```
word;C64 screen and sprite memory addresses
enable_sprite       = $D015
sprite0_mem_pointer = $07F8
sprite1_mem_pointer = $07F9
sprite2_mem_pointer = $07FA
sprite_enable_mc    = $D01C
sprite_expand_x     = $D01D
sprite_collision    = $D01E
;sprite_multicolor0  = $D025
;sprite_multicolor1  = $D026
sprite0_color       = $D027
sprite1_color       = $D028
sprite0_x           = $D000
sprite0_y           = $D001
sprite1_x           = $D002
sprite1_y           = $D003
;sprite2_x           = $D004
;sprite2_y           = $D005
msb_x               = $D010
backgrnd_color_addr = $D021
boarder_color_addr  = $D020
raster_line         = $D012
clear_screen        = $E544
;joystick1           = $DC01 ;56321
joystick2           = $DC00 ;56320

;SID sound chip memory addresses
frelo1   = $d400      ;voice 1 frequency register lo byte
frehi1   = $d401      ;voice 1 frequency register hi byte
vcreg1   = $d404      ;voice 1 control register
atdcy1   = $d405      ;voice 1 attack / decay register
surel1   = $d406      ;voice 1 sustain / release register
frelo2   = $d407      ;so+7
sigvol   = $d418      ;so+24

;status_register     = $030F

SCR_ADDR_LO    = 251
SCR_ADDR_HI    = 252
;these are memory addresses for the variables starting at 679
brick_index    = 679
brick_char     = 681
brick_char_y   = 682
dir_x          = 683
dir_y          = 684
ball_counter   = 685
score_digit1   = 686
```

```
score_digit2   = 687
score_digit3   = 688
score_digit4   = 689

y40_LO    = 690
y40_HI    = 691
y8_LO  = 692
y8_HI     = 693
y32_LO    = 694
y32_HI    = 695
ychar = 696
xchar = 697

screen_collis = 253

;ZPTempLO = 253
;ZPTempHi  = 254

ball_cnt_label_scr_addr  =  1985 ; 1024+X+(40*Y)  ; 1026
ball_counter_scr_addr    =  1991;label start addr + label length
+ 1 ; 1032
score_label_scr_addr  = 2011
score_digit1_scr_addr = 2019
score_digit2_scr_addr = 2020
score_digit3_scr_addr = 2021
score_digit4_scr_addr = 2022

;constants
left                = #24  ;left border
top                 = #50  ;top
bottom              = #243 ;bottom
right               = #81  ;right NOTE: 81 with bit set true is
256+81= 337
ball_start_x        = #180
ball_start_y        = #144
ball_color          = #1 ;white ball
paddle_start_x      = #168
paddle_start_y      = #224
paddle_color        = #6 ;blue paddle
backgrnd_color      = #0 ;default darkblue is 6 , black=0
boarder_color       = #15 ;default light blue is 14, light
gray=15

; 10 SYS (49152)
*=$801
        byte
$0E,$08,$0A,$00,$9E,$20,$28,$34,$39,$31,$35,$32,$29,$00,$00,$00
```

```
;sprite data
* = 16256;16192
        ;paddle 12x4 top left corner
        byte
$ff,$f8,$00,$ff,$f8,$00,$ff,$f8,$00,$ff,$f8,$00,$00,$00,$00,$00
        byte
$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
        byte
$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
        byte
$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$05
        ;small 8x8 ball top left corner
        byte
$38,$00,$00,$7c,$00,$00,$fe,$00,$00,$fe,$00,$00,$fe,$00,$00,$7c
        byte
$00,$00,$38,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
        byte
$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
        byte
$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$01

* = 49152 ;$C000-$CFFF, 49152-53247 Upper RAM area (4096 bytes).
        ;set sprite memory pointers
        lda  #255
        sta  sprite0_mem_pointer ;sprite0 is the ball
        lda  #254
        sta  sprite1_mem_pointer ;sprite1 is the paddle
        ;enable sprites
        lda #3
        sta enable_sprite       ;enable sprites 0 and 1
        ;enable sprite multi-color
        lda #0 ;zero to disable all multi-color
        sta sprite_enable_mc
        ;set sprite colors
        ;lda #0
        ;sta sprite_multicolor0 ;black / shadow
        ;lda #1
        ;sta sprite_multicolor1 ;white / light
        lda ball_color
        sta sprite0_color
        lda paddle_color
        sta sprite1_color

        ;sprite to foreground display priority register
        ;lda #3
        ;sta 53275
```

```
;sprite expand
lda #2
sta sprite_expand_x ;x-expanded #$d01d

;initialize ball direction vars
lda #0
sta dir_x ;set x direction
lda #1
sta dir_y ;set y direction

;set initial ball position
jsr reset_ball

;set initial paddle position
lda paddle_start_x
sta sprite1_x
lda paddle_start_y
sta sprite1_y

;init_screen
lda backgrnd_color
sta backgrnd_color_addr    ;set background color
lda boarder_color
sta boarder_color_addr       ;set border color
;clear screen
jsr clear_screen
jsr clear_sound
jsr draw_bricks
jsr color_bricks
;setup ballcounter
lda #49 ;init ballcount to one in ascii
sta ball_counter
jsr display_ball_counter_label ;"BALL:"
jsr display_ball_counter
lda #48
sta score_digit1
sta score_digit2
sta score_digit3
sta score_digit4
jsr display_score_label
jsr display_score

;test erase brick
lda #25
sta brick_index
jsr erase_brick
```

```
        lda #30
        sta brick_index
        jsr erase_brick

        lda #34
        sta brick_index
        jsr erase_brick
        ;rts


;wait for raster scan line to be off screen (>250)
raster
        lda raster_line ;$D012
        cmp #250
        bne raster
main
        jsr spritechar_colis
        jsr move_ball_horizontally
        jsr move_ball_vertically
        jsr move_paddle
        jsr move_paddle
        jsr check_ball_paddle_collision
        jsr check_ball_wall_collisions
        jmp raster
;END of MAIN

check_ball_wall_collisions
        ;check right wall collisions
        lda msb_x
        and #1
        cmp #1
        bne check_left_ball_wall_collision
        ;ball msb is set
        lda sprite0_x
        cmp right
        beq set_ball_direction_left
check_left_ball_wall_collision
        lda msb_x
        and #1
        cmp #0
        bne raster ;only check left wall when msb is zero, else
exit
        lda sprite0_x
        cmp left
        beq set_ball_direction_right
        rts
```

```
set_ball_direction_right
        lda #1
        sta dir_x
        jsr sound
        rts

set_ball_direction_left
        lda #0
        sta dir_x
        jsr sound
        rts

move_ball_horizontally
        lda dir_x
        cmp #0
        beq move_ball_left
        cmp #1
        beq move_ball_right
        rts

move_ball_right
        inc sprite0_x
        bne dont_toggle_ball_msb_right ;checks zero flag
        lda msb_x
        eor #$01 ;sprite0 x-axis msb
        sta msb_x
dont_toggle_ball_msb_right
        rts

move_ball_left
        lda sprite0_x
        bne dont_toggle_ball_msb_left ;checks zero flag
        lda msb_x
        eor #$01 ;sprite0 x-axis msb
        sta msb_x
dont_toggle_ball_msb_left
        dec sprite0_x
        rts

move_ball_vertically
        lda dir_y
        cmp #0
        beq moveball_up
        cmp #1
        beq moveball_down
        rts
```

```
moveball_down
        inc sprite0_y
        jsr check_ball_floor_collision
        rts

moveball_up
        dec sprite0_y
        jsr check__ball_ceiling_collision
        rts

check__ball_ceiling_collision
        lda sprite0_y
        cmp top
        beq set_ball_direction_down
        rts

check_ball_floor_collision
        lda sprite0_y
        cmp bottom
        beq reset_ball
        rts

set_ball_direction_down
        lda #1
        sta dir_y
        jsr sound
        rts

set_ball_direction_up
        lda #0
        sta dir_y
        jsr sound
        rts

reset_ball
        ;reset ball x,y
        lda ball_start_x
        sta sprite0_x
        lda ball_start_y
        sta sprite0_y
        ;clear ball msb if needed
        lda msb_x
        and #1
        cmp #1
        bne skip_reset_msb
        lda msb_x
```

```
        eor #$01
        sta msb_x
skip_reset_msb
        ;increment ball count
        inc ball_counter
        ;roll over digit
        lda ball_counter
        cmp #58 ;58 in ascii is 9 +1
        beq reset_ball_counter
        ;update ball count display
        jsr display_ball_counter
        rts

reset_ball_counter
        lda #48 ;zero in ascii
        sta ball_counter
        ;update ball count display
        jsr display_ball_counter
        rts

move_paddle_right
        lda msb_x
        and #2
        cmp #2
        bne mv_pad_r
        ;ball msb is set
        lda sprite1_x
        cmp #62 ;RIGHT offset with the paddle width minus ball
width
                ;NOTE: currently RIGHT is already offset with
ball width
                ;assumes paddle is x-axis expanded
        beq dont_toggle_paddle_msb_right
mv_pad_r
        inc sprite1_x
        bne dont_toggle_paddle_msb_right
        lda msb_x
        eor #$02 ;sprite1 x-axis msb
        sta msb_x
dont_toggle_paddle_msb_right
        rts

move_paddle_left
        lda sprite1_x
        bne dont_toggle_paddle_msb_left
        lda msb_x
        eor #$02 ;sprite1 x-axis msb
```

```
        sta msb_x
dont_toggle_paddle_msb_left
        lda msb_x
        and #2
        cmp #2
        beq mv_pad_l
        lda sprite1_x
        cmp left
        bne mv_pad_l
        rts
mv_pad_l
        dec sprite1_x
        rts

move_paddle
        ;read input from joystick
        lda  joystick2    ;joy port 2 $DC00
        and  #8 ;right
        beq move_paddle_right
        lda  joystick2
        and  #4 ;#left
        beq move_paddle_left
        rts

check_ball_paddle_collision
        lda sprite_collision ;53278
        and #2
        cmp #2
        beq ball_paddle_collision
        rts

ball_paddle_collision
        lda sprite_collision
        and #2
        sta sprite_collision ;clear sprite collision bit
        jsr set_ball_direction_up
        rts

display_ball_counter_label ;"BALL:"
        ldx #0
read_ball_cnt_label
        lda ball_counter_label,x
        sta ball_cnt_label_scr_addr,x
        inx
        cpx #5 ;label text length
        bne read_ball_cnt_label
        rts
```

```
display_ball_counter
        lda ball_counter
        sta ball_counter_scr_addr
        rts

display_score_label ;"SCORE: "
        ldx #0
read_score_label
        lda score_label,x
        sta score_label_scr_addr,x
        inx
        cpx #7
        bne read_score_label
        rts

display_score
        lda score_digit1
        sta score_digit1_scr_addr
        lda score_digit2
        sta score_digit2_scr_addr
        lda score_digit3
        sta score_digit3_scr_addr
        lda score_digit4
        sta score_digit4_scr_addr
        lda #0
        rts

draw_bricks
        lda #0
        sta brick_index
draw_bricks_loop
        jsr draw_brick
        inc brick_index
        lda brick_index
        cmp #40
        bne draw_bricks_loop
        rts

erase_brick
        ldx brick_index ;
        lda brick_screen_address_lo,x
        sta SCR_ADDR_LO
        lda brick_screen_address_hi,x
        sta SCR_ADDR_HI
        ldy #0
erase_brick_loop
```

```
        lda #32 ;ascii space
        sta (SCR_ADDR_LO),y
        sty brick_char_y
        tya
        clc
        adc #40
        tay
        lda #32 ;ascii space
        sta (SCR_ADDR_LO),y
        ldy brick_char_y
        iny
        cpy #4
        bne erase_brick_loop
        rts

draw_brick
        ldx brick_index ;
        lda brick_screen_address_lo,x
        sta SCR_ADDR_LO
        lda brick_screen_address_hi,x
        sta SCR_ADDR_HI
        ldy #0
read_brick_char_data_loop
        lda brick_char_data_top,y
        sta (SCR_ADDR_LO),y
        lda brick_char_data_bottom,y
        sta brick_char
        sty brick_char_y
        tya
        clc
        adc #40
        tay
        lda brick_char
        sta (SCR_ADDR_LO),y
        ldy brick_char_y
        iny
        cpy #4
        bne read_brick_char_data_loop
        rts

color_bricks
        ldx #0
read_color_addr_loop
        lda row_color_adrress,x
        sta SCR_ADDR_LO
        inx
        lda row_color_adrress,x
```

```
        sta SCR_ADDR_HI
        inx
        lda brick_color_values,x-2
        jsr color_row
        cpx #10 ; 4 rows - incr by 2 each interation for hi/lo
        bne read_color_addr_loop
        rts

color_row
        ;lda #5 ;red = 2, orange = 8, green = 5, yellow = 7
        ldy #0 ;
color_row_loop
        sta (SCR_ADDR_LO),y
        iny
        cpy #80
        bne color_row_loop
        rts

score_label
        byte 32,19,3,15,18,5,58
ball_counter_label
        byte 2,1,12,12,58

row_color_adrress
        ;55296+X+(40*Y)
        ;55376=$D850
        ;55456=$D8A0 / $A0=#160,$D8=#216
        ;55536=$D8F0 / $F0=#240,$D8=#216
        ;55616=$D940 / $40=#64,$D9=#217
        ;55696=$D990 / $90=#144,$D9=#217
        ;byte 80,216
        ;byte 160,216
        ;byte 240,216
        ;byte 64,217
        ;byte 144,217 ;note reverse order of 2 byte mem addr
        byte 120,216
        byte 200,216
        byte 24,217
        byte 104,217
        byte 193,219

brick_color_values
        ;red = 2, orange = 8, green = 5, yellow = 7, cyan = 3,
white =1, lt. gray = 15
        byte 2,0,8,0,5,0,7,0,15,0
brick_char_data_top
        byte 108,98,98,123 ;brick top
```

```
brick_char_data_bottom
        byte 124,226,226,126 ;brick bottom
        ;1024+X+(40*Y)
        ; starting 4 rows from the top 1184
        ;1104
        ; row 1 $04A0 = 1184 / $A0 = #160 , $04 = #4
        ; row 2 $04F0 = 1264 / $F0 = #240 , $04 = #4
        ; row 3 $0540 = 1344 / $40 = #64  , $05 = #5
        ; row 4 $0590 = 1424 / $90 = #144 , $05 = #5
brick_screen_address_hi
        byte 4,4,4,4,4,4,4,4,4,4
        byte 4,4,4,4,4,4,4,4,4,4
        ;byte 4,4,4,4,5,5,5,5,5,5
        byte 5,5,5,5,5,5,5,5,5,5
        byte 5,5,5,5,5,5,5,5,5,5
brick_screen_address_lo
        ;byte 80,84,88,92,96,100,104,108,112,116
        ;byte 160,164,168,172,176,180,184,188,192,196
        ;byte 240,244,248,252,0,4,8,12,16,20
        ;byte 64,68,72,76,80,84,88,92,96,100
        ;byte 144,148,152,156,160,164,168,172,176,180
        byte 120,124,128,132,136,140,144,148,152,156
        byte 200,204,208,212,216,220,224,228,232,236
        byte 24,28,32,36,40,44,48,52,56,60
        byte 104,108,112,116,120,124,128,132,136,140

sound
        jsr clear_sound
        lda #5
        sta atdcy1      ;s0+5
        lda #5
        sta surel1      ;so+6
        lda #15
        sta sigvol      ;so+24
        lda #7
        sta frelo1      ;so
        lda #27
        sta frehi1      ;so+1
        lda #33
        sta vcreg1      ;so+4
        rts

clear_sound
        ldy #23
clr
        lda #0
        sta frelo1,y
```

```
        dey
        bne clr
        rts

spritechar_colis
;Detect ASCII screen memory - this may be off by + or - 32
          lda             brick_screen_address_lo
          sta             SCR_ADDR_LO
          lda             brick_screen_address_hi
          sta             SCR_ADDR_HI

          lda             #0
          sta             253
          lda             #4
          sta             254


;                   x1: screen_tempx
;************************************************
          ;sed

;Code will calculate where the screen memory for the bricks
;will position the x,y alogorithm

;1284 + X1 + (y1 * 40)

;It looks in screen memory (brick = 48). Uses the alogorithm
;to see what value the sprite found there at x,y (53248,53249)

;xc = (sprite0_x - 40) / 8
;yc = (sprite0_y - 24) / 8

;xc = (Sprite x - Screen Width)  / Character Size
;yc = (Sprite y - Screen Height) / Character Size

          lda             sprite0_x       ;180

          sec
          sbc             #24             ;y position

          lsr             ;/ 2
          lsr             ;/ 4
          lsr             ;/ 8

;xchar = 19
          sta             xchar     ;XC=(x1-24)/8 = 19
```

```
;************************************************
;                    y1
;************************************************
;          lda           #0
;          sta           251
;          sta           252

          lda           sprite0_y      ;get ball's y position

          sec
          sbc           #25            ;calc y's (top/bottom)

;123
          lsr                          ;calc sprite width
          lsr                                ;area
          lsr                          ;divide by 8
;ychar = 15

          sta           ychar          ;YC=(y1-21)/8 = 15

          lda           #0
          sta           y32_HI

;================================================================
===

; We need to calculate ychar x 32 separate from ychar x 8.
; This is because ychar x 32 results in a number larger than
255.
; So we cannot essentially calculate ychar * 40 without setting
the
; carry flag. So the solution is to set up separate calculations
for
; each and then add them together for the final answer.
; It looks something like this according to the Commodore 64
Basic
; program "Sprite to Background Collision".

; xc = (Sprite x - ((32 + 8) * ychar)) / 8

;                     ychar*32
;================================================================
===
;This code is calculating the ychar * 32

          lda           ychar                ;  15
          asl                                ; 15 x 2 = 30
```

```
          rol             y32_HI
          asl                                   ; 15 x 4 = 60

          rol             y32_HI
          asl                                    ;15 x 8 = 120

          rol             y32_HI
          asl                                    ;15 x 16 = 240

          rol             y32_HI                 ;save the
overflow bits

          asl                                    ;15 x 32 = 480
          rol             y32_HI
                                                 ;(253)+(254)=464
;peek(253)=240
;peek(251)=224

          sta             y32_LO                  ;peek(251)
;224

;================================================================
===
;                         ychar * 8
;================================================================
===

          lda             #0
          sta             y8_HI
          lda             ychar                  ;=15

          asl
          rol             y8_HI                  ;15 x 2 = 30

          asl
          rol             y8_HI                  ;15 x 4 = 60

          asl
          rol             y8_HI                  ;15 x 8 = 120

          sta             y8_LO                  ;YC*8

;================================================================
===
;                         ychar * 40
```

```
;=================================================================
===

          clc

;32+8 = 40

; y32_Lo (32) + (y8_lo) (8) = y40_lo
          lda             y32_LO                  ;224 (ychar x
32)
          adc             y8_LO                   ;88  (ychar x 32
+ 8)

          sta             y40_LO                  ;88

          lda             y32_HI                  ;88 (ychar)
          adc             y8_HI                   ;2
          sta             y40_HI

;=================================================================
===
;                       ychar * 40 + Screen
;=================================================================
===

; vl = 1024 + xchar + (ychar * 40)

;don't forget xchar

          clc
          lda             y40_LO                  ;88
          adc             253
          sta             253
;          adc             SCR_ADDR_LO
;brick_screen_address_lo
;208

;peek(253)=88, peek(254)=6 - 1624
          lda             y40_HI                  ;2
          adc             254
          sta             254
          ;adc             xchar
          sta             254

          ldy             xchar
          ldx             #3
chk_bkchar
```

```
          lda             (253),y
          cmp             brick_char_data_top,x
          beq             found_brick
          cmp             brick_char_data_bottom,x
          beq             found_brick
          dex
          bne             chk_bkchar
          jmp             exit_colischk
found_brick
;Found a brick0
          ldx             #0
          lda             143
          sta             53280
erase_bricks
          lda             #32
          sta             (253),y
          inx
          cpx             #2
          bne             erase_bricks
          ;brk


;          adc             SCR_ADDR_HI
;brick_screen_address_hi
                                                  ;146
;xchar = 19

;          adc             xchar
;          sta             screen_collis

;          ldy             xchar                   ;ychar = 15
;          sty             251
;          tya                                     ;save ychar in
accum

;rd_screenhi
;          ldx             xchar                   ;xchar = 19
;          txa                                     ;save xchar in
accum

;rdulow    lda             #1
;          sta             (SCR_ADDR_LO),y         ;1024,x
;          adc             xchar
;          sta             (SCR_ADDR_LO),y         ;1024,x
;          tay
;          iny
;          cpy             #20
```

```
;           bne             rdulow
;           ;iny
;           ;cpy             #2
;           ;bne             rdulow
;           tax
;           inc             SCR_ADDR_HI
;           inx
;           cpx             #30
;           bne             rd_screenhi
exit_colischk
           rts
```