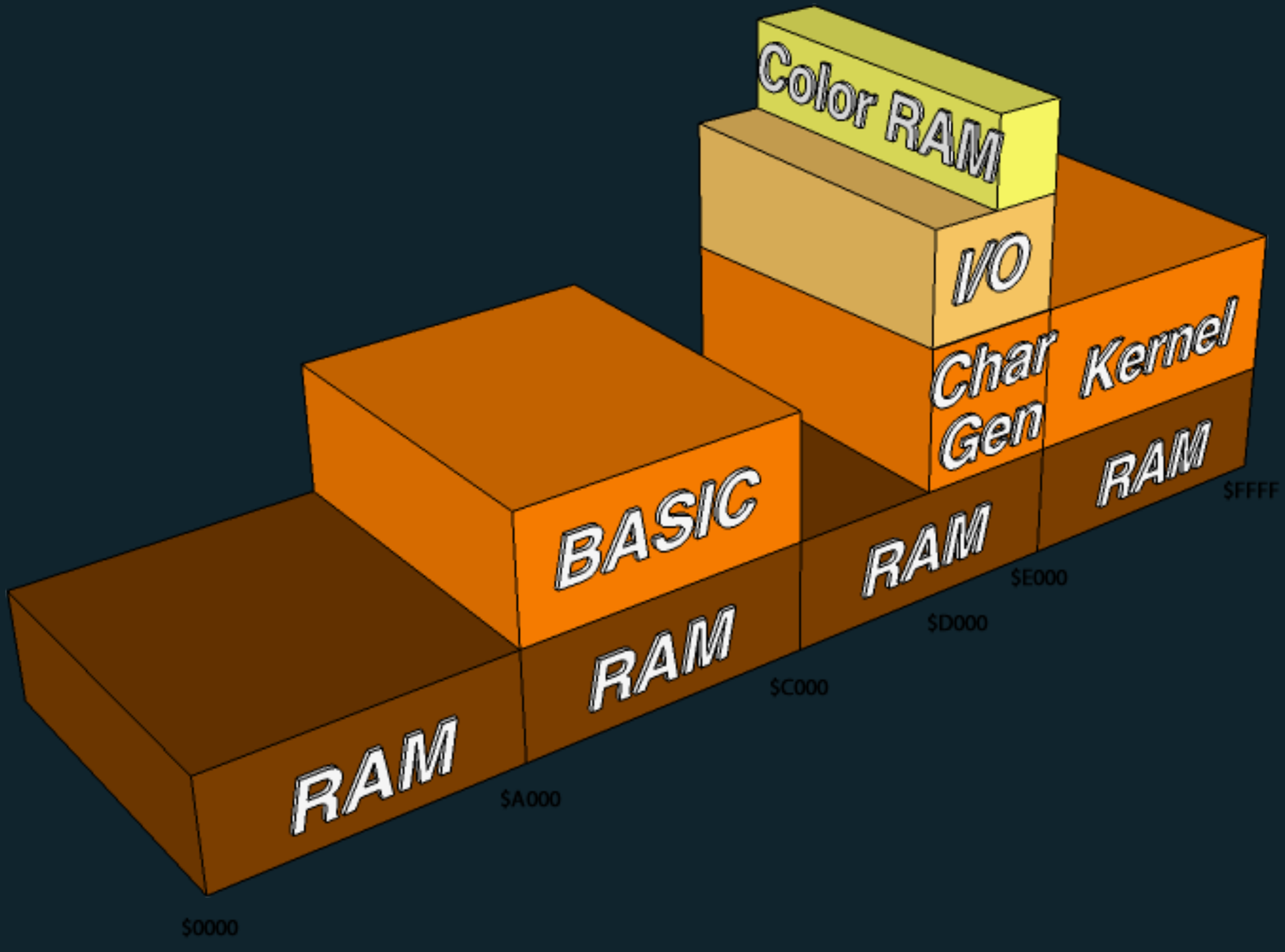
Animated sprites on the Commodore 64

Memory on the Commodore 64


Color RAM
I/O
Char Gen
Kernel
RAM
BASIC
RAM
RAM
RAM
$FFFF
$E000
$D000
$C000
$A000
$0000

— The I/O chips are memory-mapped.
— You access features of SID, VIC-II etc by reading and writing to memory addresses.

Graphics on the Commodore 64

VIC-2

— Three character modes and two bitmap modes
— 16 possible colors
— Concurrent handling of 8 sprites per scanline

What is a sprite?

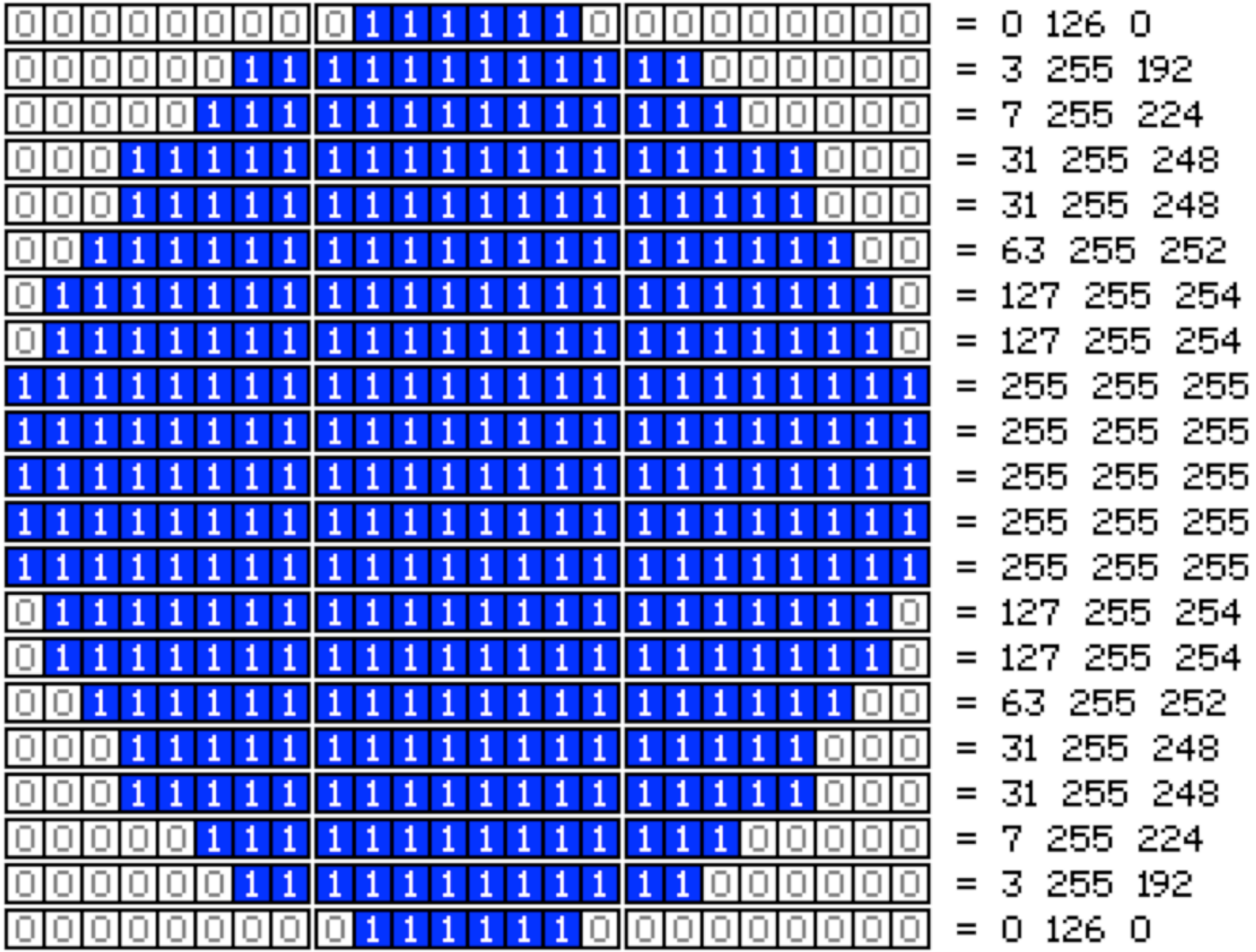
— A graphics object that has hardware support for display, positioning, and color independent of the graphics mode.

— Manipulated by writing directly to memory locations

— 2 types: High Resolution and Multicolor

High Resolution Sprite

— 24x21 bits, broken up into 63 8-bit bytes
— A "1" is on, a "0" is transparent.
— One color


0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 = 0 126 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 = 3 255 192
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 = 7 255 224
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 = 31 255 248
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 = 31 255 248
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 = 63 255 252
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 = 127 255 254
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 = 127 255 254
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = 255 255 255
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = 255 255 255
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = 255 255 255
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = 255 255 255
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = 255 255 255
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 = 127 255 254
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 = 127 255 254
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 = 63 255 252
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 = 31 255 248
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 = 31 255 248
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 = 7 255 224
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 = 3 255 192
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 = 0 126 0

```
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
.byte %00000001, %10000001, %10000000
.byte %00000000, %11000011, %00000000
.byte %00000000, %01100110, %00000000
.byte %00000000, %00111100, %00000000
.byte %00000000, %00011000, %00000000
.byte %00000000, %00111100, %00000000
.byte %00000000, %01100110, %00000000
.byte %00000000, %11000011, %00000000
.byte %00000001, %10000001, %10000000
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
.byte %00000000, %00000000, %00000000
```

Sprite Color

Sprite	Color register
#0	$D027
#1	$D028
#2	$D029
#3	$D02A
#4	$D02B
#5	$D02C
#6	$D02D
#7	$D02E

16 Colors

Set the color for Sprite 0

```
lda #5 // Green
sta $D027
```

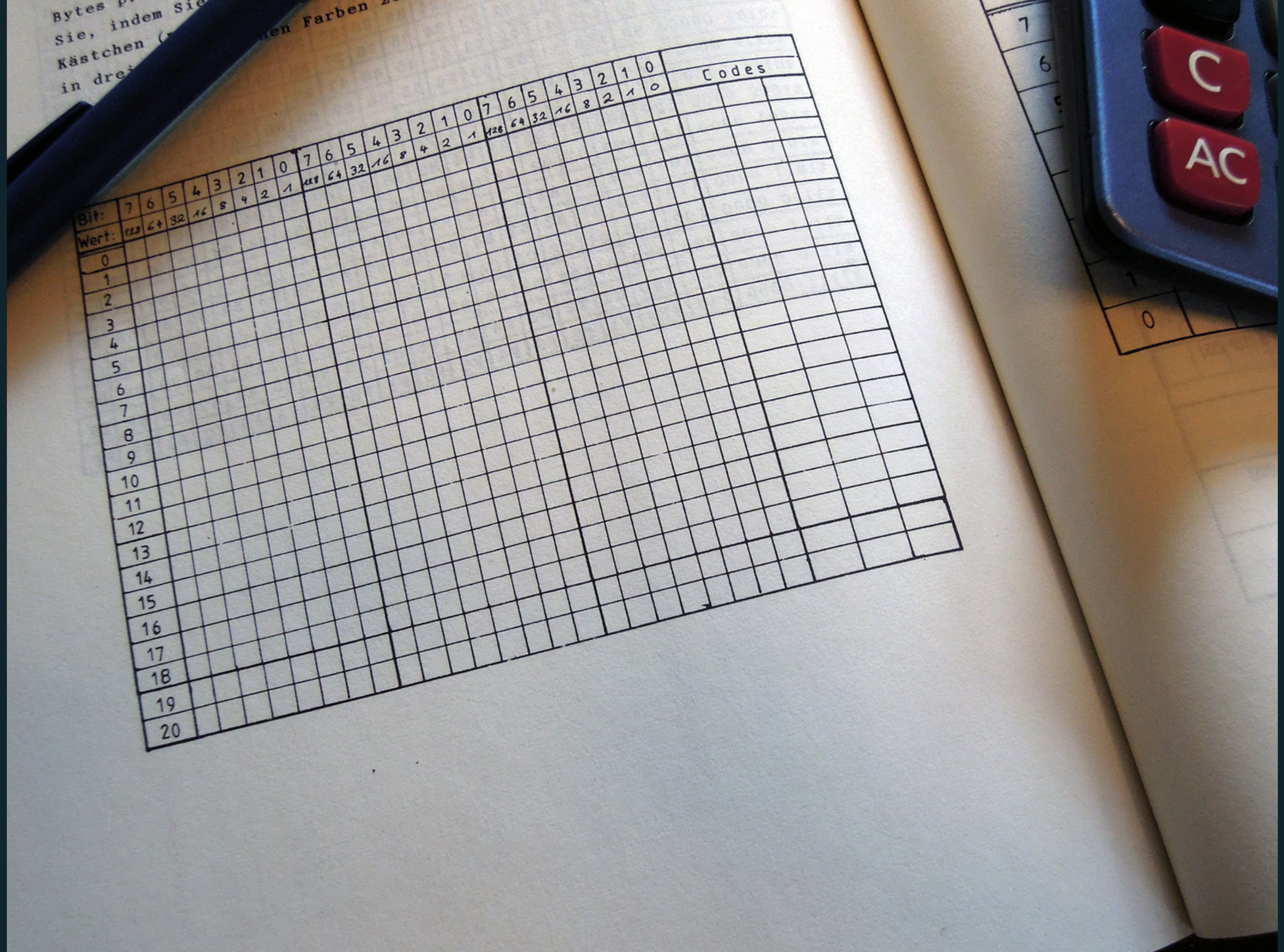
Multicolor Mode Sprites

— Supports 3 colors

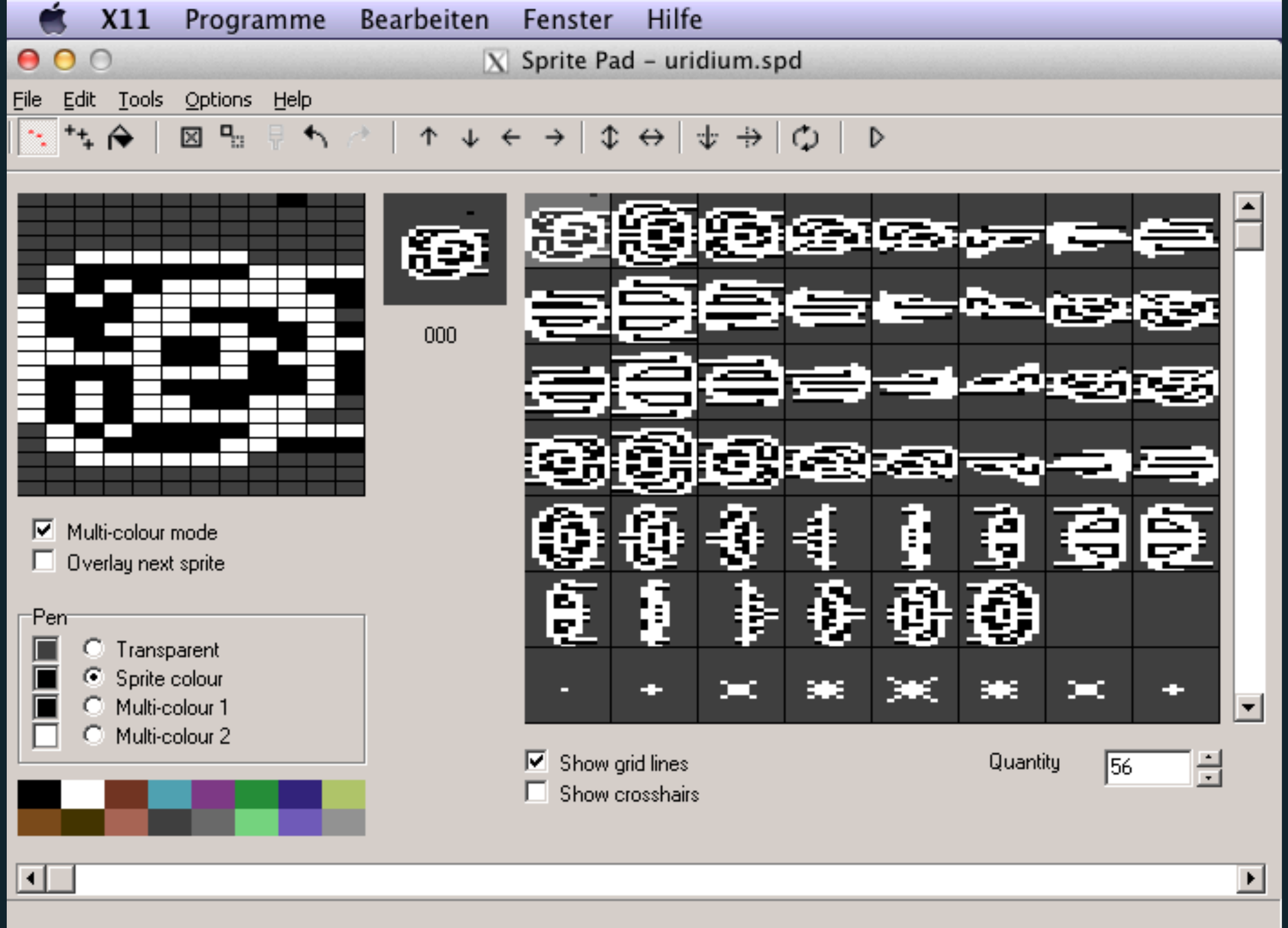
— Bits are grouped in pairs, forming pixels that are twice the width.

Bit couple	Color
%00	Transparent
%10	Sprite color register (sprite color; $D027-$D02E)
%01	Multicolor register #0 ($D025)
%11	Multicolor register #1 ($D026)

Creating a sprite by hand


Bytes p
Sie, indem Sie
Kästchen (
in drei
en Farben z
Bit: 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
Wert: 128 64 32 16 8 4 2 1 128 64 32 16 8 4 2 1 128 64 32 16 8 4 2 1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Codes
7
6
1
0
C
AC

Creating a sprite in SpritePad


X11 Programme Bearbeiten Fenster Hilfe
Sprite Pad – uridium.spd
File Edit Tools Options Help
000
Multi-colour mode
Overlay next sprite
Pen
Transparent
Sprite colour
Multi-colour 1
Multi-colour 2
Show grid lines
Show crosshairs
Quantity
56

Importing sprite data

```
*=$3000
.import binary "sprites.bin"
```

Display a sprite

— Enable/disable
— Position
— Data pointer

Enabling a sprite

— For a few sprite options like dis-/enable the VIC-2 uses only one register for all sprites. Since there are 8 sprites, we only need 1-bit in an 8-bit byte to determine if a sprite is on or off.

— Register: $D015

Sprite number	Bit Value
#7	128
#6	64
#5	32
#4	16
#3	8
#2	4
#1	2
#0	1

Turn on

```
%00000010 // Current sprite enable setting
OR
%00000001 // Sprite we want to enable
=
%00000011 // New sprite enable setting
```

Turn off

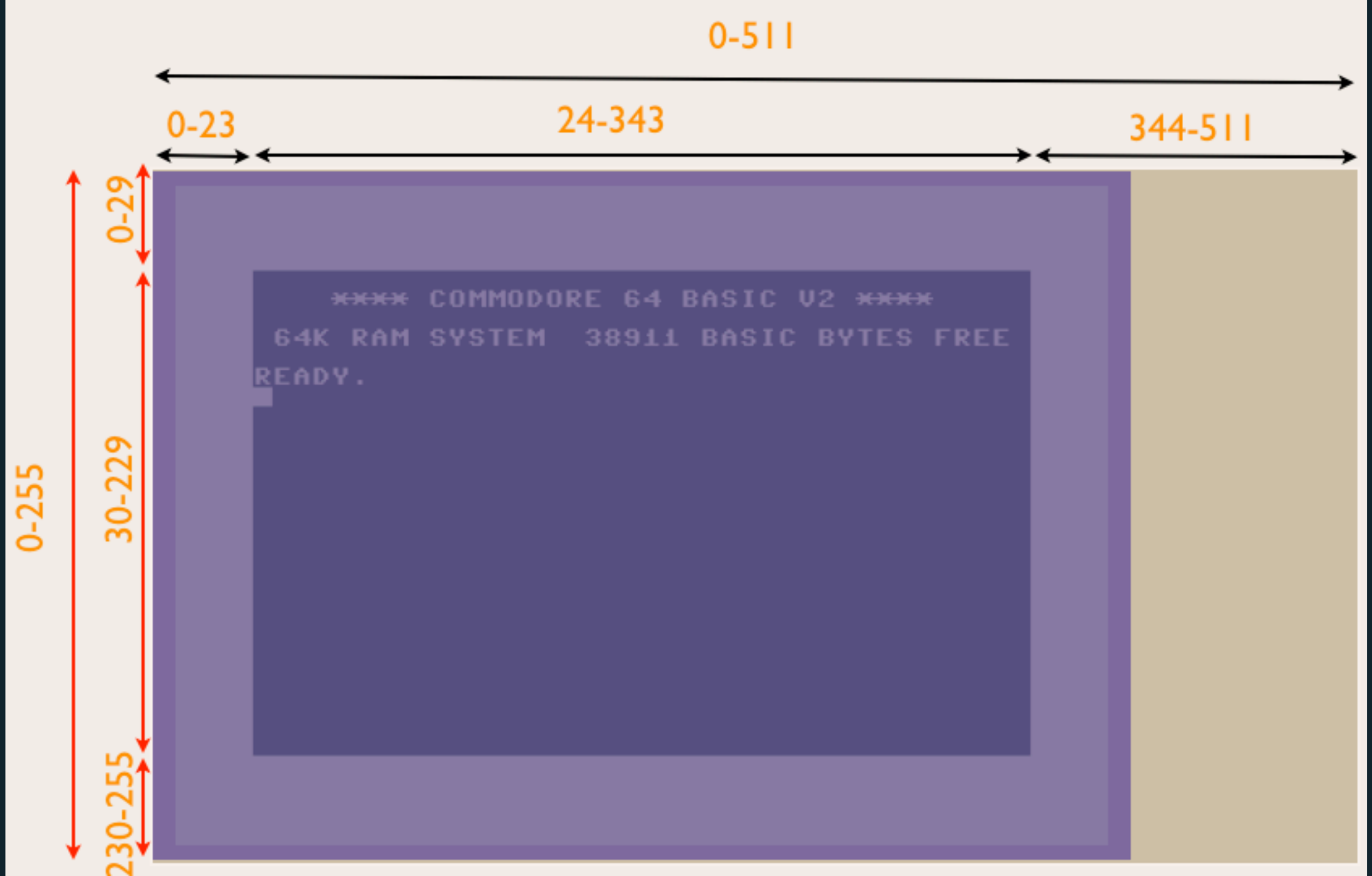
```
%00000001 // Sprite we want to turn off
XOR
%11111111 // Mask
=
%11111110 // Mask complement
AND
%00000011 // Current sprite enable setting
=
%00000010 // New sprite enable setting
```

```
.macro lib_sprite_enable(sprite_num, enable) {
  ldy sprite_num
  lda sprites.number_mask, y

  ldy #enable
  beq !disable+
!enable:
  ora sprites.enable_bits
  sta sprites.enable_bits
  jmp !done+
!disable:
  eor #$FF // get mask compliment
  and sprites.enable_bits
  sta sprites.enable_bits
!done:
}
```

Set the position a sprite

Sprite	x coordinate	y coordinate
#0	$D000	$D001
#1	$D002	$D003
#2	$D004	$D005
#3	$D006	$D007
#4	$D008	$D009
#5	$D00A	$D00B
#6	$D00C	$D00D
#7	$D00E	$D00F


0-511
0-23
24-343
344-511
0-29
30-229
230-255
0-255
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY.

High-bit

— To span more than 256 pixels in the horizontal direction you need a 9th bit!

— Register $D010 is a byte that contains the high-bit for each sprite

— Acts just like the sprite enable register, each bit corresponds to a specific sprite.

```
.macro lib_sprite_set_position_aaaa(sprite_num, xposh, xposl, ypos) {
  lda sprite_num // get sprite number
  asl // *2 as registers laid out 2 apart
  tay // copy accumulator to y register

  lda xposl // get XPos Low Byte
  sta sprites.positions, y // set the XPos sprite register
  lda ypos // get YPos
  sta sprites.positions+1, y // set the YPos sprite register

  ldy sprite_num
  lda sprites.number_mask, y // get sprite mask

  eor #$FF // get compliment
  and sprites.position_x_high_bits // clear the bit
  sta sprites.position_x_high_bits // and store

  ldy xposh // get XPos High Byte
  beq !end+ // skip if XPos High Byte is zero
  ldy sprite_num
  lda sprites.number_mask, y // get sprite mask

  ora sprites.position_x_high_bits // set the bit
  sta sprites.position_x_high_bits // and store
!end:
}
```

Sprite data pointers

— The VIC-2 has 8 registers that contain the memory location that contains the sprite data.

Sprite	Sprite pointer
#0	$07F8
#1	$07F9
#2	$07FA
#3	$07FB
#4	$07FC
#5	$07FD
#6	$07FE
#7	$07FF

The tricky part

— The VIC-II can only look at 16Kbyte of RAM at a time.
— A sprite has a fixed size of 64 Bytes.
— How do we reference 16kb of memory with only 256 possible values?
— Since each sprite is 64 bytes, as long as they are aligned to 64 byte boundaries we can reference those boundaries using only 256 values.
— 16384 / 64 = 256

Sprite data math

— Let's assume we are in the default VIC-II Bank 3 that ranges from $0000 to $3FFF and we want to store a sprite for sprite #0 starting at memory location $1600. For that location the sprite pointer in $07F8 would be $58.

— $1600 or Decimal 5632 divided by 64 equals Decimal 88 or Hexadecimal $58. When we put $58 in location $07F8 the VIC-II will read 64 Bytes starting at $1600 to fetch our sprite.

Animate the sprite

— Animating a sprite involves changing the sprite data pointer
value to point to the memory location containing the next frame of the
animation.

Demo