

Sprites

Nach soviel grauer Theorie kommt nun wieder Bewegung ins Spiel!

In diesem Kapitel geht es um die Programmierung von Sprites. Das sind kleine, bewegliche Objekte, deren Aussehen Sie innerhalb bestimmter Grenzen frei gestalten können und die sich dann beispielsweise für Spiele verwenden lassen.

Wir werden die Sprite-Programmierung zunächst in BASIC durchführen, um die grundlegenden Abläufe kennenzulernen. Aber keine Sorge, für jedes BASIC-Programm werden wir immer das entsprechende Assembler-Gegenstück erstellen.

Sprites werden vom Commodore 64 bereits seitens der Hardware unterstützt und das vereinfacht die Programmierung erheblich. Es werden standardmäßig 8 Sprites unterstützt, doch es sind durch Anwendung spezieller Techniken auch mehr Sprites möglich.

Es gibt einfarbige Sprites und mehrfarbige Sprites, wobei wir uns zunächst mit den einfarbigen Sprites beschäftigen wollen.

Einfarbige Sprites können eine von 16 Farben annehmen und maximal 24 Pixel breit bzw. maximal 21 Pixel hoch sein. Ein Sprite besteht also insgesamt aus 504 Punkten.

Bevor wir mit einem Sprite arbeiten können, müssen wir erst einmal wissen, wie es aussehen soll.

Doch wie sagt man dem C64, wie man sich das Sprite vorstellt?

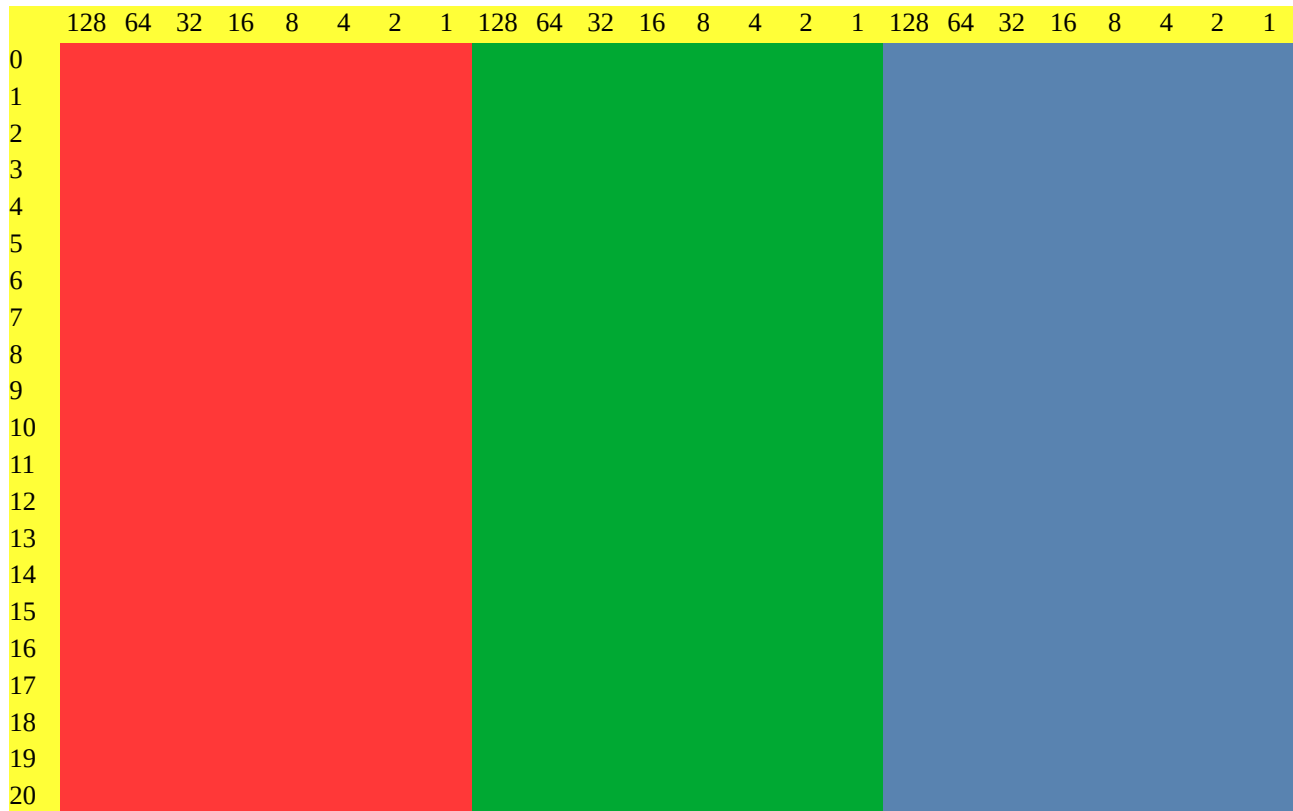
Dazu zeichnet man sich zunächst beispielsweise auf kariertem Papier einen Raster mit 24 Spalten und 21 Zeilen auf, wobei jede Zelle des Rasters einem der 504 Pixel des Sprites entspricht.

Das Sprite hat eine horizontale Auflösung von 24 Pixel und wenn man jedem Pixel ein Bit zuordnet, dann benötigen wir 3 Bytes ($3 \times 8 \text{ Bit}$ für 24 Pixel) um eine Zeile aus unserem Raster speichern zu können.

In vertikaler Richtung beträgt die Auflösung 21 Pixel, d.h. wir benötigen insgesamt ($3 \times 21 \text{ Bytes} = 63 \text{ Bytes}$) um das Aussehen unseres Sprites festzulegen.

Ein Block mit Spritedaten muss jedoch 64 Bytes umfassen, daher folgt auf das letzte Byte noch ein Platzhalter-Byte zum nächsten Block.

Unser Sprite-Raster sieht folgendermaßen aus:



Jede Zeile besteht wie gesagt aufgrund der horizontalen 24 Pixel aus 3 Bytes, der rote Bereich entspricht dem ersten, der grüne Bereich dem zweiten und der blaue Bereich dem dritten Byte in jeder Zeile.

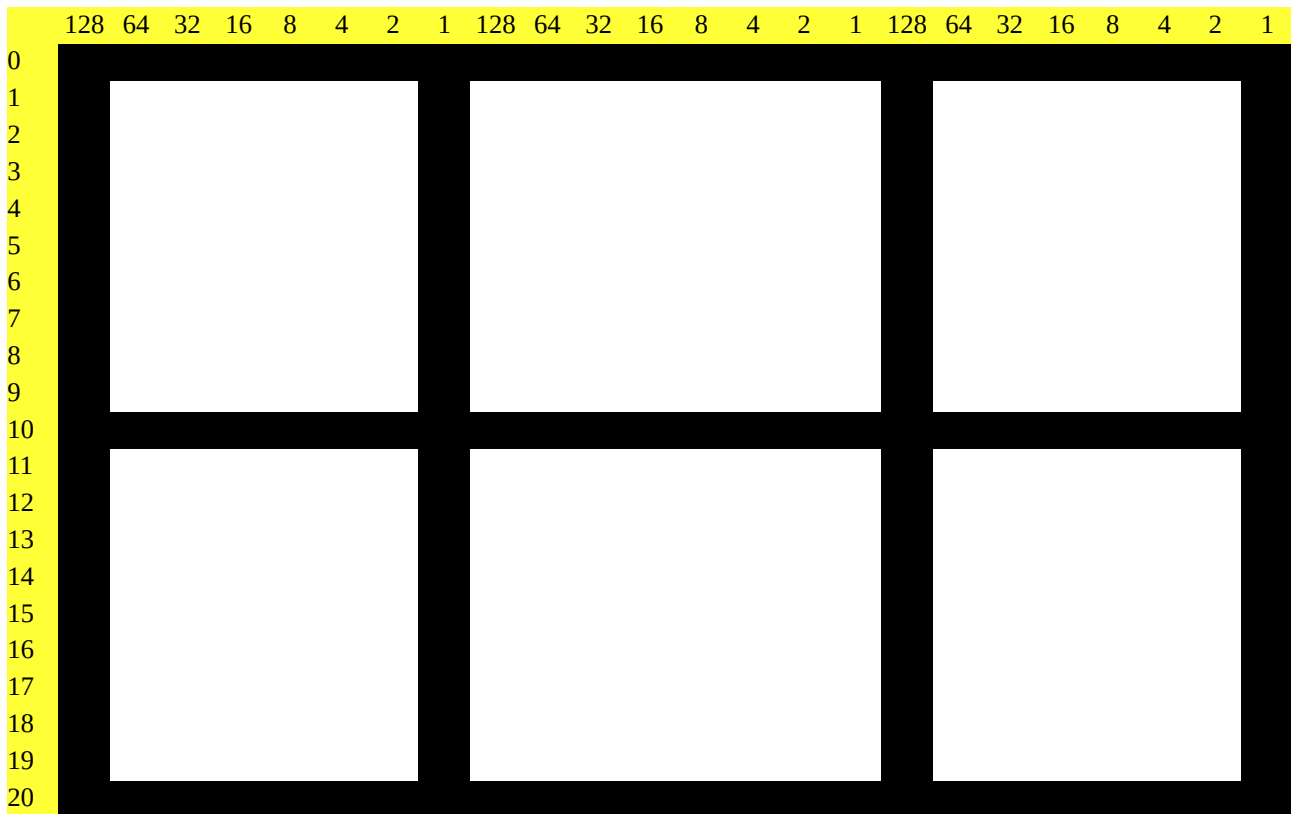
Über jedes Bit der drei Bytes schreiben wir die jeweilige Wertigkeit an der Stelle.

Diese beginnt jeweils mit 128 (2^7) und endet jeweils mit 1 (2^0)

Nehmen wir nun einen leeren Raster und zeichnen uns Pixel für Pixel ein einfach gehaltenes Sprite.

Wir füllen alle Stellen im Raster, an die wir einen Pixel setzen wollen.

Zeichnen Sie folgende einfache Form in den Raster. Jede ausgefüllte Rasterzelle entspricht einem gesetzten Pixel (das Bit hat also den Wert 1). An den weißen Stellen haben wir keinen Pixel gesetzt (das Bit hat also den Wert 0), d.h. hier scheint der Hintergrund durch.



Sehen wir uns das erste Byte in der ersten Zeile an, hier haben wir an jeder Bitposition eine ausgefüllte Zelle, also eine 1. Dies entspricht der binären Zahl %11111111 (hexadezimal \$FF bzw. dezimal 255)

Beim zweiten und dritten Byte ist ebenfalls an jeder Bitposition eine 1, d.h. wir haben auch hier den binären Wert %11111111 (hexadezimal \$FF bzw. dezimal 255)

Unsere erste Zeile wird also durch die drei Bytes 255,255,255 beschrieben.

Gehen wir nun zum ersten Byte in der zweiten Zeile.

Hier haben wir an den Bitpositionen 7 und 0 eine 1 stehen, d.h. wir haben hier die binäre Zahl %10000001 (hexadezimal \$81 bzw. dezimal 129)

Im zweiten Byte haben wir keine gesetzten Bits, d.h. wir haben hier den binären Wert %00000000 (hexadezimal \$00 bzw. dezimal 0)

Das dritte Byte entspricht dem ersten Byte, auch hier haben wir den binären Wert %10000001 (hexadezimal \$81 bzw. dezimal 129)

Die zweite Zeile wird also durch die drei Bytes 129,0,129 beschrieben.

Das setzen wir nun fort bis zur letzten Zeile und erhalten insgesamt folgende Zahlenwerte für die 21 Zeilen:

Erstes Byte	Zweites Byte	Drittes Byte
255	255	255
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
255	255	255
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
129	0	129
255	255	255

Soweit so gut. Aber wo speichern wir diese Zahlen nun ab? Da wir wie gesagt zunächst in BASIC programmieren wollen, legen wir die Zahlen in DATA-Zeilen ab.

Wir beginnen mit hohen Zeilennummern, da wir davor später noch weiteren BASIC-Code einfügen wollen.

```

READY.
1000 REM DATEN FUER SPRITE 0
1010 DATA 255,255,255
1020 DATA 129,0,129
1030 DATA 129,0,129
1040 DATA 129,0,129
1050 DATA 129,0,129
1060 DATA 129,0,129
1070 DATA 129,0,129
1080 DATA 129,0,129
1090 DATA 129,0,129
1100 DATA 129,0,129
1110 DATA 255,255,255
1120 DATA 129,0,129
1130 DATA 129,0,129
1140 DATA 129,0,129
1150 DATA 129,0,129
1160 DATA 129,0,129
1170 DATA 129,0,129
1180 DATA 129,0,129
1190 DATA 129,0,129
1200 DATA 129,0,129
1210 DATA 255,255,255

```

Nun müssen wir diese Daten an einem passenden Platz im Speicher ablegen.
Aber wo? Und wie sagen wir dann dem C64 wo wir die Daten für unser Sprite abgelegt haben?

Kümmern wir uns zuerst darum, wo wir unsere Daten im Speicher ablegen.

Sprite-Daten können wir nicht an jeder beliebigen Stelle im Speicher ablegen. Der Speicherbereich, den wir uns aussuchen, muss zwei Kriterien erfüllen:

- Er muss an einer durch 64 teilbaren Adresse beginnen
- Er muss 63 Byte durchgehend frei nutzbaren Platz bieten, denn wir dürfen unsere Sprite-Daten natürlich nicht in einen Speicherbereich schreiben, der bereits für andere Daten genutzt wird. 63 Byte deswegen, weil das 64. Byte nur als Platzhalter zum nächsten Block dient und nicht in die Spritedaten einfließt.

Durch 64 teilbare Adressen gibt es ja viele, aber wir müssen in dem Speicherbereich auch alle unsere 63 Bytes unterbringen können, ohne dabei andere Daten zu überschreiben.

Es hilft uns nichts, wenn die Adresse durch 64 teilbar ist, wir aber nur vielleicht 15 Bytes nutzen können, weil ab dem 16. Byte vielleicht bereits andere Daten folgen, die nicht überschrieben werden dürfen.

Glücklicherweise gibt es einige solcher frei verfügbaren Bereiche, welche diese Kriterien erfüllen und die wir daher zur Ablage unserer Sprite-Daten nutzen können.

Doch alles schön der Reihe nach.

Warum muss der Speicherbereich an einer durch 64 teilbaren Adresse beginnen?

Der Grund ist folgender:

Die 8 Speicherstellen von 2040 bis 2047 haben in Bezug auf Sprites eine wichtige Bedeutung.

Jede dieser 8 Speicherstellen ist einem Sprite zugeordnet, Speicherstelle 2040 ist Sprite 0 zugeordnet, Speicherstelle 2041 ist Sprite 1 zugeordnet, bis hin zur Speicherstelle 2047, welche Sprite 7 zugeordnet ist.

Jede dieser Speicherstellen enthält eine Blocknummer zwischen 0 und 255.

Diese Blocknummer multipliziert mit 64 ergibt dann jene Speicheradresse, die den Beginn des Speicherbereichs darstellt, in welchem wir die 63 Bytes Daten für unser Sprite ablegen.

Spielen wir das mal anhand der Speicherstelle 2040 durch, d.h. mit jener Speicherstelle, welche die Blocknummer für die Daten von Sprite 0 enthält.

Angenommen, sie enthielte die Blocknummer 0, dann würden die Spritedaten an Adresse $0 * 64 = 0$ beginnen. Diesen Block können wir jedoch nicht benutzen, denn wenn wir auf der Seite <https://www.c64-wiki.de/wiki/Zeropage> einen Blick auf die Belegung der Zeropage werfen, dann sehen wir, dass der Bereich von Adresse 0-63 bereits von anderen wichtigen Daten genutzt wird.

Probieren wir es mit Blocknummer 1, das wären dann die Adressen ab Adresse $1 * 64$, also Adresse 64. Tja, laut den Informationen auf der oben genannten Seite ist Block 1 leider auch schon vergeben.

Das geht leider weiter bis inklusive Block 10, also den Adressen 640 – 703.

Den Bereich mit der Blocknummer 11, also der Bereich von Adresse 704 bis 767, können wir jedoch für die Ablage unserer Spritedaten nutzen, da er nicht benutzt wird.

\$2C0 - \$2FF	704 - 767		Platz für Spritedatenblock 11, da nicht genutzt
---------------	-----------	--	---

Um es gleich vorweg zu nehmen:

Auch die Blöcke mit den Nummern 13, 14 und 15 können wir für unsere Spritedaten nutzen.

\$340 - \$37F	832 - 895		Platz für Spritedatenblock 13 (nur bei Nichtnutzung des Datasetten-/Kassettenpuffers!)
\$380 - \$3BF	896 - 959		Platz für Spritedatenblock 14 (nur bei Nichtnutzung des Datasetten-/Kassettenpuffers!)
\$3C0 - \$3FF	960 - 1023		Platz für Spritedatenblock 15 (nur bei Nichtnutzung des Datasetten-/Kassettenpuffers!)

Es gibt noch einiges anzumerken in Bezug auf die Blocknummern, doch das würde an dieser Stelle nur verwirren. Am Ende des Kapitels werde ich dies nachholen.

Festlegen der Blocknummer für die Spritedaten

Gut, dann nehmen wir doch für die Daten unseres Sprites gleich den ersten Block, den wir gefunden haben, also den mit der Nummer 11.

Wir fügen also folgende Zeile hinzu:

```
10 POKE 2040,11
```

Dadurch weiß der C64, dass die Daten für das Sprite 0 in Block 11 liegen, also ab der Speicheradresse 704 ($11 * 64$) zu finden sind.

Doch das ist erst die halbe Miete, denn bis jetzt stehen unsere Spritedaten nur in den DATA-Zeilen und noch nicht in dem Speicherblock 11.

Das Kopieren führen wir mittels folgender Schleife durch:

```
20 FOR I=0 TO 62  
30 READ A  
40 POKE 704+I,A  
50 NEXT I
```

Nun müssen wir noch eine ganze Reihe bestimmter Speicherstellen verändern, damit unser Sprite in der gewünschten Form auf dem Bildschirm angezeigt wird.

Typ des Sprites festlegen (einfarbig oder mehrfarbig)

In der Speicherstelle 53276 ist jedes der 8 Bits mit einem Sprite verbunden, Bit 0 mit Sprite 0 bis hin zu Bit 7, welches mit Sprite 7 verbunden ist. Setzt man ein Bit auf den Wert 0, dann gibt man dadurch an, dass es sich bei dem Sprite, welches mit diesem Bit verbunden ist, um ein einfarbiges Sprite handelt. Setzt man den Wert hingegen auf den Wert 1, dann gibt man dadurch an, dass es sich um ein mehrfarbiges Sprite handelt.

Da wir uns aktuell mit den einfarbigen Sprites beschäftigen, setzen wir das Bit an der Position 0 auf den Wert 0. Dadurch wird das Sprite 0 als einfarbig markiert.

Hier kommt uns nun unser Wissen über logische Verknüpfungen entgegen, denn wir müssen hier das Bit 0 auf den Wert 0 setzen.

Dazu brauchen wir folgende UND-Verknüpfung:

```
60 POKE 53276,PEEK(53276) AND (NOT (1))
```

Farbe des Sprites festlegen

Die Speicherstellen von 53287 bis 53294 enthalten die Farben für die 8 Sprites (falls es sich um einfarbige Sprites handelt)

Wir wählen für Sprite 0 die Farbe Weiß, also müssen wir den Wert 1 in die Speicherstelle 53287 schreiben.

```
70 POKE 53287,1
```

Festlegen der Spriteposition

Die Position eines Sprites wird durch eine Pixelposition in horizontaler und durch eine Pixelposition in vertikaler Richtung angegeben. In horizontaler Richtung (X) sind Werte von 0 bis

511 möglich und in vertikaler Richtung (Y) sind es Werte zwischen 0 und 255, wobei die Position X=0, Y=0 in der linken oberen Ecke des Bildschirms liegt.

Hier ist jedoch wirklich die linke obere Ecke des gesamten Bildschirms inklusive Rahmen gemeint, nicht die linke, obere Ecke des Ausgabebereichs, in dem beispielsweise die Textausgaben erfolgen.

Für die X-Koordinaten der 8 Sprites sind die Speicherstellen 53248, 53250, 53252, 53254, 53256, 53258, 53260 und 53262 zuständig, im Falle von Sprite 0 müssen wir die X-Koordinate also in der Speicherstelle 53248 ablegen.

Um das Sprite an den linken Rand des sichtbaren Bereichs zu positionieren, ist nicht, wie vielleicht vermutet, der Wert 0 erforderlich, sondern der Wert 24.

Die Einstellung der X-Koordinate führen wir mit dem Befehl

```
80 POKE 53248,24
```

durch.

Nun müssen wir uns noch um die Y-Koordinate kümmern.

Für die Y-Koordinaten der 8 Sprites sind die Speicherstellen 53249, 53251, 53253, 53255, 53257, 53259, 53261 und 53263 zuständig, im Falle von Sprite 0 müssen wir die Y-Koordinate also in der Speicherstelle 53249 ablegen.

Der Wert für den obersten Rand des sichtbaren Bereichs lautet 50.

Die Einstellung der Y-Koordinate für diese Position führen wir mit dem Befehl

```
90 POKE 53249,50
```

durch.

Festlegen der Sprite-Priorität in Bezug auf den Hintergrund

Dazu brauchen wir die Speicherstelle 53275. Auch diese Speicherstelle folgt dem bereits beschriebenen Schema und enthält für jedes Sprite ein eigenes Bit.

Enthält dieses Bit den Wert 0, dann hat das Sprite eine höhere Priorität als der Hintergrund und wird daher vor dem Hintergrund dargestellt. Enthält das jeweilige Bit jedoch den Wert 1, dann hat der Hintergrund höhere Priorität und das Sprite wird hinter dem Hintergrund dargestellt.

Wir entscheiden uns dafür, das Sprite vor dem Hintergrund darzustellen und setzen daher das Bit 0 auf den Wert 0.

```
100 POKE 53275,PEEK(53275) AND (NOT(1))
```

Sprite aktivieren

Nun müssen wir unser Sprite nur noch einschalten, damit es auch auf dem Bildschirm angezeigt wird.

In der Speicherstelle 53269 ist jedes der 8 Bits mit einem Sprite verbunden, Bit 0 mit Sprite 0 bis hin zu Bit 7, welches mit Sprite 7 verbunden ist. Setzt man ein Bit auf den Wert 1, dann wird das Sprite, das mit diesem Bit verbunden ist, angezeigt. Setzt man es umgekehrt auf den Wert 0, dann verschwindet das jeweilige Sprite.

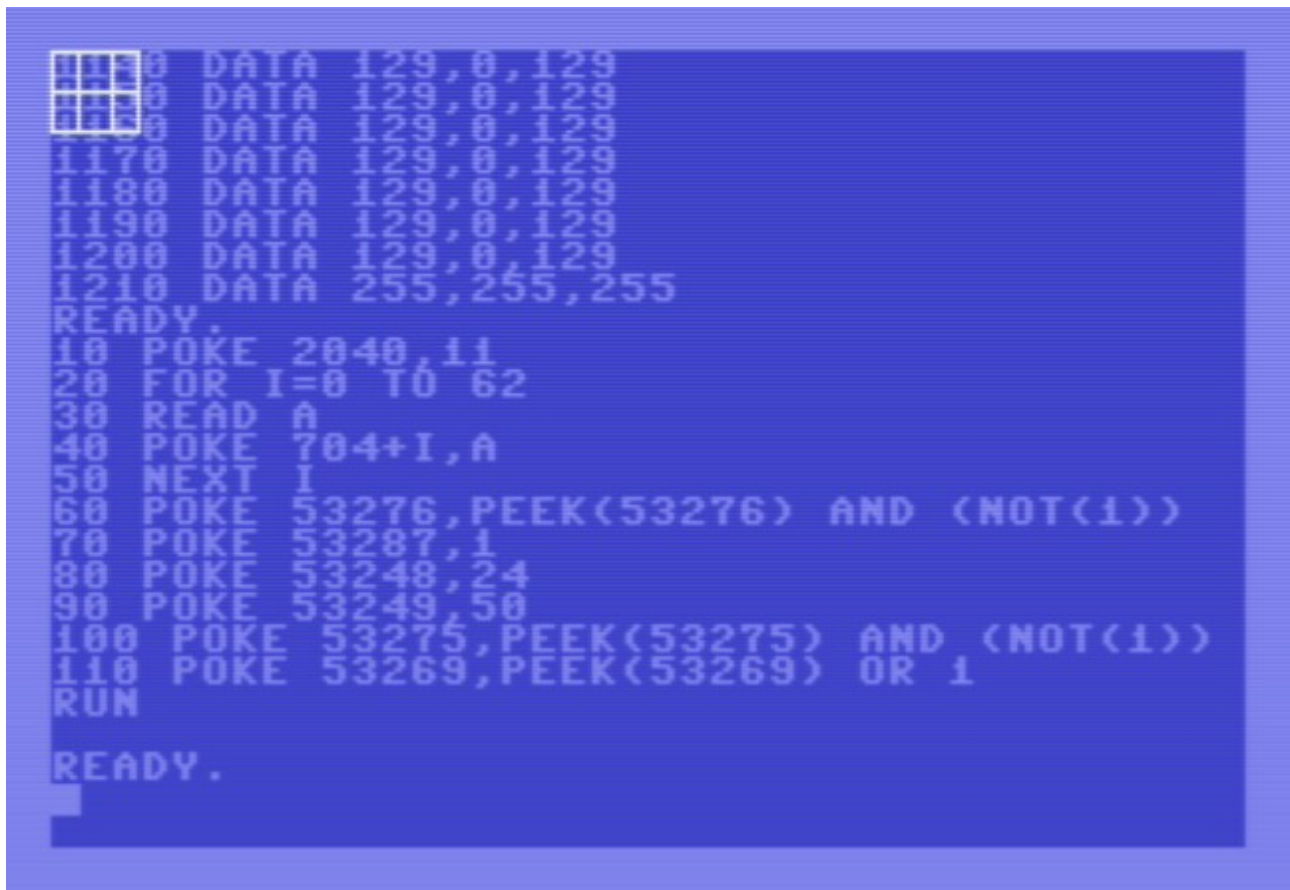
Wichtig:

Durch Aktivieren des Sprites wird das Sprite zwar grundsätzlich sichtbar gemacht, das bedeutet jedoch nicht, dass es sich gerade auch im sichtbaren Bereich auf dem Bildschirm befindet. Es kann je nach Koordinate beispielsweise vom Rahmen teilweise oder ganz verdeckt werden.

Setzen wir also Bit 0 in dieser Speicherstelle auf den Wert 1:

```
110 POKE 53269,PEEK(53269) OR 1
```

Wenn wir das Programm nun mit RUN starten, dann sollte folgendes zu sehen sein.



```
10 DATA 129,0,129
20 DATA 129,0,129
30 DATA 129,0,129
40 DATA 129,0,129
50 DATA 129,0,129
60 DATA 129,0,129
70 DATA 129,0,129
80 DATA 255,255,255
READY.
10 POKE 2040,11
20 FOR I=0 TO 62
30 READ A
40 POKE 704+I,A
50 NEXT I
60 POKE 53276,PEEK(53276) AND (NOT(1))
70 POKE 53287,1
80 POKE 53248,24
90 POKE 53249,50
100 POKE 53275,PEEK(53275) AND (NOT(1))
110 POKE 53269,PEEK(53269) OR 1
RUN
READY.
```

Es hat also soweit alles funktioniert und wir haben unser erstes Sprite auf dem Bildschirm dargestellt.

Wählen wir doch mal eine andere Farbe, z.B. Gelb (Farbcode 7) und geben gleich im Direktmodus den Befehl

```
POKE 53287,7
```

ein.

Das Sprite sollte nun in gelb angezeigt werden.

Lassen wir unser Sprite mal verschwinden? Aber sicher, das funktioniert mit dem Befehl

```
POKE 53269,PEEK(53269) AND (NOT(1))
```

Das Sprite sollte nun verschwunden sein.

Sichtbar machen können wir es wieder mit dem Befehl

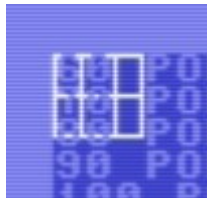
```
POKE 53269,PEEK(53269) OR 1
```

Das Sprite sollte nun wieder zu sehen sein.

Legen wir es doch mal hinter den Hintergrund, dazu ist der Befehl

```
POKE 53275,PEEK(53275) OR 1
```

nötig.



Nun befinden sich die BASIC-Zeilennummern im Vordergrund und überdecken das Sprite an manchen Stellen.

Hier zum Vergleich die vorherige Anzeige, bei der das Sprite im Vordergrund liegt und die Zeilennummern an manchen Stellen verdeckt.



Experimentieren wir nun ein wenig mit der Position des Sprites.

Verändern wir doch mal die X-Koordinate auf den Wert 100, was über den Befehl

```
POKE 53248,100
```

möglich ist.



Wichtig:

Wenn wir für unser Sprite eine X-Koordinate größer als 255 wählen, dann müssen wir hier einen anderen Weg einschlagen, denn in einem Byte kann man ja nur Werte zwischen 0 und 255 ablegen.

Hier sehen Sie die Position bei einer X-Koordinate von 255, also die höchstmögliche X-Koordinate, die in der Speicherstelle 53248 möglich ist.

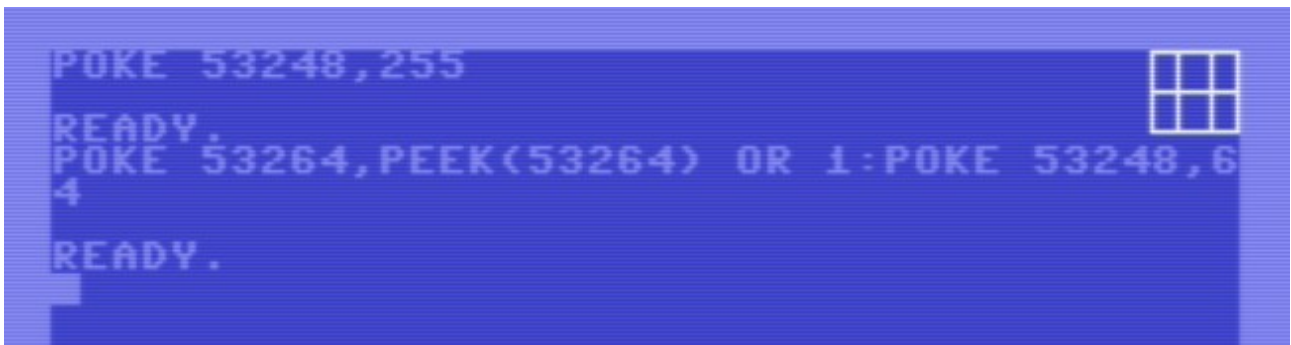


Wollen wir das Sprite an den rechten Rand des sichtbaren Bereichs positionieren, also auf die Position 320, dann müssen wir die Speicherstelle 53264 zu Hilfe nehmen.

Auch diese Speicherstelle enthält nach dem bereits erwähnten Schema für jedes Sprite ein eigenes Bit. Dieses Bit dient als zusätzliches Bit für die Darstellung von X-Koordinaten, welche größer als 255 sind und hat in Bezug auf die X-Koordinate die Wertigkeit 256.

Wir wollen das Sprite auf die X-Koordinate 320 setzen, d.h. wir müssen das Bit 0 (für das Sprite 0) in der Speicherstelle 53264 auf den Wert 1 setzen und den Rest, also was vom Wert 256 noch auf den Wert 320 fehlt, schreiben wir wie gehabt in die Speicherstelle 53248.

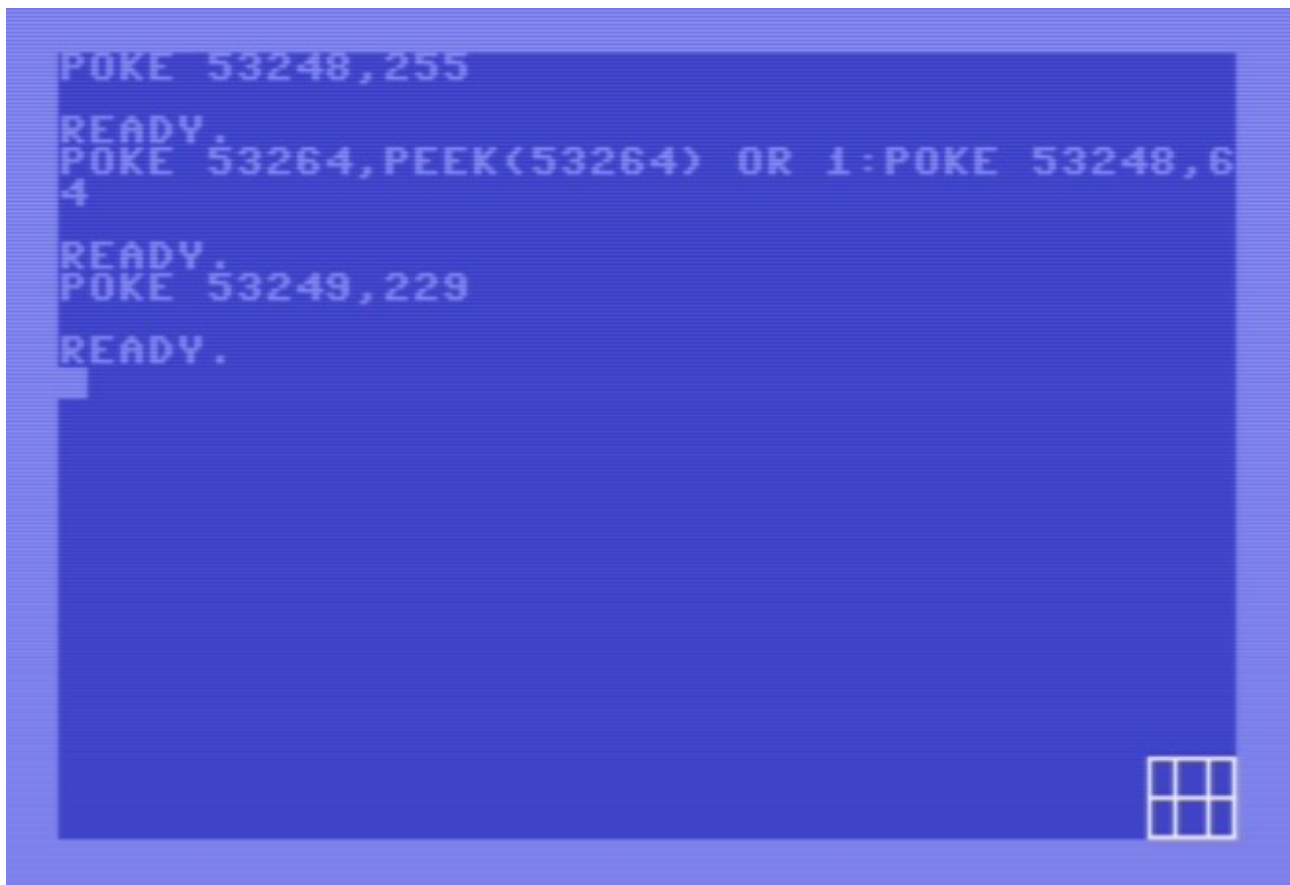
```
POKE 53264,PEEK(53264) OR 1:POKE 53248,64
```



Wichtig ist hier, dass wir das Bit 0 in Speicherstelle 53264 wieder auf 0 setzen, wenn wir die X-Koordinate auf einen Wert zwischen 0 und 255 setzen wollen.

Dies funktioniert mit dem Befehl `POKE 53264,PEEK(53264) AND (NOT(1))`

Nun verschieben wir noch mit dem Befehl `POKE 53249,229` das Sprite an den unteren Rand des sichtbaren Bildschirmbereichs.



Wir haben auch die Möglichkeit, das Sprite sowohl in horizontaler als auch in vertikaler Richtung zu vergrößern. Die Auflösung wird dadurch nicht verdoppelt, das Sprite wird nur doppelt so breit oder hoch dargestellt.

Für eine horizontale Vergrößerung ist die Speicherstelle 53277 zuständig. Auch hier ist jedes Sprite mit einem eigenen Bit vertreten. Setzt man es auf den Wert 1, so wird das Sprite in horizontaler Richtung verdoppelt. Setzt man es umgekehrt auf den Wert 0, so wird das Sprite in Normalgröße angezeigt.

Hier eine Vergrößerung des Sprites in horizontaler Richtung:



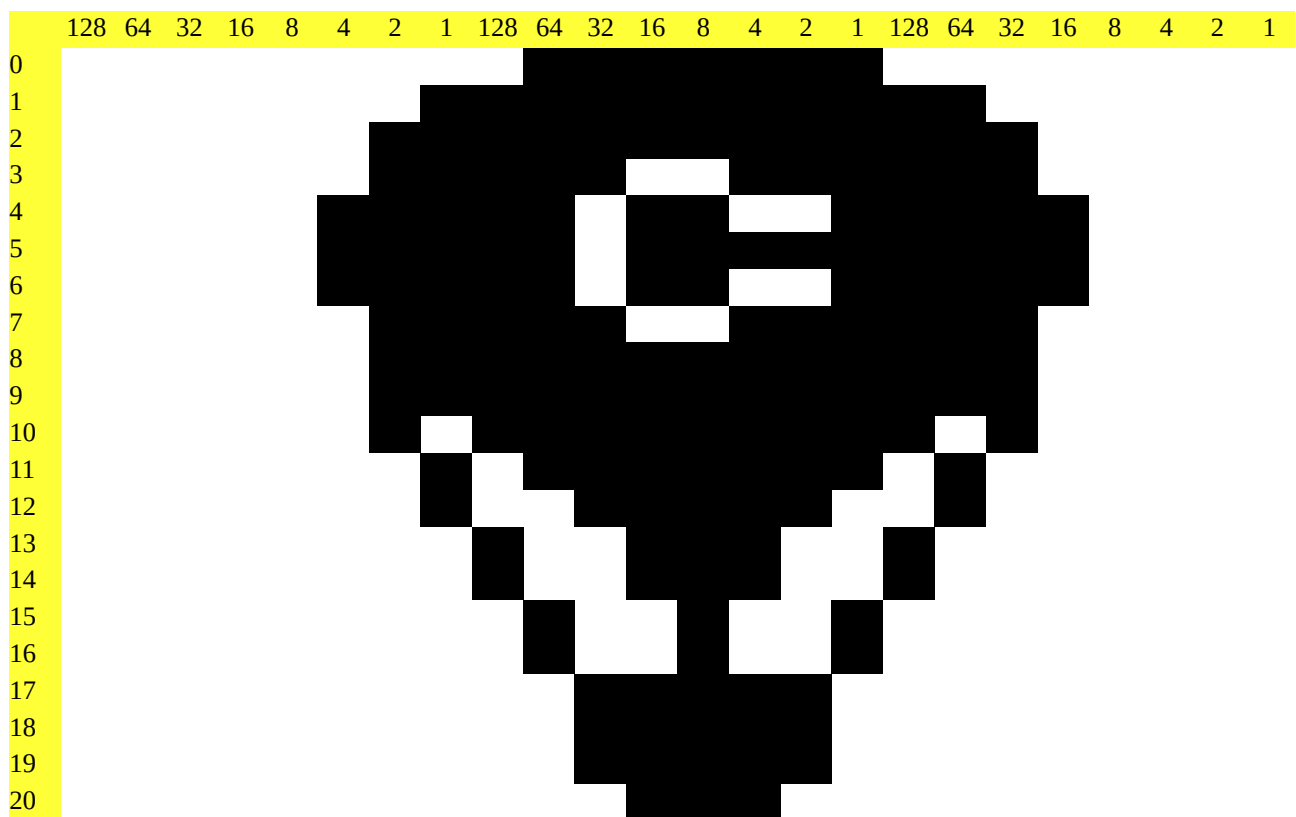
Für eine vertikale Vergrößerung ist die Speicherstelle 53271 zuständig. Auch hier ist wieder jedes Sprite mit einem eigenen Bit vertreten. Setzt man es auf den Wert 1, so wird das Sprite in vertikaler Richtung verdoppelt. Setzt man es umgekehrt auf den Wert 0, so wird das Sprite in Normalgröße angezeigt.

Hier eine Vergrößerung des Sprites in vertikaler Richtung:



Fügen wir nun ein weiteres Sprite hinzu.

Aus dem Handbuch des C64 kennen Sie sicherlich diesen Ballon, der dort als Beispiel für die Sprite-Programmierung verwendet wird. Bauen wir uns diesen doch mal nach.



Damit Sie sich nicht die Mühe machen brauchen, habe ich hier gleich die Tabelle mit den entsprechenden Bytes für Sie.

Erstes Byte	Zweites Byte	Drittes Byte
0	127	0
1	255	192
3	255	224
3	231	224
7	217	240
7	223	240
7	217	240
3	231	224
3	255	224
3	255	224
2	255	160
1	127	64
1	62	64
0	156	128
0	156	128
0	73	0
0	73	0
0	62	0
0	62	0
0	62	0
0	28	0

Wie bei unserem ersten Sprite legen wir diese Daten wieder in DATA-Zeilen ab.

```

1210 DATA 255,255,255
READY.
1220 REM DATEN FUER SPRITE 1
1230 DATA 0,127,0
1240 DATA 1,255,192
1250 DATA 3,255,224
1260 DATA 3,231,224
1270 DATA 7,217,240
1280 DATA 7,223,240
1290 DATA 7,217,240
1300 DATA 3,231,224
1310 DATA 3,255,224
1320 DATA 3,255,224
1330 DATA 2,255,160
1340 DATA 1,127,64
1350 DATA 1,62,64
1360 DATA 0,156,128
1370 DATA 0,156,128
1380 DATA 0,73,0
1390 DATA 0,73,0
1400 DATA 0,62,0
1410 DATA 0,62,0
1420 DATA 0,62,0
1430 DATA 0,28,0

```

Dann führen wir exakt dieselben Schritte durch, die wir auch beim ersten Sprite durchgeführt haben.

Als Speicherort werden wir für unser zweites Sprite den Block Nummer 13 verwenden, denn wie bereits erwähnt, stehen die Blöcke 13, 14 und 15 zur freien Verfügung.

Wir ergänzen also folgende Zeile:

```
120 POKE 2041,13
```

Da wir dieses mal ja Sprite 1 meinen, müssen wir hier die Speicherstelle 2041 verwenden.

Nun kopieren wir die Spritedaten in den Block Nummer 13, dieser hat die Startadresse 832.

```
130 FOR I=0 TO 62
140 READ A
150 POKE 832+I,A
160 NEXT I
```

Typ des Sprites festlegen (einfarbig oder mehrfarbig)

Da es sich wieder um ein einfarbigen Sprite handelt, setzen wir das Bit an der Position 1 auf den Wert 0. Dadurch wird das Sprite 1 als einfarbig markiert.

Dazu brauchen wir folgende UND-Verknüpfung:

```
170 POKE 53276,PEEK(53276) AND (NOT (2))
```

Farbe des Sprites festlegen

Wir wählen für Sprite 1 die Farbe Gelb, also müssen wir den Wert 7 in die Speicherstelle 53288 schreiben.

```
180 POKE 53288,7
```

Festlegen der Spriteposition

Nehmen wir für dieses Sprite die X-Koordinate 160 und die Y-Koordinate 140.

```
190 POKE 53250,160
200 POKE 53251,140
```

Festlegen der Sprite-Priorität in Bezug auf den Hintergrund

Wir entscheiden uns auch bei diesem Sprite dafür, dass das Sprite vor dem Hintergrund dargestellt wird und wählen daher den Wert 0 für das Bit 1 in der Speicherstelle 53275.

```
210 POKE 53275,PEEK(53275) AND (NOT(2))
```

Sprite aktivieren

Setzen wir also Bit 1 in der Speicherstelle 53269 auf den Wert 1.

```
220 POKE 53269,PEEK(53269) OR 2
```

Nun starten wir das Programm mit RUN und es wird nun auch das zweite Sprite angezeigt.

```
100  FOR I=0 TO 62
110  READ A
120  POKE 704+I,A
130  NEXT I
140  POKE 53276,PEEK(53276) AND (NOT(1))
150  POKE 53287,1
160  POKE 53248,24
170  POKE 53249,50
180  POKE 53275,PEEK(53275) AND (NOT(1))
190  POKE 53269,PEEK(53269) OR 1
200  POKE 2041,13
210  FOR I=0 TO 62
220  READ A
230  POKE 832+I,A
240  NEXT I
250  POKE 53276,PEEK(53276) AND (NOT(2))
260  POKE 53288,7
270  POKE 53250,160
280  POKE 53251,140
290  POKE 53275,PEEK(53275) AND (NOT(2))
300  POKE 53269,PEEK(53269) OR 2
310  REM DATEN FUER SPRITE 0
320  READY.
```



Nun bauen wir uns noch ein drittes Sprite, das Zeichnen ist dieses mal sehr einfach, da es die komplette Fläche ausfüllt.

	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																								

Das macht die Ermittlung der Werte für die DATA-Zeilen natürlich auch recht einfach.
Wir benötigen für die 21 DATA-Zeilen immer denselben Inhalt 255,255,255.

[illegible]

Dann ergänzen wir noch folgende Zeilen, um die Einstellungen für das Sprite 2 vorzunehmen. Als Speicherblock verwenden wir dieses mal den Block 14 (Startadresse 896).

Als Farbe wählen wir Türkis, für die X-Koordinate 100 und für die Y-Koordinate ebenfalls 100.

```

230 POKE 2042,14
240 FOR I=0 TO 62
250 READ A
260 POKE 896+I,A
270 NEXT I
280 POKE 53276,PEEK(53276) AND (NOT(4))
290 POKE 53289,3
300 POKE 53252,100
310 POKE 53253,100
320 POKE 53275,PEEK(53275) AND (NOT(4))
330 POKE 53269,PEEK(53269) OR 4

```

Wir starten das Programm wieder mit RUN und nun sehen wir auch unser drittes Sprite.

Wie auch bei dem Ballon sieht man bei diesem Sprite besonders gut, dass es vor dem Hintergrund liegt, weil es die Werte in den DATA-Zeilen verdeckt.



So, nun wollen wir das alles mal in Assembler umsetzen. Logische Verknüpfungen sind hier sehr stark vertreten. Wenn Sie diesbezüglich fit sind, sollten Sie keine Probleme damit haben, den Code nachvollziehen zu können.

Ich habe die drei 64 Byte Blöcke für die Spritedaten ab Adresse \$1500 abgelegt. Rechts unten sehen Sie den Wert \$00 welcher den Block für Sprite 2 abschließt. Damit Sie diese vielen Zahlen und auch das Assembler-Programm nicht abtippen müssen, habe ich es auf der Diskette zur Verfügung gestellt.



Hier der Code für Sprite 0:

,15CC0	A9	0B		LDA	#0B
,15CC2	8D	F8	07	STA	07F8
,15CC5	A2	00		LDX	#00
,15CC7	BD	00	15	LDA	1500,X
,15CCA	9D	C0	02	STA	02C0,X
,15CCD	E8			INX	
,15CCE	E0	40		CPX	#40
,15CD0	D0	F5		BNE	15C7
,15CD2	A9	01		LDA	#01
,15CD4	49	FF		EOR	#FF
,15CD6	2D	1C	D0	AND	D01C
,15CD9	8D	1C	D0	STA	D01C
,15CDC	A9	01		LDA	#01
,15CDE	8D	27	D0	STA	D027
,15FE1	A9	18		LDA	#18
,15FE3	8D	00	D0	STA	D000
,15FE6	A9	32		LDA	#32
,15FE8	8D	01	D0	STA	D001
,15FEB	A9	01		LDA	#01
,15FED	49	FF		EOR	#FF
,15FEF	2D	1B	D0	AND	D01B
,15FF2	8D	1B	D0	STA	D01B
,15FF5	A9	01		LDA	#01
,15FF7	0D	15	D0	ORA	D015
,15FFA	8D	15	D0	STA	D015

Hier der Code für Sprite 1:

,15FFD	A9	0D		LDA	#0D
,15FFF	8D	F9	07	STA	07F9
,16002	A2	00		LDX	#00
,16004	BD	40	15	LDA	1540,X
,16007	9D	40	03	STA	0340,X
,1600A	E8			INX	
,1600B	E0	40		CPX	#40
,1600D	D0	F5		BNE	1604
,1600F	A9	02		LDA	#02
,16111	49	FF		EOR	#FF
,16113	2D	1C	D0	AND	D01C
,16116	8D	1C	D0	STA	D01C
,16119	A9	07		LDA	#07
,1611B	8D	28	D0	STA	D028
,1611E	A9	A0		LDA	#A0
,16220	8D	02	D0	STA	D002
,16223	A9	8C		LDA	#8C
,16225	8D	03	D0	STA	D003
,16228	A9	02		LDA	#02
,1622A	49	FF		EOR	#FF
,1622C	2D	1B	D0	AND	D01B
,1622F	8D	1B	D0	STA	D01B
,16332	A9	02		LDA	#02
,16334	0D	15	D0	ORA	D015
,16337	8D	15	D0	STA	D015

Hier der Code für Sprite 2:

,163A	A9	0E		LDA	#0E
,163C	8D	FA	07	STA	07FA
,163F	A2	00		LDX	#00
,1641	BD	80	15	LDA	1580,X
,1644	9D	80	03	STA	0380,X
,1647	E8			INX	
,1648	E0	40		CPX	#40
,164A	D0	F5		BNE	1641
,164C	A9	04		LDA	#04
,164E	49	FF		EOR	#FF
,1650	2D	1C	D0	AND	D01C
,1653	8D	1C	D0	STA	D01C
,1656	A9	03		LDA	#03
,1658	8D	29	D0	STA	D029
,165B	A9	64		LDA	#64
,165D	8D	04	D0	STA	D004
,1660	8D	05	D0	STA	D005
,1663	A9	04		LDA	#04
,1665	0D	15	D0	ORA	D015
,1668	8D	15	D0	STA	D015
,166B	60			RTS	

Wechseln Sie mit X zurück nach BASIC und starten das Programm mit SYS 5568.

Als erstes fällt sofort der enorme Unterschied in der Geschwindigkeit auf. Die Sprites werden nach der Eingabe von SYS 5568 augenblicklich angezeigt. Bei der BASIC-Version dauerte das um einiges länger.

Ich werde den Code anhand von Sprite 0 erklären, der Code für Sprite 1 und Sprite 2 ist bis auf die unterschiedlichen Einstellungen für Blocknummer, Farbe, Position usw. identisch.

\$15C0 - \$15C2

```
LDA #0B
STA $07FB
```

Ist das Gegenstück zu

```
POKE 2040,11
```

\$15C5 - \$15D0

```
LDX #00
LDA 1500,X
STA 02C0,X
INX
CPX #40
BNE zu LDA Befehl
```

Eine Schleife, welche die 63 Bytes an Daten für Sprite 0, die sich ab Adresse \$1500 im Speicher befinden, in den Block 11 (Startadresse \$02C0, dezimal 704) umkopiert.

Sie ist das Gegenstück zur BASIC-Schleife, welche die Daten aus den DATA-Zeilen in den Block 11 umkopiert.

\$15D2 - \$15D9 (Einstellung dass Sprite 0 einfärbig ist)

```
LDA #01  
EOR #FF  
AND D01C  
STA D01C
```

Ist das Gegenstück zu

```
POKE 53276,PEEK(53276) AND (NOT(1))
```

In Assembler sind hier die beiden Zahlen vertauscht, d.h. zuerst wird der Akkumulator mit dem Wert \$01 geladen, dann durch Anwendung von EOR #FF das Einerkomplement gebildet, dieses dann durch AND mit dem Inhalt der Speicherstelle D01C (dezimal 53276) verknüpft und durch STA D01C wird das Ergebnis wieder zurück in die Speicherstelle D01C geschrieben.

In BASIC würde man dies so schreiben:

```
POKE 53276,(NOT (1)) AND PEEK(53276)
```

\$15DC - \$15DE (Farbe Weiß einstellen)

```
LDA #01  
STA D027
```

Ist das Gegenstück zu

```
POKE 53287,1
```

\$15E1 - \$15E3 (X-Koordinate auf den Wert 24 einstellen)

```
LDA #18  
STA D000
```

Ist das Gegenstück zu

```
POKE 53248,24
```

\$15E6 - \$15E8 (Y-Koordinate auf den Wert 50 einstellen)

```
LDA #32  
STA D001
```

Ist das Gegenstück zu

```
POKE 53249,50
```

\$15EB - \$15F2 (Einstellen, dass Sprite vor dem Hintergrund liegt)

```
LDA #01
EOR #FF
AND D01B
STA D01B
```

Ist das Gegenstück zu

POKE 53275,PEEK(53275) AND (NOT(1))

Die AND-Verknüpfung funktioniert analog wie vorhin bei der Speicherstelle 53276.

\$15F5 - \$15FA (Sprite aktivieren)

```
LDA #01
ORA D015
STA D015
```

Ist das Gegenstück zu

POKE 53269,PEEK(53269) OR 1

Hier ist es ähnlich wie vorhin bei der AND-Verknüpfung. Die beiden Zahlen sind vertauscht, d.h. zuerst wird der Akkumulator mit dem Wert 1 geladen, dieser dann durch OR mit dem Inhalt der Speicherstelle D015 (dezimal 53269) verknüpft und durch STA D015 wird das Ergebnis wieder zurück in die Speicherstelle D015 geschrieben.

Sprite-Priorität

Ich habe Sprite 0 auf die Position 87,87 verschoben und Sprite 1 auf die Position 110,110 damit man die Prioritäten der Sprites erkennen kann.



Hier sieht man, dass Sprite 2 (Quadrat) sowohl von Sprite 0 (Gitter) als auch von Sprite 1 (Ballon) überdeckt wird.

Die weißen Linien des Gitters überdecken die türkisen Stellen des Quadrats.

Das liegt daran, dass Sprite 0 die höchste Priorität hat. Das geht weiter bis Sprite 7 mit der niedrigsten Priorität.

Die weißen Linien des Gitters würden auch den Ballon stellenweise überdecken, weil die Priorität des Gitters höher ist.

Je niedriger die Nummer des Sprites ist, desto höher ist die Priorität gegenüber den Sprites mit höheren Nummern.

Diese Prioritäten kann man nicht verändern. Sprite 0 wird also immer die höchste und Sprite 7 immer die niedrigste Priorität haben. Es ist also nicht möglich, beispielsweise Sprite 2 eine höhere Priorität als Sprite 1 zu geben.

Die Priorität der Sprites untereinander ist nicht zu verwechseln mit der Priorität, welche die einzelnen Sprites in Bezug auf den Hintergrund haben.

Auf dem Bild sieht man, dass alle Sprites den BASIC-Code verdecken, weil wir dies bei jedem Sprite über die Speicherstelle 53275 so eingestellt haben.

TODO: Detailliertere Erklärung zu den Speicherblöcken und größeres Beispielprogramm