $ffd8
LOAD       = $ffd5
BORDER     = $d020
BACKGROUND = $d021
MESSAGE    = $9d

;SCREEN_MEM = $a000           ; $8000 + $2000
SCREEN_MEM = BANK+$2000
MEM_TWO    = 2
BUFFER16   = 2024
TEXT_MEM   = 1024
TMP        = 4

VIC_BANK   = $dd00
MEM_SETUP  = $d018
COLOR_RAM  = $d800           ; 0000.0011

SCREEN_CONTROL_1 = $d011
SCREEN_CONTROL_2 = $d016

BLACK       = 0
WHITE       = 1
RED         = 2
CYAN        = 3
PURPLE      = 4
GREEN       = 5
BLUE        = 6
YELLOW      = 7
ORANGE      = 8
BROWN       = 9
PINK        = 10
DARK_GREY   = 11
GREY        = 12
LIGHT_GREEN = 13
LIGHT_BLUE  = 14
LIGHT_GREY  = 15
```

foo.s

```
; cl65 -o test -u __EXEHDR__ -t c64 -C /usr/local/share/cc65/cfg/c64-asm.cfg main.s && mv test ~/Vice/vicefs/
; ~/github/cc65/bin/cl65 -o test -u __EXEHDR__ -t c64 -C ~/github/cc65/cfg/c64-asm.cfg foo.s && mv test ~/Vice/vicefs/

   jmp main

.include "defs.s"
.include "vol_1.s"
.include "my_math.s"
.include "quadrants.s"
.include "draw_ticks.s"
.include "draw_line.s"

main:
   jsr set_multi_color_mode

   lda #WHITE
   jsr set_color_ram

   lda #$26                  ; red / blue
   jsr set_color_cells

   ldx #BLACK
   ldy #DARK_GREY
   jsr fill_background

   jsr draw_ticks

   ; ***** draw line
   lda #10
   sta x1
   lda #10
   sta y1

   lda #90
   sta x2
   lda #45
   sta y2
   jsr draw_line

loop: jmp loop

; ********************************************************
; SUMMARY
;
; 00: from background color
; 01: (BANK + MEM_SETUP high nibble)
; 10: (BANK + MEM_SETUP high nibble)
; 11: COLOR_RAM
```

draw_line.s

```
; ************************************************************************************
draw_line:
   ; ***** clear deltas and flags
   lda #0
   sta delta_X
   sta delta_X+1
   sta delta_Y
   sta delta_Y+1
   sta flags

   ; ***** define deltaX
   sec
   lda x2
   sbc x1
   sta delta_X
   bpl :+
   inc flags
:
   ; ***** define deltaY
   sec
   lda y2
   sbc y1
   sta delta_Y
   bpl :+
   lda flags
   ora #2
   sta flags
:
   lda flags
   cmp #1
   beq :+
   cmp #2
   beq :++
   cmp #3
   beq :+++
   jmp default
:                              ;  y,-x
   ldx #5
   ldy #5
   lda #2
   jsr put_dot
   jmp continue0
:                              ; -y, x
   ldx #10
   ldy #10
   lda #2
   jsr put_dot
   jmp continue0
:                              ; -y,-x
   ldx #15
   ldy #15
   lda #2
   jsr put_dot
   jmp continue0
default:                       ;  y, x
   jsr do_quad_I
continue0:
   nop

   rts

; ****************************************************
;  A       B
; (10,10) (90,90) = 20
; (90,90) (10,10) = 15
; (10,90) (10,90) = 20
; (10,90) (90,10) = 10
; (90,10) (10,90) =  5
```

quadrants.s

```
; ************************************************************************************************
do_quad_I:
   lda delta_X
   cmp delta_Y
   bpl :+
   jsr do_q1_b               ; X < Y
   rts
:
   jsr do_q1_a               ; X >= Y
   rts

; ***** Quad_I: X >= Y
do_q1_a:
   ldx x1
   stx tmp_X
   ldy y1
   sty tmp_Y
   lda #2
   jsr put_dot

   ldx x2
   ldy y2
   lda #2
   jsr put_dot

@loop:
   ldx x1
   ldy y1
   lda #3
   jsr put_dot
   jsr redefine_Y1
   inc x1
   lda x1
   cmp x2
   bne @loop

   rts

; ***** Quad_I: X < Y
do_q1_b:
   nop

   rts

; ============================================================================================
redefine_Y1:
   ; A(10,10)
   ; B(90,80)

   ; ***** x1 - Ax
   sec
   lda x1
   sbc tmp_X

   ; ***** multiply by delta_Y
   sta numer
   lda #0
   sta numer+1
   sta multi+1
   lda delta_Y
   sta multi
   jsr do_multiply

   ; ***** divide by delta_X
   lda delta_X
   sta denom
   lda #0
   sta denom+1
   jsr do_divide

   lda numer
   sta y1

   ; ***** add Ay
   clc
   lda numer
   adc tmp_Y
   sta y1

   ;lda #15
   ;sta y1

   rts
```

vol_1.s

```
; ********************************************************************************************
set_multi_color_mode:
   ; ***** disable I/O & error messages
   lda MESSAGE
   and #$3f
   sta MESSAGE

   ; ***** turn off BASIC
   lda $1
   and #$fc
   ora #2
   sta $1

   ; ***** turn on bitmap
   lda SCREEN_CONTROL_1
   ora #32
   sta SCREEN_CONTROL_1

   ; ***** turn on multi-color
   lda SCREEN_CONTROL_2
   ora #16
   sta SCREEN_CONTROL_2

   ; ***** Bank
   lda VIC_BANK
   and #$fc
   ora #1            ; Bank #2, $8000-$BFFF, 32768-49151.
   sta VIC_BANK

   ; ***** (high nibble; 0) $0
   ; ***** (low nibble; 8)  $8
   lda #$8
   sta MEM_SETUP

   rts

; ********************************************************************************************
put_dot:
   ; ***** BUFFER16 *****
   ; +0: X
   ; +1: remainder X
   ; +2: Y
   ; +3: remainder Y
   ; +4: palette
   stx BUFFER16
   sty BUFFER16+2
   sta BUFFER16+4

   ; ************************ dealing with X
   lda BUFFER16
   sta BUFFER16+1
   and #3                    ; 0000.0011
   sta BUFFER16+1            ; store remainder
   lsr BUFFER16              ;; X divided
   lsr BUFFER16              ;; by 4

   ; ************************ dealing with Y
   lda BUFFER16+2
   sta BUFFER16+3
   and #7                    ; 0000.0111
   sta BUFFER16+3            ; store remainder
   lsr BUFFER16+2            ;; Y divided
   lsr BUFFER16+2            ;;
   lsr BUFFER16+2            ;; by 8

   ; ************************************************ (Y * 40) + X = CELL
   ; ************************ Y * 40
   ; ************************ 40 = 0010.1000
   ; ******************** Y << 5; Y * 32
   lda BUFFER16+2
   sta BUFFER16+6            ; using BUFFER16+6
   lda #0                    ; for
   sta BUFFER16+7            ; 16-bit integer

   clc
   ldx #5
:
   asl BUFFER16+7
   asl BUFFER16+6
   bcc :+
   inc BUFFER16+7
:
   dex
   bne :--

   ; ******************** Y << 3; Y * 8
```

```
   lda BUFFER16+2
   sta BUFFER16+8            ; using BUFFER16+8
   lda #0                    ; for
   sta BUFFER16+9            ; 16-bit integer

   clc
   ldx #3
:
   asl BUFFER16+8
   bcc :+
   inc BUFFER16+9
:
   dex
   bne :--

   ; ******************** BUFFER16+6 plus BUFFER16+8 = BUFFER16+10
   ; ******************** Note: Y * 40
   clc
   lda BUFFER16+6
   adc BUFFER16+8
   sta BUFFER16+10
   lda BUFFER16+7
   adc BUFFER16+9
   sta BUFFER16+11

   ; ******************** Add X to BUFFER16+10 to get CELL
   ; 6,7
   lda BUFFER16
   sta BUFFER16+6
   lda #0
   sta BUFFER16+7
   ; 8,9
   lda BUFFER16+10
   sta BUFFER16+8
   lda BUFFER16+11
   sta BUFFER16+9
   ; ***** add the X
   clc
   lda BUFFER16+6
   adc BUFFER16+8
   sta BUFFER16+10
   lda BUFFER16+7
   adc BUFFER16+9
   sta BUFFER16+11

   ; ******************** CELL * 8
   clc
   ldx #3
:
   asl BUFFER16+11
   asl BUFFER16+10
   bcc :+
   inc BUFFER16+11
:
   dex
   bne :--

   ; ********** add remainder Y
   ; 6,7
   lda BUFFER16+3
   sta BUFFER16+6
   lda #0
   sta BUFFER16+7
   ; 8,9
   lda BUFFER16+10
   sta BUFFER16+8
   lda BUFFER16+11
   sta BUFFER16+9
   clc
   lda BUFFER16+6
   adc BUFFER16+8
   sta BUFFER16+10
   lda BUFFER16+7
   adc BUFFER16+9
   sta BUFFER16+11

   ; ********** add SCREEN_MEM
   ; 6,7
   lda #<SCREEN_MEM
   sta BUFFER16+6
   lda #>SCREEN_MEM
   sta BUFFER16+7
   ; 8,9
   lda BUFFER16+10
   sta BUFFER16+8
   lda BUFFER16+11
   sta BUFFER16+9
   clc
```

```
   lda BUFFER16+6
   adc BUFFER16+8
   sta BUFFER16+10
   lda BUFFER16+7
   adc BUFFER16+9
   sta BUFFER16+11

   ; ************************************************ define palette-pixel
   lda BUFFER16+10
   sta 2
   lda BUFFER16+11
   sta 3

   ; ***** store original value of screen-byte in memory 4
   ldy #0
   lda (2),y
   sta 4

   ; ***** store palette in memory 5
   lda BUFFER16+4
   sta 5
   ; *** duplicate palette; EG, from 0000.0010 to 1010.1010
   asl
   asl
   ora 5
   sta 5
   asl
   asl
   ora 5
   sta 5
   asl
   asl
   ora 5
   sta 5

   ; ***** convert remainder X to pixel pattern and store value in memory 6; mask
   lda BUFFER16+1
   cmp #3
   beq three
   jmp :+
three:
   lda #3                   ; 0000.0011
   jmp continue
:
   cmp #2
   beq two
   jmp :+
two:
   lda #12                  ; 0000.1100
   jmp continue
:
   cmp #1
   beq one
   jmp :+
one:
   lda #48                  ; 0011.0000
   jmp continue
:
   lda #192                 ; 1100.0000
continue:
   sta 6                    ; mask
   ; ***** create ~mask and store it in memory 7
   eor #$ff
   sta 7

   ; ***** palette AND mask and store it in memory 8; aaa
   lda 5
   and 6
   sta 8

   ; ***** screen-byte AND ~mask and store it in memory 9; bbb
   lda 4
   and 7
   sta 9

   ; ***** aaa OR bbb and store it in screen
   lda 8
   ora 9
   ldy #0
   sta (2),y

   rts

; ****************************************************** clear background with a color
fill_background:
   stx BACKGROUND
   sty BORDER
```

```
   lda #<SCREEN_MEM
   sta MEM_TWO
   lda #>SCREEN_MEM
   sta MEM_TWO+1

   ldy #0
   lda #0
   ldx #32
:
   sta (MEM_TWO),y
   iny
   bne :-
   inc MEM_TWO+1
   dex
   bne :-

   rts

; ********************************************************************************************
set_color_ram:
   sta TMP

   lda #<COLOR_RAM
   sta MEM_TWO
   lda #>COLOR_RAM
   sta MEM_TWO+1

   lda TMP
   ldy #0
   ldx #4
:
   sta (MEM_TWO),y
   iny
   bne :-
   inc MEM_TWO+1
   dex
   bne :-

   rts

; ********************************************************************************************
set_color_cells:
   sta TMP

   lda #<BANK
   sta MEM_TWO
   lda #>BANK
   sta MEM_TWO+1

   lda TMP
   ldy #0
   ldx #4
:
   sta (MEM_TWO),y
   iny
   bne :-
   inc MEM_TWO+1
   dex
   bne :-
   rts
```

my_math.s

```
; ********************************************************************************************
do_multiply:
   lda multi
   adc multi+1
   bne :+
   lda #0                    ; if multiply by 0, set clear numer
   sta numer
   sta numer+1
   rts
:
   clc
   lda multi
   adc multi+1
   cmp #1                    ; if multiply by 1, return
   bne :+
   rts
:
   lda #0
   sta summy
   sta summy+1

multiply_loop:
   clc
   lda summy
   adc numer
   sta summy
   lda summy+1
   adc numer+1
   sta summy+1

   sec
   lda multi
   sbc #1
   sta multi
   lda multi+1
   sbc #0
   sta multi+1
   beq multiply_loop

   ; ***** correction
   sec
   lda summy
   sbc numer
   sta numer
   lda summy+1
   sbc numer+1
   sta numer+1

   rts

; ********************************************************************************************
do_divide:
   lda denom
   adc denom+1
zero: beq zero               ; if divided by 0, infinate loop. LOL

   lda denom
   and #$fe
   beq return

   lda #$ff
   sta count
   sta count+1
:
   clc
   lda count
   adc #1
   sta count
   lda count+1
   adc #0
   sta count+1

   sec
   lda numer
   sbc denom
   sta numer
   lda numer+1
   sbc denom+1
   sta numer+1
   bpl :-                    ; if positive, loop

   lda count
   sta numer
   lda count+1
   sta numer+1
```

```
return:
   rts
```