

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?

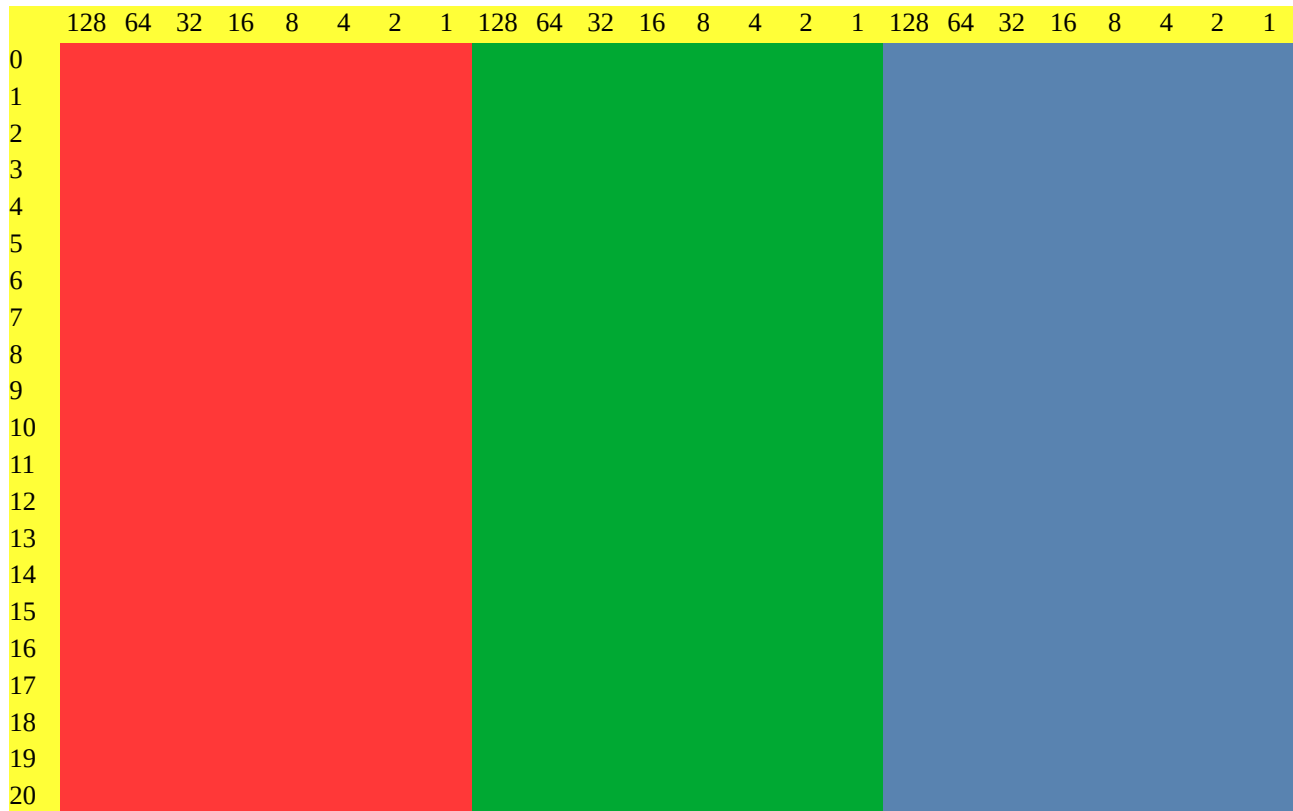
Dazu zeichnet man sich zunächst beispielsweise auf kariertem Papier einen Raster mit 24 Spalten und 21 Zeilen auf, wobei jede Zelle des Rasters einem der 504 Pixel des Sprites entspricht.

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.

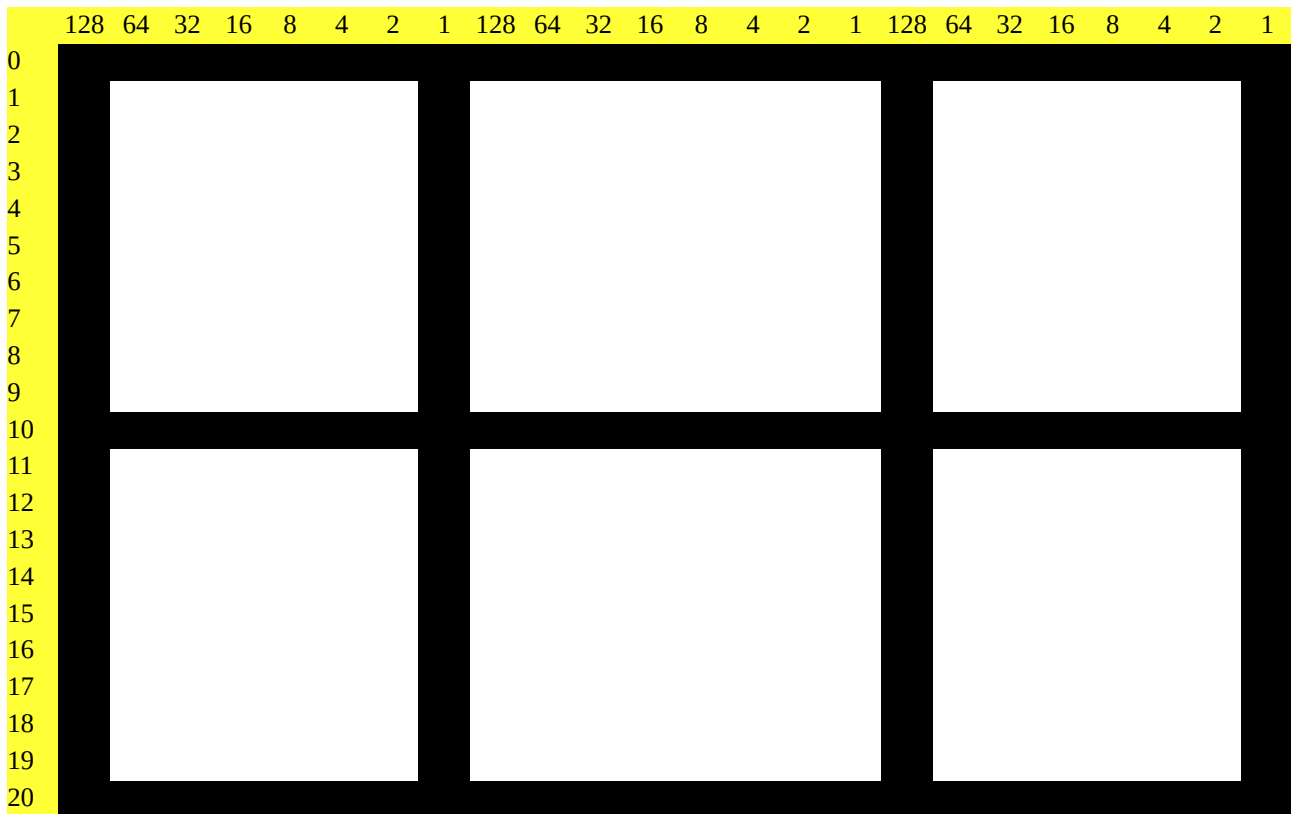
Über jedes Bit der drei Bytes schreiben wir die jeweilige Wertigkeit an der Stelle.

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.

Spiele 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.

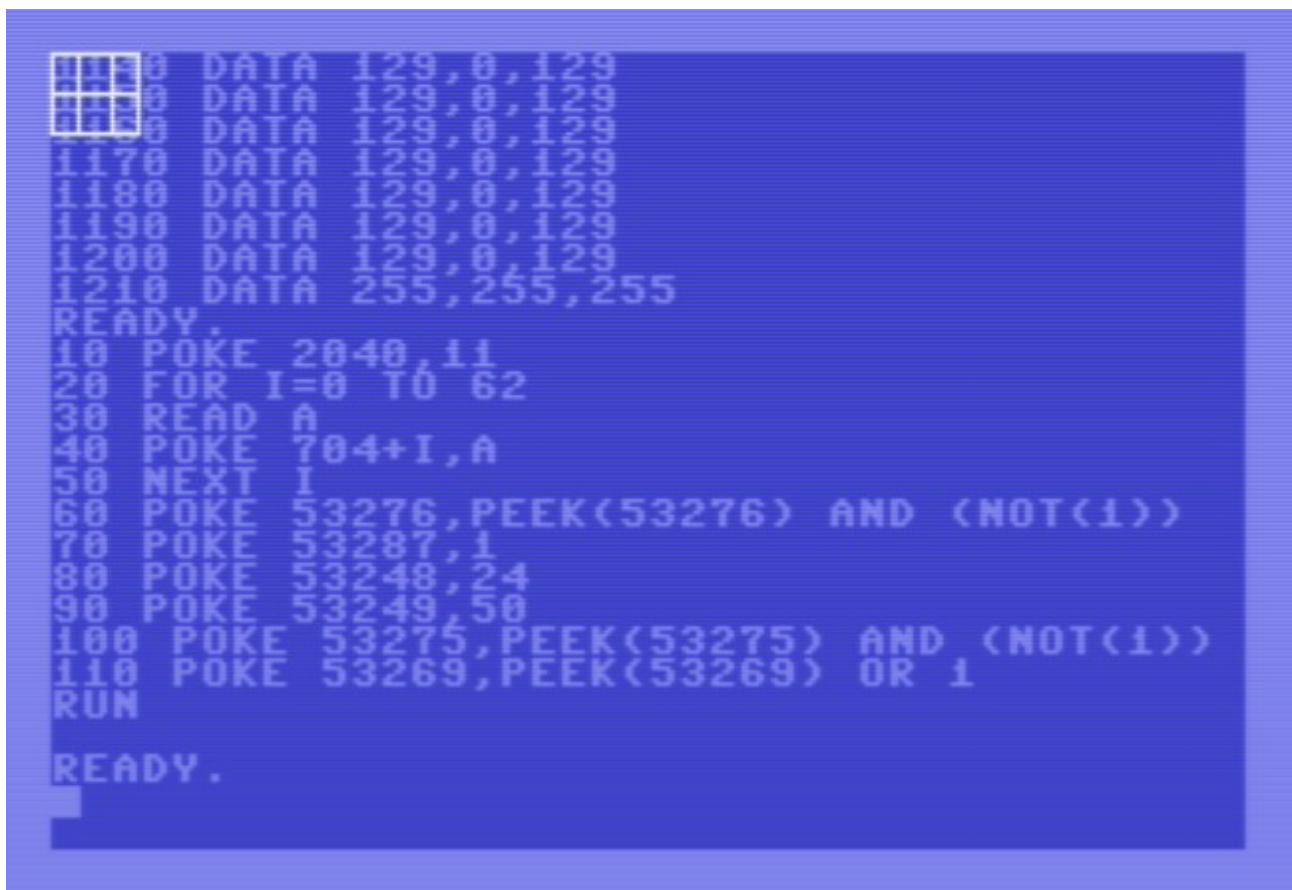
Wichtig:

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
90 READY.
100 POKE 2040,11
110 FOR I=0 TO 62
120 READ A
130 POKE 704+I,A
140 NEXT I
150 POKE 53276,PEEK(53276) AND (NOT(1))
160 POKE 53287,1
170 POKE 53248,24
180 POKE 53249,50
190 POKE 53275,PEEK(53275) AND (NOT(1))
200 POKE 53269,PEEK(53269) OR 1
210 RUN
220 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

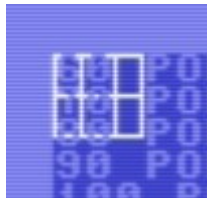
```
POKE 53269,PEEK(53269) OR 1
```

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.

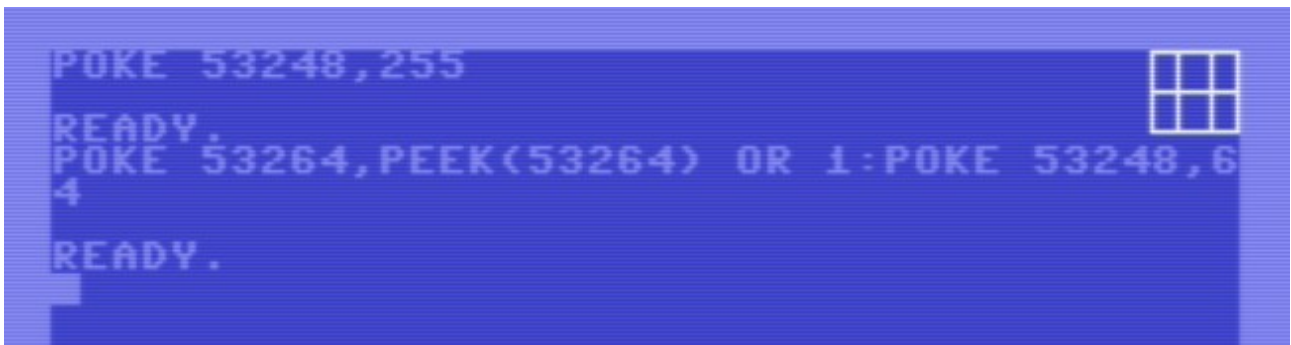


Wollen wir das Sprite an den rechten Rand des sichtbaren Bereichs positionieren, also auf die Position 320, dann müssen wir die Speicherstelle 53264 zu Hilfe nehmen.

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.

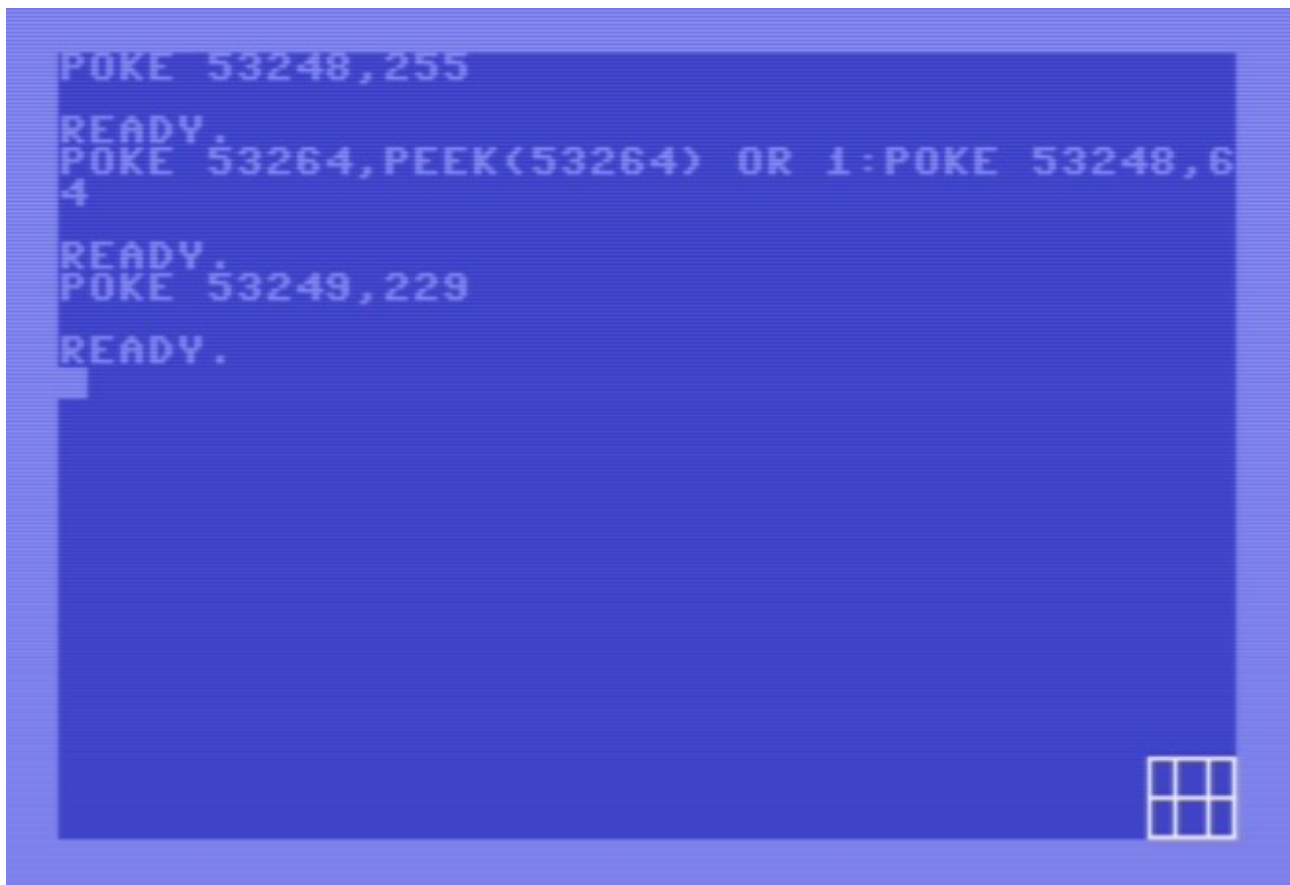
```
POKE 53264,PEEK(53264) OR 1:POKE 53248,64
```



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:



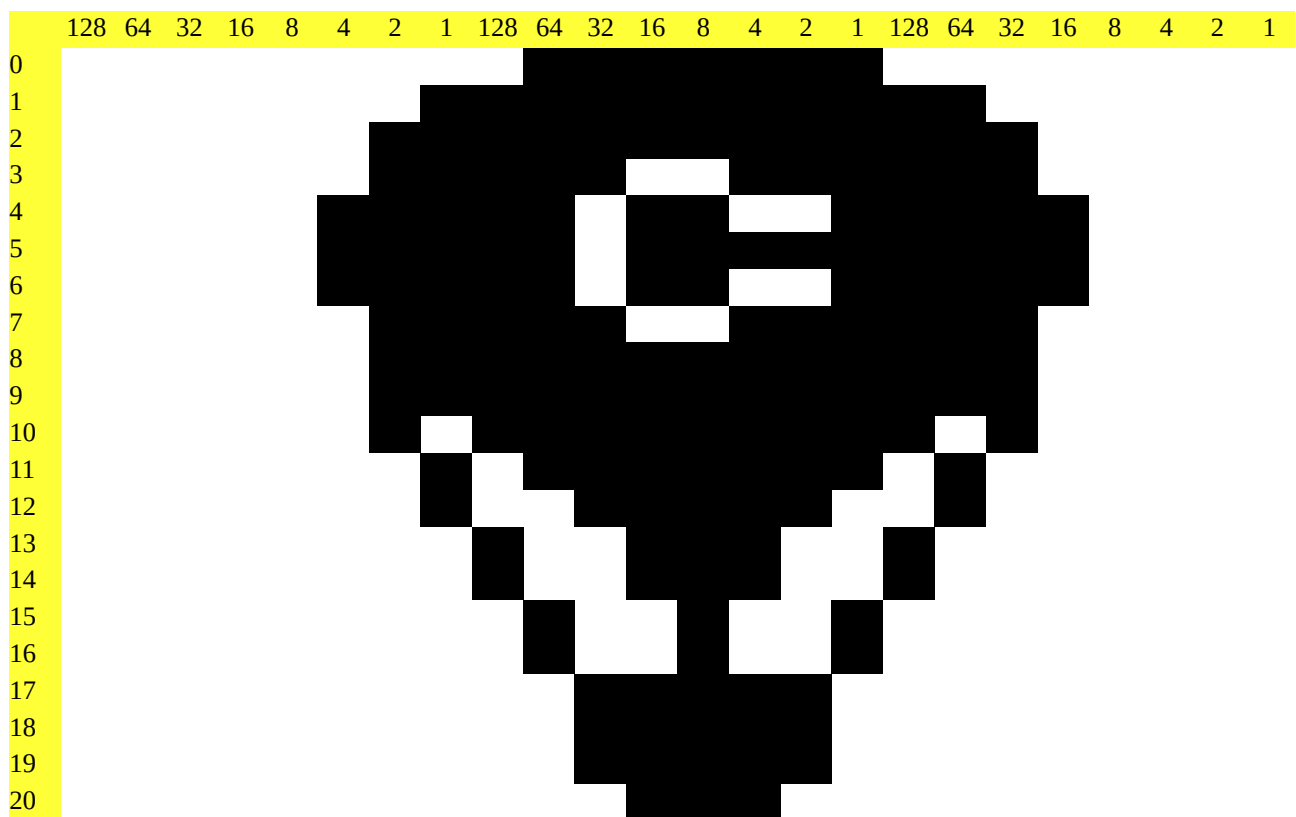
Für eine vertikale Vergrößerung ist die Speicherstelle 53271 zuständig. Auch hier ist wieder jedes Sprite mit einem eigenen Bit vertreten. Setzt man es auf den Wert 1, so wird das Sprite in vertikaler Richtung verdoppelt. Setzt man es umgekehrt auf den Wert 0, so wird das Sprite in Normalgröße angezeigt.

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.



Sprite-Priorität

Ich habe über die entsprechenden POKE-Befehle 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 (Viereck) sowohl von Sprite 0 (Gitter) als auch von Sprite 1 (Ballon) überdeckt wird.

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

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.

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.

Laden Sie dazu den Sourcecode sprite1src in den Editor, sodass wir ihn gemeinsam Schritt für Schritt durchgehen können.

Da ich versucht habe, das Programm gut zu dokumentieren, sollte der Sourcecode fast selbsterklärend sein.

Folgende Schritte werden im Programm durchlaufen:

Vergeben der Blocknummern für die drei Sprites

Speicherstelle \$07F8 (dezimal 2040) => Blocknummer \$0B (dezimal 11)

Speicherstelle \$07F9 (dezimal 2041) => Blocknummer \$0D (dezimal 13)

Speicherstelle \$07FA (dezimal 2042) => Blocknummer \$0E (dezimal 14)

```
        ; blocknr. fuer gitter
        lda #$0b
        sta $07f8

        ; blocknr. fuer ballon
        lda #$0d
        sta $07f9

        ; blocknr. fuer viereck
        lda #$0e
        sta $07fa
```

Umkopieren der Spritedaten

Dies können wir für alle drei Sprites in einer Schleife erledigen.

```
copyloop    ldx #$00
            lda spritegitter,x
            sta $02c0,x

            lda spriteballon,x
            sta $0340,x

            lda spriteviereck,x
            sta $0380,x

            inx
            cpx #$3f
            bne copyloop
```

Der Inhalt des X Registers wird wie gewohnt innerhalb der Schleife von 0 beginnend hochgezählt, bis es den Wert \$3F (dezimal 63) enthält. Solange dies nicht der Fall ist, wird immer wieder zum Label copyloop gesprungen und die Schleife erneut durchlaufen.

Auf diese Weise wird Byte für Byte in den jeweiligen Block kopiert.

Sprites einschalten

```
; sprite 0,1,2 einschalten  
lda $d015  
ora #$07  
sta $d015
```

Dazu müssen wir in der Speicherstelle \$D015 (dezimal 53269) die Bits 0, 1 und 2 setzen. Als Erstes lesen wir den Inhalt der Speicherstelle \$D015 aus, führen mit diesem eine ODER-Verknüpfung mit dem Wert 7 (binär %00000111) durch, wodurch die Bits 0, 1 und 2 gesetzt werden. Abschließend schreiben wir den Wert wieder in die Speicherstelle \$D015 zurück.

Sprites als einfarbig definieren

```
; alle sprites einfarbig  
lda $d01c  
and $f8  
sta $d01c
```

Dazu müssen wir in der Speicherstelle \$D01C (dezimal 53276) die Bits 0, 1 und 2 zurücksetzen. Als Erstes lesen wir den Inhalt der Speicherstelle \$D01C aus, führen mit diesem eine UND-Verknüpfung mit dem Wert \$F8 (binär %11111000) durch, wodurch die Bits 0, 1 und 2 zurückgesetzt werden. Abschließend schreiben wir den Wert wieder in die Speicherstelle \$D01C zurück.

Farben für die Sprites vergeben

```
; farbe fuer gitter  
lda #$01  
sta $d027  
  
; farbe fuer ballon  
lda #$07  
sta $d028  
  
; farbe fuer viereck  
lda #$03  
sta $d029
```

Das Gitter (Sprite 0) erhält die Farbe Weiß, d.h. wir müssen den Wert 1 in die Speicherstelle \$D027 (dezimal 53287) schreiben.

Der Ballon (Sprite 1) soll in gelber Farbe angezeigt werden, was durch den Wert 7 in Speicherstelle \$D028 (dezimal 53288) erreicht wird.

Das Viereck (Sprite 2) wollen wir in Türkis anzeigen und schreiben dazu den Wert 3 in die Speicherstelle \$D029 (dezimal 53289).

Priorität der Sprites gegenüber dem Hintergrund einstellen



Alle unsere Sprites sollen Priorität gegenüber dem Hintergrund haben, d.h. wir müssen die Bits 0, 1 und 2 in der Speicherstelle \$D01B (dezimal 53275) zurücksetzen. Als Erstes lesen wir den Inhalt der Speicherstelle \$D01B aus, führen mit diesem eine UND-Verknüpfung mit dem Wert \$F8 (binär %11111000) durch, wodurch die Bits 0, 1 und 2 zurückgesetzt werden. Abschließend schreiben wir den Wert wieder in die Speicherstelle \$D01B zurück.

Positionieren der Sprites

Wie wir bereits wissen, sind in den Speicherstellen \$D000, \$D002, \$D004, \$D006, \$D008, \$D00A, \$D00C und \$D00E die x-Koordinaten der Sprites gespeichert.

In diesen Speicherstellen können wir jedoch nur Werte zwischen 0 und 255 speichern. Wie man x-Koordinaten größer als 255 einstellt und welche Rolle die Speicherstelle \$D010 (dezimal 53264) dabei spielt, habe ich bereits bei der BASIC-Umsetzung erklärt. Trotzdem möchte ich die Zusammenhänge noch einmal wiederholen, um sie wieder in Erinnerung zu rufen.

Um auch die x-Koordinaten jenseits der Grenze von 255 nutzen zu können, gibt es die Speicherstelle \$D010. Sie stellt für jedes der 8 Sprites ein zusätzliches Bit für die Festlegung der x-Koordinate zur Verfügung, wodurch die vorhin genannten Speicherstellen quasi um ein zusätzliches Bit erweitert werden. Dieses zusätzliche Bit reicht aus, um auch alle möglichen x-Koordinaten von 256 bis 511 darstellen zu können.

Angenommen, wir wollen für das Sprite 0 eine x-Koordinate von 320 einstellen. Mit der Speicherstelle \$D000 allein kommen wir hier nicht aus, da wir dort ja nur Werte zwischen 0 und 255 speichern können. Wir müssen also das Bit 0 in der Speicherstelle \$D010 zu Hilfe nehmen, um die x-Koordinate 320 einstellen zu können.

Bit 0 aus \$D010	Speicherstelle \$D000							
8	7	6	5	4	3	2	1	0
256	128	64	32	16	8	4	2	1
1	0	1	0	0	0	0	0	0

Das zusätzliche Bit hat die Wertigkeit 256, d.h. wir müssen in der Speicherstelle \$D000 nur mehr den Wert eintragen, der uns noch auf den Wert 320 fehlt.

$320 - 256 = 64$, d.h. um die x-Koordinate für das Sprite 0 auf den Wert 320 zu setzen, müssen wir Bit 0 in der Speicherstelle \$D010 setzen und den Wert 64 in die Speicherstelle \$D000 schreiben.

Dieses Schema gilt analog auch für die anderen Sprites.

Kommen wir nun zu den drei Unterprogrammen, welche an der Positionierung der Sprites beteiligt sind.

Unterprogramm setbit8forx

```
setbit8forx
    tay
    lda zweierpotenzen,y
    bcc clearbit
    ora $d010
    jmp setd010
clearbit
    eor #$ff
    and $d010
setd010
    sta $d010
    rts
```

Das Unterprogramm übernimmt zwei Parameter. Im Y Register wird die Nummer des Sprites übergeben dessen Bit man in der Speicherstelle \$d010 ändern will. Will man also das Bit für das Sprite 0 ändern, dann muss man im Y Register den Wert 0 übergeben. Für das Sprite 7 wäre es der Wert 7.

Der zweite Parameter kommt über das Carry Flag ins Unterprogramm. Setzt man es vor dem Aufruf des Unterprogramms, dann bedeutet das, dass man das jeweilige Bit setzen will. Ist es hingegen zurückgesetzt, dann signalisiert man dadurch, dass man das jeweilige Bit zurücksetzen will.

Im Datenbereich ist ein Bereich namens zweierpotenzen definiert. Hier stehen nacheinander die Wertigkeiten der Bitpositionen in einem Byte, also die Werte 1, 2, 4, 8, 16, 32, 64, 128 (im Assemblercode habe ich sie jedoch hexadezimal angegeben: \$01, \$02, \$04, \$08, \$10, \$20, \$40, \$80)

Durch den ersten Befehl TAY wird die Nummer des Sprites in das Y Register kopiert, damit wir über die indizierte Adressierung auf den Bereich zweierpotenzen zugreifen können.

Wozu brauchen wir diese Werte? Wir haben als Parameter eine Spritenummer übergeben. Diese Nummer entspricht 1:1 der Position des Bits in der Speicherstelle \$D010, welches für das jeweilige Sprite zuständig ist.

Bitposition 0 ist für das Sprite 0 zuständig, Bitposition 1 ist für das Sprite 1 zuständig usw. Durch die Spritenummer kennen wir also gleichzeitig die Position des Bits in der Speicherstelle \$D010, welches wir ändern müssen.

Durch den Befehl lda zweierpotenzen,y wird die Wertigkeit an dieser Bitposition in den Akkumulator geladen. Im Falle von Sprite 0 also der Wert 1, im Falle von Sprite 1 der Wert 2 usw. Diesen Wert brauchen wir für die anschließende logische Verknüpfung.

Falls das Carry Flag nicht gesetzt ist, wird durch den Befehl bcc (branch on carry clear) zum Label clearbit verzweigt. Dort wird durch den Befehl eor #\$ff zunächst das Einerkomplement des Akkumulator-Inhalts gebildet und das Ergebnis anschließend über den Befehl and \$D010 eine UND-Verknüpfung mit dem aktuellen Inhalt der Speicherstelle \$D010 durchgeführt. Dies bewirkt ein Zurücksetzen des jeweiligen Bits im Akkumulator-Inhalt.

Im Anschluss wird das Ergebnis durch den Befehl sta \$D010 in die Speicherstelle \$D010 zurückgeschrieben, damit die Änderung wirksam wird. Durch den Befehl rts erfolgt dann der Rücksprung aus dem Unterprogramm.

Ist das Carry Flag hingegen gesetzt, wird über den Befehl ora \$D010 eine ODER-Verknüpfung des Akkumulator-Inhalts mit dem aktuellen Inhalt der Speicherstelle \$D010 durchgeführt, wodurch das jeweilige Bit im Akkumulator-Inhalt gesetzt wird.

Dann wird zum Label setd010 gesprungen, wodurch analog zum anderen Fall das Ergebnis in die Speicherstelle \$D010 zurückgeschrieben wird und über den Befehl rts der Rücksprung aus dem Unterprogramm erfolgt.

Unterprogramm setspritex

```
setspritex
    pha
    asl a
    tay
    lda $fa
    sta $d000,y
    lda $fb
    beq xk1g255
    pla
    sec
    jsr setbit8forx
    jmp setxende
xk1g255
    pla
    clc
    jsr setbit8forx
setxende
    rts
```

Dieses Unterprogramm ist für die Einstellung der x-Koordinate eines Sprites zuständig und nutzt dafür das soeben beschriebene Unterprogramm setbit8forx.

Als Parameter wird im Akkumulator wieder die Nummer des Sprites übergeben, dessen x-Koordinate man einstellen will. Die x-Koordinate selbst setzt sich aus dem Inhalt der Speicherstellen \$fa (niederwertiges Byte) und \$fb (höherwertiges Byte) zusammen.

Vor dem Aufruf des Unterprogramms muss also die Nummer des Sprites im Akkumulator und die gewünschte x-Koordinate aufgeteilt auf das niederwertige und höherwertige Byte in den Speicherstellen \$fa und \$fb stehen.

Mit dem ersten Befehl pha wird die übergebene Spritenummer auf dem Stack gesichert, da sie gleich durch den nächsten Befehl verändert wird.

Dies ist der neue Befehl ASL (Arithmetic Shift Left)

Dieser Befehl bewirkt, dass alle Bits im Akkumulator um eine Position nach links geschoben werden, wobei das Bit 7, welches an der linken Stelle dadurch herausfällt, in das Carry Flag wandert. Auf der rechten Seite kommt ein 0-Bit herein.

Hier ein Beispiel:

Angenommen der Akkumulator enthält den binären Wert %10110010:

Vor ASL	1	0	1	1	0	0	1	0
Nach ASL	0	1	1	0	0	1	0	0

Das Bit 7 mit dem Wert 1 ist herausgefallen und wird in das Carryflag übertragen, dieses wird also gesetzt.

Auf der rechten Seite kam ein 0-Bit herein.

Mathematisch gesehen bewirkt eine Verschiebung um eine Position nach links einer Multiplikation mit zwei.

Umgekehrt bewirkt eine Verschiebung um eine Position nach rechts einer Division durch zwei. Hierzu gibt es den Befehl LSR (Logical Shift Right). Umgekehrt wandert hier auf der linken Seite ein 0-Bit herein und das rechte herausfallende Bit wird durch das Carryflag aufgefangen.

Vor LSR	1	0	1	1	0	0	1	0
Nach LSR	0	1	0	1	1	0	0	1

Das Bit 0 mit dem Wert 0 ist herausgefallen und wird in das Carryflag übertragen, wodurch dieses zurückgesetzt wird.

Auf der linken Seite kam ein 0-Bit herein. Wir brauchen die Spritenummer später jedoch noch, daher müssen wir sie vor der Multiplikation auf dem Stack sichern.

Durch den Befehl tay wird das Ergebnis der Multiplikation in das Y Register kopiert, damit wir über die indizierte Adressierung auf die Speicherstellen zugreifen können, welche für die x-Koordinate der Sprites zuständig sind.

Diese sind jeweils um zwei Stellen verschoben, weswegen es zuvor nötig war, die Spritenummer mit zwei zu multiplizieren.

Hier eine Tabelle, welche veranschaulicht, wie die Adresse der Speicherstellen für die jeweilige x-Koordinate durch Angabe der Spritenummer gebildet wird.

Spritenummer	Spritenummer * 2	Adresse
0	0	\$d000
1	2	\$d002
2	4	\$d004
3	6	\$d006
4	8	\$d008
5	10	\$d00a
6	12	\$d00c
7	14	\$d00e

Durch den Befehl `sta $d000,y` wird das niederwertige Byte der x-Koordinate in diese Speicherstelle geschrieben. Dieses Byte wurde vorher durch den Befehl `lda $fa` in den Akkumulator geladen.

Kommen wir nun zum höherwertigen Byte der gewünschten x-Koordinate. Diese findet das Unterprogramm in der Speicherstelle `$fb` vor und deshalb laden wir es mit dem Befehl `lda $fb` in den Akkumulator.

Falls das höherwertige Byte den Wert 0 enthält, es sich also um eine x-Koordinate kleiner oder gleich 255 handelt, dann wird zum Label `xklg255` verzweigt. Dort wird die zuvor auf dem Stack gesicherte Spritenummer wieder vom Stack in den Akkumulator geholt, denn diese brauchen wir nun für den Aufruf des Unterprogramms `setbit8forx`.

Da es sich um eine x-Koordinate kleiner oder gleich 255 handelt, wird mit dem Befehl `clc` das Carry Flag zurückgesetzt und das Unterprogramm `setbit8forx` aufgerufen. Dadurch wird das jeweilige Bit in der Speicherstelle `$d010` zurückgesetzt. Durch den Befehl `rts` erfolgt dann der Rücksprung aus dem Unterprogramm.

Enthält das höherwertige Byte jedoch einen Wert ungleich 0, handelt es sich also um eine x-Koordinate, die größer als 255 ist, dann wird ebenfalls zunächst die zuvor auf dem Stack gesicherte Spritenummer in den Akkumulator geholt, da wir sie für den Aufruf des Unterprogramms `setbit8forx` brauchen. Da es sich um eine x-Koordinate größer als 255 handelt, muss das dem Sprite zugehörige Bit in der Speicherstelle `$d010` gesetzt werden.

Daher wird vor dem Aufruf des Unterprogramms `setbit8forx` das Carry Flag gesetzt, wodurch das soeben erwähnte Bit gesetzt wird. Nach der Rückkehr aus dem Unterprogramm wird zum Label `setxende` verzweigt. Dort erfolgt dann mittels des Befehls `rts` der Rücksprung aus dem Unterprogramm.

Nun ist abhängig von der gewünschten x-Koordinate das niederwertige Byte in der korrekten Speicherstelle eingetragen und das jeweilige Bit in der Speicherstelle `$d010` entweder gesetzt oder nicht.

Unterprogramm setspritey

```
setspritey
    asl a
    tay
    lda $fa
    sta $d001,y
    rts
```

Dieses Unterprogramm ist für die Einstellung der y-Koordinate eines Sprites zuständig. Hier haben wir es leichter als bei der x-Koordinate, weil hier keine Werte über 255 möglich sind.

Auch hier wird im Akkumulator die Spritenummer übergeben und die gewünschte y-Koordinate muss sich vor dem Aufruf des Unterprogramms in der Speicherstelle \$fa befinden. Da die Speicherstellen für die y-Koordinaten ebenfalls jeweils um zwei Stellen versetzt sind, wird die Spritenummer wiederum durch den Befehl asl mit zwei multipliziert und das Ergebnis in das Y Register kopiert, damit wir über die indizierte Adressierung auf die Speicherstellen zugreifen können, welche für die y-Koordinate der Sprites zuständig sind.

Hier wieder eine Tabelle, welche veranschaulicht, wie die Adresse der Speicherstellen für die jeweilige y-Koordinate durch Angabe der Spritenummer gebildet wird.

Spritenummer	Spritenummer * 2	Adresse
0	0	\$d001
1	2	\$d003
2	4	\$d005
3	6	\$d007
4	8	\$d009
5	10	\$d00b
6	12	\$d00d
7	14	\$d00f

Durch den Befehl sta \$fa wird dann die gewünschte y-Koordinate in diese Speicherstelle geschrieben. Zuvor haben wir die y-Koordinate mit dem Befehl lda \$fa in den Akkumulator geladen. Und das war's auch schon, sodass mit dem Befehl rts der Rücksprung aus dem Unterprogramm erfolgen kann.

Möglicherweise haben Sie sich gefragt, warum ich die Einstellung der x- und y-Koordinate auf separate Unterprogramme aufgeteilt habe. Ursprünglich hatte ich beide Einstellungen in einem Unterprogramm, aber es hat sich später herausgestellt, dass es besser ist, die beiden Einstellungen auf zwei separate Unterprogramme aufzuteilen.

Angenommen, man will ein Sprite horizontal über den Bildschirm bewegen. Dann verändert sich nur die x-Koordinate, aber die y-Koordinate bleibt konstant, sodass man sie auch nicht bei jedem Schritt immer wieder neu einstellen muss.

Dasselbe gilt für die vertikale Bewegung. Hier ändert sich nur die y-Koordinate und die x-Koordinate bleibt konstant. Gerade bei einer vertikalen Bewegung spart man hier einiges an

Rechenzeit, weil das Einstellen der x-Koordinate ja doch ein wenig aufwändiger ist, als das Einstellen der y-Koordinate wie wir gesehen haben.

Durch die Aufteilung auf zwei Unterprogramme muss man vor dem Start der vertikalen Bewegung die gewünschte x-Koordinate nur ein einziges mal einstellen und nicht bei jedem Schritt in vertikaler Richtung.

Gleiches gilt natürlich auch für die horizontale Bewegung, hier stellt man die gewünschte y-Koordinate vor dem Start der horizontalen Bewegung ein einziges mal ein und erspart sich die Einstellung bei jedem Schritt in horizontaler Richtung.

Möchte man sowohl die x- als auch die y-Koordinate ändern, dann ruft man die beiden Unterprogramme setspritex und setspritey einfach hintereinander mit den gewünschten Werten auf.

Auf diese Art und Weise führt man die Schritte wirklich nur dann aus, wenn sie wirklich nötig sind. Kommen wir nun zu dem Programmteil, welcher die drei Sprites am Bildschirm positioniert und sich dabei der Unterprogramme bedient, die wir soeben besprochen haben.

Ich werde hier nur die Positionierung des Gitters erläutern, der Ballon und das Viereck werden auf dieselbe Art und Weise positioniert.



```
; gitter positionieren
; x = 320, y = 50

lda #$40 ; lo(x)
sta $fa ; in $fa
lda #$01 ; hi(x)
sta $fb ; in $fb
lda #$00 ; spritenummer 0
jsr setspritex
lda #$32 ; y
sta $fa ; in $fa
lda #$00 ; spritenummer 0
jsr setspritey
```

Die x-Koordinate des Gitters soll auf den Wert 320 eingestellt werden. Dies entspricht dem hexadezimalen Wert \$0140. Das niederwertige Byte \$40 schreiben wir in die Speicherstelle \$fa und das höherwertige Byte \$01 schreiben wir in die Speicherstelle \$fb.

Abschließend schreiben wir noch die Spritenummer 0 in den Akkumulator und rufen dann das Unterprogramm setspritex auf.

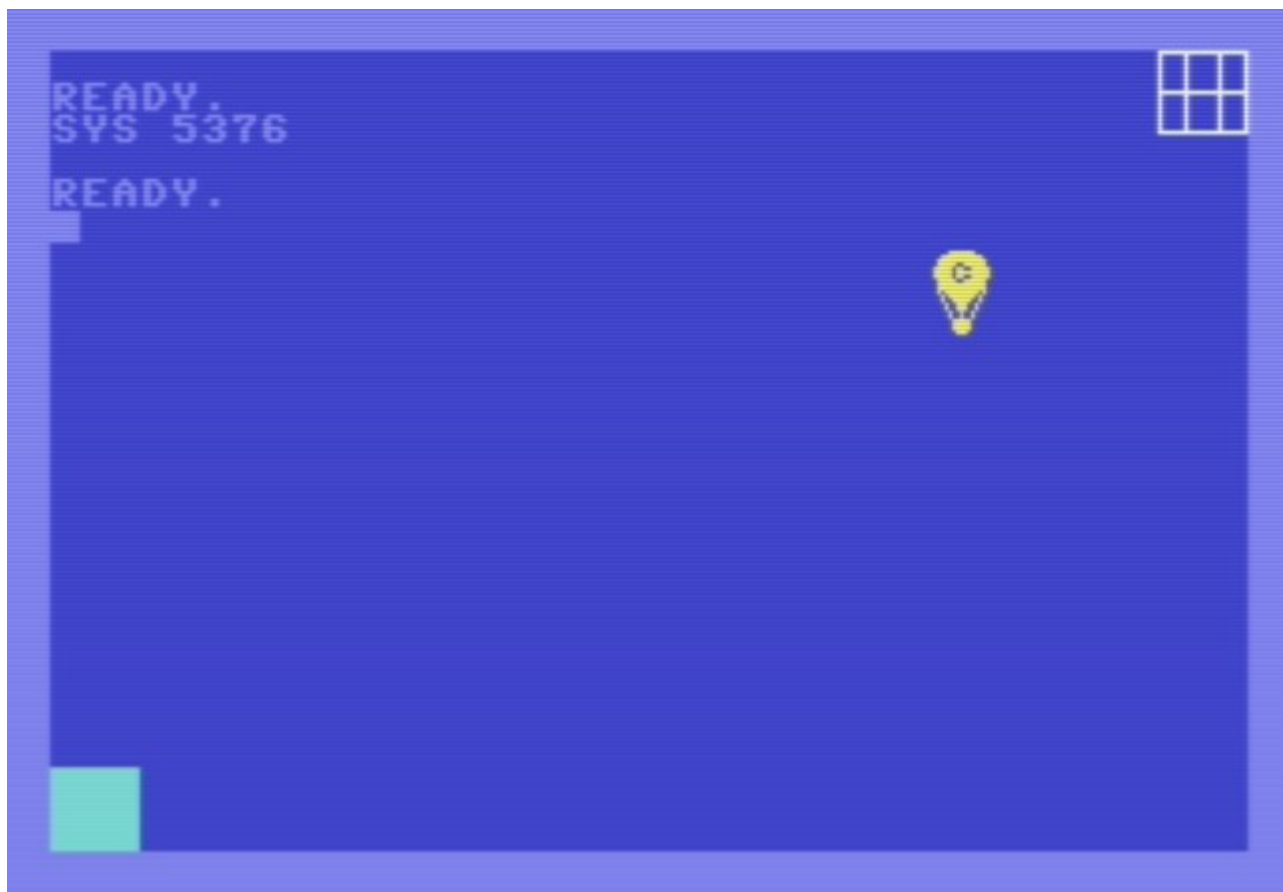
Dann schreiben wir die y-Koordinate in die Speicherstelle \$fa, laden den Akkumulator mit der Spritenummer 0 und rufen das Unterprogramm setspritey auf.

Somit haben wir das Gitter auf die Koordinaten x=320, y=50 positioniert.

Im Assemblercode folgt nun nur noch die Positionierung des Ballons und des Vierecks, welche wie bereits erwähnt vollkommen gleich funktioniert wie die des Gitters.

Den Abschluss des Programms bildet der Befehl RTS.

Wenn Sie das Programm mit SYS 5376 starten, sollten Sie folgendes Ergebnis erhalten:



Mehrfarbige Sprites

Bei den mehrfarbigen Sprites reduziert sich die horizontale Auflösung auf die Hälfte, also auf 12 Pixel, da jeweils zwei Bits für die Farbeinstellung eines Pixels gebraucht werden. Hierbei wird in den beiden Bits jedoch nicht direkt eine Farbe angegeben (dafür wären 4 Bits nötig wenn man alle 16 Farben abbilden will), sondern die beiden Bits bilden eine Bitkombination, welche folgende mögliche Werte darstellen kann:

Bitkombination	Farbinformation
00	Hintergrund (transparent)
10	Spritefarbe (Register 53287 - 53294)
01	Mehrfarbenregister #0 (Register 53285)
11	Mehrfarbenregister #1 (Register 53286)

Hier ein Beispiellaster für ein mehrfarbiges Sprite:

	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	00	00	00	00	10	10	10	10	00	00	00	00
1	00	00	00	10	10	10	10	10	10	00	00	00
2	00	00	10	10	10	10	10	10	10	10	00	00
3	00	00	10	10	10	10	10	10	10	10	00	00
4	00	10	10	10	10	10	10	10	10	10	10	00
5	00	10	10	11	10	10	10	10	11	10	10	00
6	00	10	11	11	11	10	10	11	11	11	10	00
7	10	10	11	11	11	10	10	11	11	11	10	10
8	10	10	11	01	11	10	10	11	01	11	10	10
9	10	10	11	01	11	10	10	11	01	11	10	10
10	10	10	10	11	10	10	10	10	11	10	10	10
11	10	10	10	10	10	10	10	10	10	10	10	10
12	10	10	10	10	10	10	10	10	10	10	10	10
13	10	10	10	10	10	10	10	10	10	10	10	10
14	10	10	10	10	10	10	10	10	10	10	10	10
15	10	10	10	10	10	10	10	10	10	10	10	10
16	10	10	10	10	10	10	10	10	10	10	10	10
17	10	10	00	10	10	00	10	10	10	00	10	10
18	10	10	00	10	10	00	00	10	10	00	10	10
19	10	00	00	00	10	00	00	10	00	00	00	10
20	10	00	00	00	10	00	00	10	00	00	00	10

Enthält eine Zelle die Bitkombination 00, dann enthält das Sprite an dieser Stelle keinen Pixel, d.h. hier scheint der Hintergrund durch (in dem Beispiel sind dies die schwarzen Zellen).

Enthält eine Zelle die Bitkombination 10, dann enthält der Pixel an dieser Stelle jene Farbe, welche in dem Farbregister, welches dem Sprite zugeordnet ist, hinterlegt ist. Dies sind dieselben Register wie bei den einfärbigen Sprites, d.h. Register 53287 enthält die Farbinformation für Sprite 0,

Register 53288 enthält die Farbinformation für Sprite 1 usw. bis hin zum Register 53294, welches die Farbinformation für Sprite 7 enthält. In dem Beispiel sind das die grünen Zellen.

Enthält eine Zelle die Bitkombination 01, dann enthält das Sprite an dieser Stelle jene Farbe, welche in dem Mehrfarbenregister #0 (53285) hinterlegt ist. In dem Beispiel sind das die blauen Zellen.

Enthält eine Zelle die Bitkombination 11, dann enthält das Sprite an dieser Stelle jene Farbe, welche in dem Mehrfarbenregister #1 (53286) hinterlegt ist. In dem Beispiel sind das die weißen Zellen.

Die Farben, die in diesen beiden Registern hinterlegt sind, gelten für alle Sprites.

Das heißt, wenn zwei Sprites an derselben Position die Bitkombination 01 oder 11 enthalten, dann haben sie an dieser Stelle auch dieselbe Farbe. Je nach Bitkombination entweder jene aus dem Register 53285 (01) oder jene aus dem Register 53286 (11)

Die Pixel haben im Vergleich zu einfarbigen Sprites jedoch die doppelte Breite, d.h. die sichtbare Breite des Sprites ändert sich trotzdem nicht (24 einfach breite Pixel sind gleich breit wie 12 doppelt breite Pixel)

In horizontaler Richtung haben wir nun nur mehr 12 Zellen, in der vertikalen Richtung hat sich nichts verändert, d.h. es sind hier nach wie vor 21 Zeilen.

Jede Zelle repräsentiert nun jedoch 2 Bits, die jeweils eine der oben genannten Kombinationen annehmen können.

Bei den einfarbigen Sprites entsprach jede Zelle einem Bit (Pixel oder kein Pixel)

Das heißt also, dass wir insgesamt trotzdem wieder auf 24 Bits, also 3 Bytes kommen, weil ja jede der 12 Zellen 2 Bits beinhaltet.

Eine Zeile ist also nach wie vor durch drei Bytes definiert, nur der Inhalt der Bytes wird bei mehrfarbigen Sprites anders interpretiert.

Für die obige Figur lauten die Byte-Werte für die Zeilen:

Erstes Byte	Zweites Byte	Drittes Byte
0	170	0
2	170	128
10	170	160
10	170	160
42	170	168
43	170	232
47	235	248
175	235	250
173	235	122
173	235	122
171	170	234
170	170	170
170	170	170
170	170	170
170	170	170
170	170	170
170	170	170
162	138	138
162	130	138
128	130	2
128	130	2

Gut, dann erstellen wir doch gleich mal ein Programm mit diesem mehrfarbigen Sprite.
Zuerst erfassen wir die Daten aus der Tabelle in DATA-Zeilen:


```

1000 REM DATEN FUER SPRITE
1010 DATA 0,170,0
1020 DATA 2,170,128
1030 DATA 10,170,160
1040 DATA 10,170,160
1050 DATA 42,170,168
1060 DATA 43,170,232
1070 DATA 47,235,248
1080 DATA 175,235,250
1090 DATA 173,235,122
1100 DATA 173,235,122
1110 DATA 171,170,234
1120 DATA 170,170,170
1130 DATA 170,170,170
1140 DATA 170,170,170
1150 DATA 170,170,170
1160 DATA 170,170,170
1170 DATA 170,170,170
1180 DATA 162,138,138
1190 DATA 162,130,138
1200 DATA 128,130,2
1210 DATA 128,130,2
READY.

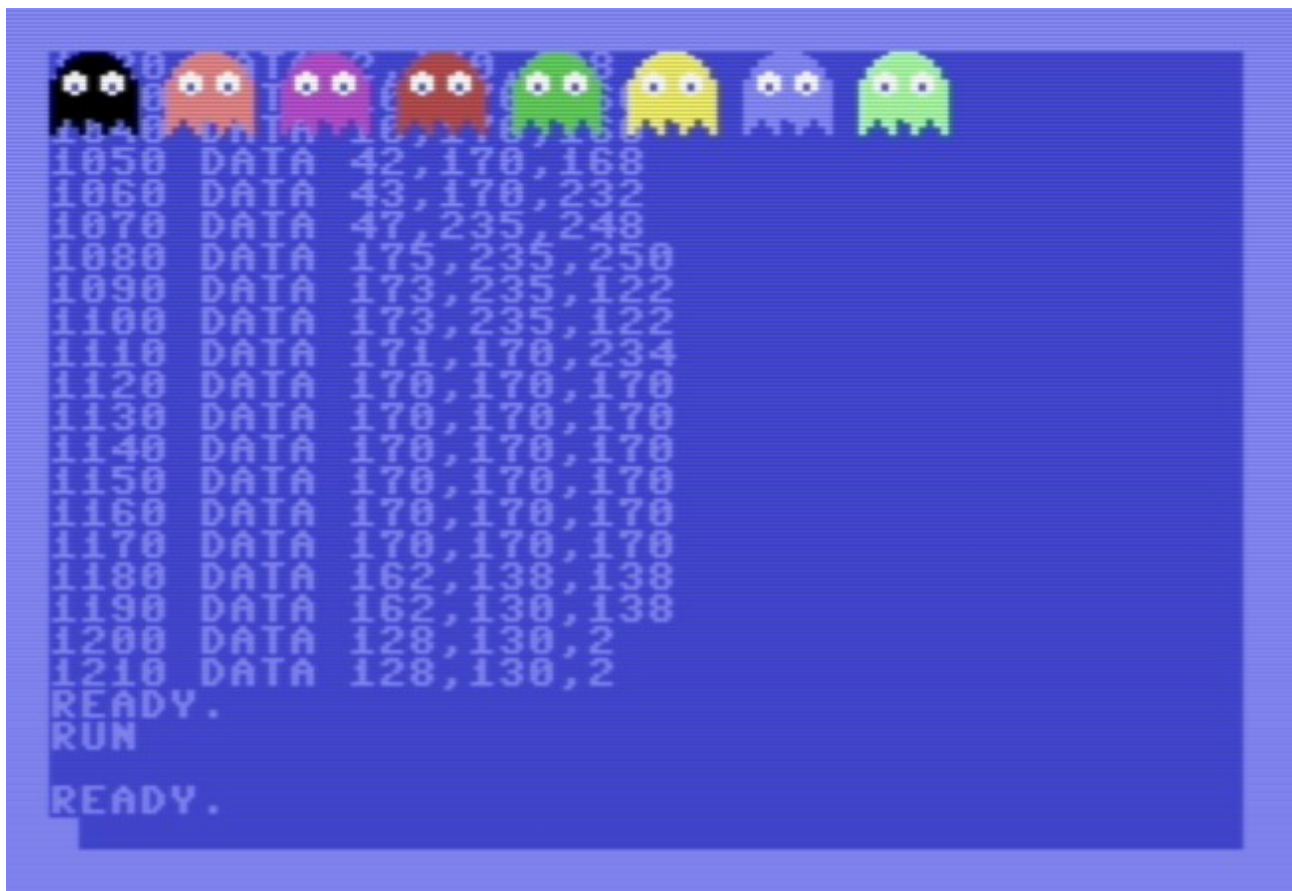
```

Was soll unser Programm machen?

Es soll 8 Sprites, welche sich nur in der Farbgebung unterscheiden, am oberen Rand des Bildschirms anzeigen. Da sich die Sprites nur farblich unterscheiden, ist es nicht nötig, für jedes Sprite eigene Spritedaten zu definieren, sondern wir können für alle Sprites denselben Datenblock nutzen (hier Block Nummer 11) und das individuelle Aussehen der Sprites über die Farbgebung steuern.

Dann sollen sich diese 8 Sprites vom oberen zum unteren Rand des Bildschirms bewegen und danach wieder zurück in die Ausgangsposition am oberen Rand.

Hier die 8 Geister am Ausgangspunkt bzw. nachdem sie wieder an den oberen Bildschirmrand zurückgekehrt sind:



Gehen wir das BASIC-Programm nun Schritt für Schritt durch.

```
LIST 10-170
10 REM BLOCKNUMMER EINTRAGEN
20 FOR I=2040 TO 2047
30 POKE I,11
40 NEXT I
50 REM SPRITEDATEN IN BLOCK 11 KOPIEREN
60 FOR I=0 TO 62
70 READ A
80 POKE 704+I,A
90 NEXT I
100 REM ALLE SPRITES SIND MEHRFARBIG
110 POKE 53276,255
120 REM ALLE SPRITES VOR HINTERGRUND
130 POKE 53275,0
140 REM ERSTE GEMEINSAME FARBE
150 POKE 53285,6
160 REM ZWEITE GEMEINSAME FARBE
170 POKE 53286,1
READY.
```

Zeile 10 – 40

Hier wird für jedes Sprite dieselbe Blocknummer (11) eingetragen, da ja jedes Sprite gleich aussieht und die Unterschiede nur in der Farbgebung bestehen.

Zeile 50 – 90

Hier werden die Spritedaten aus den DATA-Zeilen in den Speicherblock 11 kopiert.

Zeile 100 – 110

Da alle Sprites mehrfarbig sind, können wir in Speicherstelle 53276 alle 8 Bits auf den Wert 1 setzen, also den Wert 255 dort eintragen.

Zeile 120 – 130

Alle Sprites sollen sich vor dem Hintergrund befinden, also höhere Priorität als der Hintergrund haben. Daher setzen wir alle 8 Bits auf den Wert 0 und tragen den Wert 0 in die Speicherstelle 53275 ein.

Zeile 140 – 150

Hier wird die erste Farbe, welchen allen Sprites gemeinsam ist, eingestellt. In unserem Beispiel hier ist das die Farbe Blau (Farbcode 6). Dies ist die Augenfarbe der Geister.

Zeile 160 – 170

Hier wird die zweite Farbe, welchen allen Sprites gemeinsam ist, eingestellt. In unserem Beispiel hier ist das die Farbe Weiß (Farbcode 1). Dies ist „das Weiße“ im Auge der Geister.

Zeile 180 – 330

Hier wird die individuelle Farben für jedes der 8 Sprites eingestellt.

```
LIST 180-330
180 REM FARBE FUER SPRITE 0
190 POKE 53287,0
200 REM FARBE FUER SPRITE 1
210 POKE 53288,10
220 REM FARBE FUER SPRITE 2
230 POKE 53289,4
240 REM FARBE FUER SPRITE 3
250 POKE 53290,2
260 REM FARBE FUER SPRITE 4
270 POKE 53291,5
280 REM FARBE FUER SPRITE 5
290 POKE 53292,7
300 REM FARBE FUER SPRITE 6
310 POKE 53293,14
320 REM FARBE FUER SPRITE 7
330 POKE 53294,13

READY.
```

Zeile 340 – 490

Hier werden sowohl die X- als auch die Y – Koordinate für jedes der 8 Sprites eingestellt. Zu Beginn befinden sich alle 8 Sprites am oberen Bildschirmrand, daher ist zu Beginn die Y – Koordinate bei allen Sprites gleich.

```
LIST 340-510
```

```
340 REM POSITION VON SPRITE 0
350 POKE 53248,24:POKE 53249,50
360 REM POSITION VON SPRITE 1
370 POKE 53250,55:POKE 53251,50
380 REM POSITION VON SPRITE 2
390 POKE 53252,86:POKE 53253,50
400 REM POSITION VON SPRITE 3
410 POKE 53254,117:POKE 53255,50
420 REM POSITION VON SPRITE 4
430 POKE 53256,148:POKE 53257,50
440 REM POSITION VON SPRITE 5
450 POKE 53258,179:POKE 53259,50
460 REM POSITION VON SPRITE 6
470 POKE 53260,210:POKE 53261,50
480 REM POSITION VON SPRITE 7
490 POKE 53262,241:POKE 53263,50
500 REM ALLE SPRITES AKTIVIEREN
510 POKE 53269,255
```

```
READY.
```

Zeile 500 – 510

Hier werden alle Sprites aktiviert, also auf den Positionen, die wir eingestellt haben, angezeigt. Da wir alle 8 Sprites aktivieren wollen, ist es nicht nötig, jedes einzelne zu aktivieren, sondern wir können alle Sprites in einem Schwung sichtbar machen, in dem wir alle 8 Bits in der Speicherstelle 53269 auf den Wert 1 setzen, also den Wert 255 dorthin schreiben.

Zeile 520 – 790

Hier findet die Bewegung der Sprites nach unten bzw. anschließend wieder nach oben statt.

Die Werte für die Y – Koordinate werden jeweils in einer Schleife durchlaufen und durch das Unterprogramm ab Zeile 700 werden die aktuellen Y - Koordinaten für jedes Sprite aktualisiert.


```

LIST 520-790
520 REM SPRITES NACH UNTEN BEWEGEN
530 FOR Y=51 TO 229
540 GOSUB 700
550 NEXT Y
560 REM SPRITES NACH OBEN BEWEGEN
570 FOR Y=228 TO 50 STEP-1
580 GOSUB 700
590 NEXT Y
600 END
700 REM Y KOORDINATEN SETZEN
710 POKE 53249,Y
720 POKE 53251,Y
730 POKE 53253,Y
740 POKE 53255,Y
750 POKE 53257,Y
760 POKE 53259,Y
770 POKE 53261,Y
780 POKE 53263,Y
790 RETURN
READY.

```

Nun wird es wieder spannend, denn wir wollen die Assembler-Version dieses BASIC-Programms in Angriff nehmen.

Ok, was müssen wir als Erstes machen? Richtig, wir müssen zunächst mal die Daten für unser Sprite im Speicher ablegen und von dort dann in den Speicherblock 11 kopieren.

Hier die Spritedaten im Speicher ab Adresse \$1500:

```

.M 1500 1540
:1500 00 AA 00 02 AA 80 0A AA
:1508 A0 0A AA A0 2A AA A8 2B
:1510 AA E8 2F EB F8 AF EB FA
:1518 AD EB 7A AD EB 7A AB AA
:1520 EA AA AA AA AA AA AA AA
:1528 AA AA AA AA AA AA AA AA
:1530 AA AA AA A2 8A 8A A2 82
:1538 8A 80 82 02 80 82 02 00
.

```

Assembler-Gegenstück für die Zeilen 10 – 40 (Blocknummer 11 für alle Sprites einstellen)

```

.D 1540
,1540 A9 0B LDA #0B
,1542 A2 00 LDX #00
,1544 9D F8 07 STA 07F8,X
,1547 E8 INX
,1548 E0 08 CPX #08
,154A D0 F8 BNE 1544

```

Assembler-Gegenstück für die Zeilen 50 – 90 (Spritedaten in Block 11 kopieren)

```
,154C A2 00 LDX #00
,154E BD 00 15 LDA 1500,X
,1551 9D C0 02 STA 02C0,X
,1554 E8 INX
,1555 E0 3F CPX #3F
,1557 D0 F5 BNE 154E
```

Assembler-Gegenstück für die Zeilen 100 – 110 (Alle 8 Sprites sind mehrfarbig)

```
,1559 A9 FF BNE 154E
,155B 8D 1C D0 LDA #FF
,155D 8D 1C D0 STA D01C
```

Assembler-Gegenstück für die Zeilen 120 – 130 (Alle 8 Sprites vor dem Hintergrund)

```
,155E A9 00 LDA #00
,1560 8D 1B D0 STA D01B
```

Assembler-Gegenstück für die Zeilen 140 – 150 (Erste gemeinsame Farbe einstellen)

```
,1563 A9 06 LDA #06
,1565 8D 25 D0 STA D025
```

Assembler-Gegenstück für die Zeilen 160 – 170 (Zweite gemeinsame Farbe einstellen)

```
,1568 A9 01 LDA #01
,156A 8D 26 D0 STA D026
```

Assembler-Gegenstück für die Zeilen 180 – 330 (Individuelle Farben der Sprites einstellen)

```
,156D A9 00 LDA #00
,156F 8D 27 D0 STA D027
,1572 A9 0A LDA #0A
,1574 8D 28 D0 STA D028
,1577 A9 04 LDA #04
,1579 8D 29 D0 STA D029
,157C A9 02 LDA #02
,157E 8D 2A D0 STA D02A
,1581 A9 05 LDA #05
,1583 8D 2B D0 STA D02B
,1586 A9 07 LDA #07
,1588 8D 2C D0 STA D02C
,158B A9 0E LDA #0E
,158D 8D 2D D0 STA D02D
,1590 A9 0D LDA #0D
,1592 8D 2E D0 STA D02E
```

Assembler-Gegenstück für die Zeilen 340 – 490 (X- und Y – Koordinaten der Sprites einstellen)

```
,1595F 8D 18 D0 LDA #18
,1597 8D 00 D0 STA D000
,1599A 8D 32 D0 LDA #32
,1599C 8D 01 D0 STA D001
,1599F 8D 37 D0 LDA #37
,15A1 8D 02 D0 STA D002
,15A4 8D 32 D0 LDA #32
,15A6 8D 03 D0 STA D003
,15A9 8D 56 D0 LDA #56
,15AB 8D 04 D0 STA D004
,15AE 8D 32 D0 LDA #32
,15B0 8D 05 D0 STA D005
,15B3 8D 75 D0 LDA #75
,15B5 8D 06 D0 STA D006
,15B8 8D 32 D0 LDA #32
,15BA 8D 07 D0 STA D007
,15BD 8D 94 D0 LDA #94
,15BF 8D 08 D0 STA D008
,15C2 8D 32 D0 LDA #32
,15C4 8D 09 D0 STA D009
,15C7 8D B3 D0 LDA #B3
,15C9 8D 0A D0 STA D00A
,15CC 8D 32 D0 LDA #32
,15CE 8D 0B D0 STA D00B
```

```
,15D1 8D 02 D0 LDA #D2
,15D3 8D 0C D0 STA D00C
,15D6 8D 32 D0 LDA #32
,15D8 8D 0D D0 STA D00D
,15DB 8D F1 D0 LDA #F1
,15DD 8D 0E D0 STA D00E
,15E0 8D 32 D0 LDA #32
,15E2 8D 0F D0 STA D00F
,15E5 8D FF D0 LDA #FF
```


Assembler-Gegenstück für die Zeilen 500 – 510 (Alle 8 Sprites aktivieren)

```
,15E5 A9 FF 00 LDA #FF
,15E7 8D 15 D0 STA D015
,15E9 4C 05 16 JMP 1605
```

Von Adresse \$15EA bis \$1604 sehen Sie hier die Assembler-Gegenstücke zu den beiden Schleifen, welche die Bewegung der Sprite nach unten bzw. nach oben bewirken.

Zuerst wird das Y Register mit dem Startwert 51 geladen. Dann werden die Sprites durch den Befehl JSR 1605 an der neuen Position angezeigt und durch den Befehl JSR 161E eine Warteschleife durchlaufen.

Diese ist zwischen den einzelnen Bewegungsschritten der Sprites notwendig, da man ansonsten die Bewegung der Sprites gar nicht als solche wahrnehmen würde, weil das Assembler-Programm im Vergleich zum BASIC-Programm um ein Vielfaches schneller läuft.

Sobald die Warteschleife durchlaufen ist, wird der Inhalt des Y Registers um 1 erhöht und der nächste Durchlauf beginnt. Die Schleife endet, sobald der Inhalt des Y Registers den Wert 229 erreicht hat.

Dann wird das Y Register mit dem Wert 228 (hexadezimal \$E4) geladen und die Bewegung nach oben durch laufende Verringerung des Inhalts des Y Registers durchgeführt. Auch hier wird wieder nach jedem Bewegungsschritt die Warteschleife durchlaufen.

Sobald im Y Register der Wert 50 unterschritten wird, endet die Schleife und die Geister sind wieder am oberen Bildschirmrand angelangt.

```
,15EA A0 33 00 LDY #33
,15EC 20 05 16 JSR 1605
,15EF 20 1E 16 JSR 161E
,15F2 C8 00 00 INY
,15F3 C0 E6 00 CPY #E6
,15F5 D0 F5 00 BNE 15EC
,15F7 A0 E4 00 LDY #E4
,15F9 20 05 16 JSR 1605
,15FC 20 1E 16 JSR 161E
,15FF 88 00 00 DEY
,1600 C0 31 00 CPY #31
,1602 D0 F5 00 BNE 15F9
,1604 60 00 00 RTS
```

Von Adresse \$1605 bis \$161D sehen Sie das Assembler-Gegenstück zum BASIC-Unterprogramm von Zeile 700 bis 790 (setzen der Y Koordinaten für die Sprites) und von Adresse \$161E bis \$1625 das Unterprogramm für die Warteschleife.

```

-----
,1605 8C 01 D0 STY D001
,1608 8C 03 D0 STY D003
,160B 8C 05 D0 STY D005
,160E 8C 07 D0 STY D007
,1611 8C 09 D0 STY D009
,1614 8C 0B D0 STY D00B
,1617 8C 0D D0 STY D00D
,161A 8C 0F D0 STY D00F
,161D 60      RTS
-----

```

```

-----
,161E A2 00      LDX #00
,1620 E8        INX
,1621 E0 FF      CPX #FF
,1623 D0 FB      BNE 1620
,1625 60      RTS
-----

```

Abschließend möchte ich Ihnen noch ein Programm vorstellen, welches hauptsächlich die Positionierung der Sprites in horizontaler Richtung demonstrieren soll. Im Unterschied zur Y Koordinate kommen wir bei der X Koordinate ab den Werten 256 nicht mehr mit einem Byte aus.

Was soll das Programm machen? Im Grunde nichts, was wir nicht schon gemacht haben. Es soll die 8 Sprites an bestimmten Positionen am Bildschirm anzeigen. Besonders an den Positionen ist aber eben, dass sie teilweise jenseits der X Koordinate 255 angesiedelt sind.

Vor dem eigentlichen Programmcode habe ich einen Datenbereich vorgesehen, in dem die Daten für das Sprite, die Farben der Sprites, die Positionen welche die Sprites beim Starten des Programms einnehmen sollen und noch einige weitere Daten abgelegt sind.

```

.M 1500
:1500 00 AA 00 02 AA 80 0A AA
:1508 A0 0A 0A 2A 2A A8 2B
:1510 AA E8 2F EB F8 AF EB FA
:1518 AD EB 7A AD EB 7A AB AA
:1520 EA AA AA AA AA AA AA AA
:1528 AA AA AA A2 8A 8A A2 82
:1530 AA AA AA 02 80 80 02 00
:1538 8A 80 82 02 80 82 02 80
:1540 01 02 04 08 10 20 40 80
:1548 00 02 03 04 07 0D 0E 0F
:1550 40 01 32 FF 00 4B 64 00
:1558 64 18 00 7D 18 00 96 18
:1560 00 AF 18 00 C8 18 00 E1
:1568 06 01 01 99 49 15 60 0A
:1570 A9 0B A2 00 9D F8 07 E8
:1578 E0 08 D0 F8 A2 00 BD 00
:1580 15 9D C0 02 E8 E0 3F D0

```

Von Adresse \$1500 bis \$153F liegen die Spritedaten.

Von Adresse \$1540 bis \$1547 liegen die 2er Potenzen von 2 hoch 0 bis 2 hoch 7 (warum wir die brauchen wird sich im Laufe der Beschreibung noch zeigen)

Von Adresse \$1550 bis \$1567 liegen die Sprite-Koordinaten, auf denen sie beim Starten des Programms positioniert werden sollen.

An Adresse \$1568 liegt die gemeinsame Farbe 1 und an Adresse \$1569 liegt die gemeinsame Farbe 2.

Ab Adresse \$1570 geht es dann los mit dem Programmcode.

.D	1570				
,	1570	A9	0B		LDA #0B
,	1572	A2	00		LDX #00
,	1574	9D	F8	07	STA 07F8,X
,	1577	E8			INX
,	1578	E8	08		CPX #08
,	157A	D0	F8		BNE 1574
,	157C	A2	00		LDX #00
,	157E	BD	00	15	LDA 1500,X
,	1581	9D	C0	02	STA 02C0,X
,	1584	E8			INX
,	1585	E8	3F		CPX #3F
,	1587	D0	F5		BNE 157E
,	1589	A9	FF		LDA #FF
,	158B	8D	1C	D0	STA D01C
,	158E	A9	00		LDA #00
,	1590	8D	1B	D0	STA D01B
,	1593	AD	68	15	LDA 1568
,	1596	8D	25	D0	STA D025
,	1599	AD	69	15	LDA 1569
,	159C	8D	26	D0	STA D026
,	159F	A2	00		LDX #00
,	15A1	BD	48	15	LDA 1548,X
,	15A4	9D	27	D0	STA D027,X
,	15A7	E8			INX

\$1570 - \$157A

Blocknummer 11 in die Speicherstellen 2040 – 2047 eintragen

\$157C - \$1587

Kopieren der Spritedaten in den Speicherblock 11

\$1589 - \$158B

Alle Sprites als mehrfarbig festlegen.

\$158E - \$1590

Alle Sprites haben Priorität gegenüber dem Hintergrund

\$1593 - \$1596

Gemeinsame Farbe 1 festlegen

\$1599 - \$159C

Gemeinsame Farbe 2 festlegen

\$159F - \$15AA

Individuelle Farben für die Sprites festlegen

\$15AC - \$15AE

Alle Sprites aktivieren

\$15B2 - \$15C6

.D	15B2								
,	15B2	A8				TAY			
,	15B3	B9	40	15		LDA	1540,Y		
,	15B6	90	06			BCC	15BE		
,	15B8	0D	10	D0		ORA	D010		
,	15BB	4C	C3	15		JMP	15C3		

,	15BE	49	FF			EOR	#FF		
,	15C0	2D	10	D0		AND	D010		
,	15C3	8D	10	D0		STA	D010		
,	15C6	60				RTS			

Hier kommt nun die Speicherstelle \$D010 (dezimal 53264) ins Spiel. Auch in dieser Speicherstelle ist jedem Bit ein Sprite zugeordnet. Jedes dieser Bits dient als zusätzliches Bit für die Darstellung von X Koordinaten, welche größer als 255 sind.

Der Wertebereich für die X Koordinaten bewegt sich zwischen 0 und 511, wobei das Sprite im Bereich von 24 bis 320 voll sichtbar ist.

In den Speicherstellen, in denen die X Koordinaten jedes Sprites gespeichert sind, bringt man jedoch nur Werte von 0 bis 255 unter. Für die Darstellung der Werte von 256 bis 511 fehlt uns noch ein zusätzliches Bit und genau aus diesem Grund gibt es in der Speicherstelle \$D010 ein zusätzliches Bit für jedes Sprite, um auch diesen Wertebereich abdecken zu können.

Das Unterprogramm im oben genannten Adress-Bereich dient dazu, das Bit für ein bestimmtes Sprite entweder auf den Wert 0 oder auf den Wert 1 zu setzen. Im Akkumulator übergibt man dem Unterprogramm die Nummer des Sprites (0-7) und über das Carryflag signalisiert man dem Unterprogramm, ob man das Bit auf 0 oder auf 1 setzen will.

Für den Fall, dass man das Bit auf 0 setzen will, setzt man das Carry Flag mit dem Befehl CLC auf 0 und für den Fall, dass man das Bit auf 1 setzen will, setzt man das Carry Flag mit dem Befehl SEC auf 1.

Hier kommen nun die 2er Potenzen ins Spiel, welche in den Speicherstellen \$1540 bis \$1547 abgelegt sind.

Mit dem Befehl TAY kopieren wir die Nummer des Sprites in das Y Register und holen uns mit dem Befehl LDA \$1540,Y die 2er Potenz in den Akkumulator, welche zu der jeweiligen Spritenummer gehört.

Hier eine Tabelle mit den dezimalen 2er Potenzen für jede Spritenummer:

Sprite 0	Sprite 1	Sprite 2	Sprite 3	Sprite 4	Sprite 5	Sprite 6	Sprite 7
1	2	4	8	16	32	64	128

Haben wir beim Aufruf des Unterprogramms das Carryflag mit dem Befehl CLC zurückgesetzt, dann wird durch den Befehl BCC zur Adresse \$15BE verzweigt, weil wir in diesem Fall das jeweilige Bit in der Speicherstelle \$D010 zurücksetzen müssen.

Dazu wird zunächst durch den Befehl EOR #\$FF das Einerkomplement der Spritenummer gebildet (welche sich ja nun im Akkumulator befindet) und dann eine UND-Verknüpfung mit dem aktuellen Inhalt der Speicherstelle \$D010 durchgeführt, wodurch das zur Spritenummer zugehörige Bit zurückgesetzt wird. Anschließend wird das Ergebnis zurück in die Speicherstelle \$D010 geschrieben, wodurch die Änderung wirksam wird.

Ist das Carryflag beim Aufruf des Unterprogramms hingegen gesetzt, dann wird eine ODER-Verknüpfung der Spritenummer im Akkumulator mit dem Inhalt der Speicherstelle \$D010 durchgeführt, wodurch das zur Spritenummer zugehörige Bit gesetzt wird.

Anschließend wird ein Sprung zur Adresse \$15C3 durchgeführt.

Dort steht der Befehl, welcher den Inhalt des Akkumulators, also das Ergebnis der ODER-Verknüpfung, wieder in die Speicherstelle \$D010 zurückschreibt, wodurch die Änderung wirksam wird.

Durch den Befehl RTS findet dann der Rücksprung aus dem Unterprogramm statt.

Address	Op Code	Op Name	Op Type	Op Comment
.D 15C7				
, 15C7	48	PHA		
, 15C8	0A	ASL		
, 15C9	A8	TAY		
, 15CA	A5 FA	LDA	FA	
, 15CC	99 00 D0	STA	D000	,Y
, 15CF	A5 FC	LDA	FC	
, 15D1	99 01 D0	STA	D001	,Y
, 15D4	A5 FB	LDA	FB	
, 15D6	F0 08	BEQ	15E0	
, 15D8	68	PLA		
, 15D9	38	SEC		
, 15DA	20 B2 15	JSR	15B2	
, 15DD	4C E5 15	JMP	15E5	

, 15E0	68	PLA		
, 15E1	18	CLC		
, 15E2	20 B2 15	JSR	15B2	
, 15E5	60	RTS		

Die Spritenummer wird im Akkumulator übergeben, die Speicherstellen \$FA (dezimal 250), \$FB (dezimal 251) und \$FC (dezimal 252) werden genutzt, um die einzelnen Komponenten der Position an das Unterprogramm zu übergeben.

Durch den Befehl PHA wird die übergebene Spritenummer zunächst auf dem Stack gesichert, weil sie durch den folgenden Befehl gleich wieder überschrieben wird.

Durch den Befehl ASL wird also die übergebene Spritenummer mit zwei multipliziert. Warum machen wir das?

Deswegen multiplizieren wir die Spritenummer mit 2 und addieren das Ergebnis zur Adresse 53248 hinzu, um auf die richtige Speicherstelle für den niederwertigen Anteil der X Koordinate des Sprites zu kommen.

Spritenummer	Spritenummer * 2	Adresse 53248 + Spritenummer * 2
--------------	------------------	-------------------------------------

0	0	53248
1	2	53250
2	4	53252
3	6	53254
4	8	53256
5	10	53258
6	12	53260
7	14	53262

Damit wir die Y indizierte absolute Adressierung nutzen können, kopieren wir den Inhalt des Akkumulators (welcher ja nun die Spritenummer * 2 enthält) durch den Befehl TAY in das Y Register.

Durch den Befehl LDA \$FA laden wir den niederwertigen Anteil der X Koordinate in den Akkumulator und durch den Befehl STA \$D000,Y schreiben wir diesen Wert in die für das jeweilige Sprite korrekte Speicherstelle (siehe Tabelle oben)

Als nächstes holen wir uns aus der Speicherstelle \$FC die Y Koordinate und schreiben den Wert durch den Befehl STA \$D001,Y an die richtige Speicherstelle für das jeweilige Sprite.

Als Basisadresse nehmen wir dieses mal jedoch nicht die Adresse 53248, sondern die Adresse 53249, damit wir auf die richtigen Speicherstelle für die Y Koordinate kommen.

Spritenummer	Spritenummer * 2	Adresse 53249 + Spritenummer * 2
0	0	53249
1	2	53251
2	4	53253
3	6	53255
4	8	53257
5	10	53259
6	12	53261
7	14	53263

Nun müssen wir nur noch den höherwertigen Anteil der X Koordinate unterbringen. Diesen haben wir als Parameter für das Unterprogramm in der Speicherstelle \$FB abgelegt.

Wir laden diesen Wert also mit dem Befehl LDA \$FB in den Akkumulator.

Ist der Wert gleich 0, dann ist die X Koordinate kleiner als 256 und wir müssen in der Speicherstelle 53264 das Bit für das jeweilige Sprite auf 0 setzen.

Falls der Wert in der Speicherstelle \$FB gleich 0 ist, dann wird durch den LDA Befehl das Zeroflag gesetzt und der Befehl BEQ verzweigt zu der Adresse \$15E0.

Dort wird die Spritenummer wieder vom Stack in den Akkumulator geholt, das Carryflag zurückgesetzt und das Unterprogramm zum Setzen / Zurücksetzen des Bits in der Speicherstelle 53264 aufgerufen.

Dieses beginnt an der Speicherstelle \$15B2, daher der Befehl JSR \$15B2.

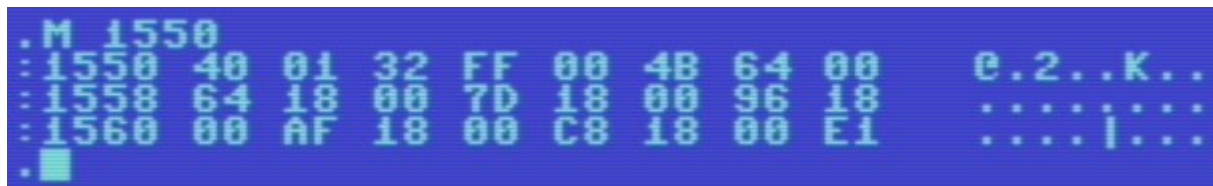
Durch den Befehl JMP \$15E5 springen wir zum Befehl RTS, um den Rücksprung aus dem Unterprogramm durchzuführen.

Ist der Wert in der Speicherstelle \$FB ungleich 0, dann wird ebenfalls wieder die Spritenummer vom Stack in den Akkumulator geholt, dieses mal jedoch das Carryflag gesetzt und das Unterprogramm zum Setzen / Zurücksetzen des Bits in der Speicherstelle 53264 aufgerufen.

Danach sind wir beim Befehl RTS angelangt und jede Komponente für die Position des Sprites befindet sich in der richtigen Speicherstelle.

Im Adressbereich \$1550 bis \$1567 (dezimal 5456 bis 5479) liegen die Koordinaten auf welche die Sprites beim Start des Programms positioniert werden sollen.

Hier ist zuerst das niederwertige Byte der X Koordinate abgelegt, dann folgt das höherwertige Byte der X Koordinate und dann die Y Koordinate, für die wir ja nur ein Byte benötigen.



.M 1550											
:1550	40	01	32	FF	00	4B	64	00		e.2..K..	
:1558	64	18	00	7D	18	00	96	18		
:1560	00	AF	18	00	C8	18	00	E1	I...	

Den Beginn machen die drei Bytes für Sprite 0, dann folgen die drei Bytes für Sprite 1 und das setzt sich fort bis Sprite 7.

Das niederwertige Byte der X Koordinate von Sprite 0 lautet \$40 (dezimal 64) und das höherwertige Byte lautet \$01 (dezimal 1), was eine X Koordinate von $64 + 256 * 1 = 320$ ergibt.

Dann folgt die Y Koordinate mit dem Wert \$32 (dezimal 50)

Sprite 0 wird also in der rechten oberen Ecke des sichtbaren Ausgabebereichs angezeigt.

Das niederwertige Byte der X Koordinate von Sprite 1 lautet \$FF (dezimal 255) und das höherwertige Byte lautet \$00 (dezimal 0), was eine X Koordinate von $255 + 256 * 0 = 255$ ergibt.

Dann folgt die Y Koordinate mit dem Wert \$4B (dezimal 75)

Dieses Schema setzt sich für alle restlichen Sprites fort.

\$15E6 - \$1600

.D	15E6				
,15E6	48			PHA	
,15E7	85	FD		STA	FD
,15E9	0A			ASL	
,15EA	65	FD		ADC	FD
,15EC	A8			TAY	
,15ED	B9	50	15	LDA	1550,Y
,15F0	85	FA		STA	FA
,15F2	B9	51	15	LDA	1551,Y
,15F5	85	FB		STA	FB
,15F7	B9	52	15	LDA	1552,Y
,15FA	85	FC		STA	FC
,15FC	68			PLA	
,15FD	20	C7	15	JSR	15C7
,1600	60			RTS	

Dieses Unterprogramm ist dafür zuständig, die drei Komponenten für die Position des Sprites, dessen Nummer im Akkumulator übergeben wird, aus dem Datenbereich zu holen und damit die Speicherstellen \$FA (dezimal 250), \$FB (dezimal 251) und \$FC (dezimal 252) zu befüllen, damit man das Unterprogramm zur Positionierung des Sprites aufrufen kann.

Als erstes wird die übergebene Spritenummer auf dem Stack zwischengespeichert und ebenso in der Speicherstelle \$FD (dezimal 253). Warum werden wir gleich noch sehen.

Da die Koordinaten für ein Sprite drei Bytes im Datenbereich belegen, müssen wir die Spritenummer mit drei multiplizieren, um jeweils auf die richtigen Bytes für das Sprite zu kommen.

Aber wie machen wir das? Multiplizieren können wir aktuell noch nicht in Assembler.

Das ist richtig, aber was wir bereits können, ist eine Multiplikation der Spritenummer mit zwei mittels des Befehls ASL, durchzuführen.

Dann bräuchten wir die Spritenummer nur noch einmal dazu addieren und schon haben wir das gewünschte Ergebnis, also die Spritenummer mal drei.

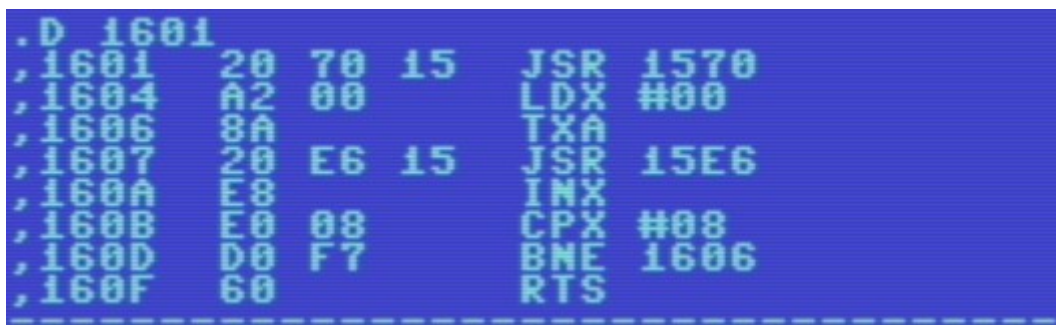
Das ist also die Erklärung für den Befehl ASL an der Adresse \$15E9 und den anschließenden Befehl ADC \$FD an der Adresse \$15EA.

Das Ergebnis der Multiplikation der Spritenummer mit drei kopieren wir wieder in das Y Register, um mittels der Y indizierten, absoluten Adressierung die drei Werte aus dem Datenbereich zu laden und auf die drei Speicherstellen \$FA, \$FB und \$FC zu verteilen, damit der Aufruf des Unterprogramms an der Adresse \$15C7 stattfinden kann.

Doch vor dem Aufruf müssen wir noch die Spritenummer vom Stack wieder in den Akkumulator holen, was durch den Befehl PLA erledigt wird.

Nachdem das Sprite durch den Aufruf des Unterprogramms auf seine Position gesetzt wurde, kehren wir mit dem Befehl RTS zum Aufrufer zurück.

\$1601 - \$160F



.D	1601				
,	1601	20	70	15	JSR 1570
,	1604	A2	00		LDX #00
,	1606	8A			TXA
,	1607	20	E6	15	JSR 15E6
,	160A	E8			INX
,	160B	E0	08		CPX #08
,	160D	D0	F7		BNE 1606
,	160F	60			RTS

Nun sind wir endlich beim Hauptprogramm angelangt.

Hier wird zunächst das Unterprogramm ab der Adresse \$1570 aufgerufen, welches die grundlegenden Einstellungen für die Sprites vornimmt (Farben, Priorität usw.)

Dann werden in einer Schleife die Koordinaten der Sprites aus dem Datenbereich gelesen und durch Aufruf des vorherigen Unterprogramms die Sprites auf diesen Koordinaten positioniert.

Um unser Programm nun endlich starten zu können, wechseln wir mit X zurück zu BASIC und führen den Befehl SYS 5633 aus.



Das folgende BASIC-Programm zeigt die unterschiedlichen Positionierungen der Sprites. Es befindet sich unter dem Namen SPXBAS auf der Diskette. Es enthält auch gleich das Maschinenprogramm als BASIC-Loader.

Wenn Sie das Programm mit RUN starten, werden die Sprites zunächst an den Positionen angezeigt, auf denen Sie auf dem vorherigen Screenshot zu sehen sind. Dann wird auf einen Tastendruck gewartet und wenn dieser erfolgt ist, erscheinen die Sprites auf neuen Positionen.



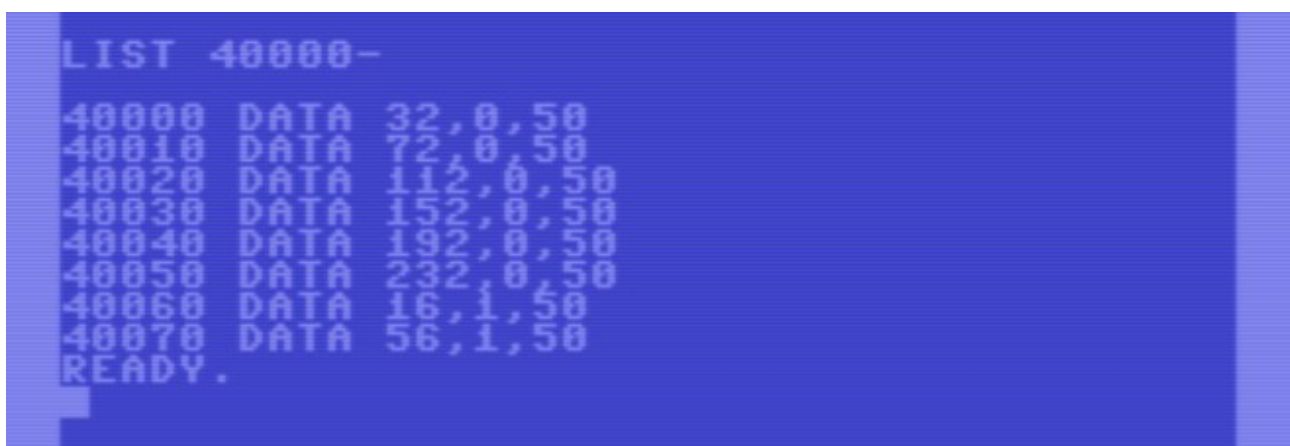
Erklärung zum Ablauf des Programms:

Zuerst wird das Maschinenprogramm aus den DATA-Zeilen von Zeile 32000 bis 32013 in den Speicher geschrieben und dann mit dem Befehl SYS 5633 gestartet.

Dadurch wird der Programmcode ab Adresse \$1601 aufgerufen und die Sprites gemäß ihrer im Datenbereich festgelegten Startpositionen auf dem Bildschirm positioniert.

Dann wird in Zeile 60 auf einen Tastendruck gewartet und durch die folgende Schleife werden die Sprites an andere Positionen verlagert.

Diese Positionen sind in den DATA-Zeilen ab Zeile 40000 enthalten.



Jede Zeile enthält als ersten Wert den niederwertigen Teil der X Koordinate, als zweiten Wert den höherwertigen Teil der X Koordinate und als dritten Wert die Y Koordinate.

Durch die POKE-Befehle in den Zeilen 90 bis 110 werden die neuen Positionen aus den DATA-Zeilen an die Stellen im Datenbereich geschrieben, an denen die Koordinaten der Sprites abgelegt sind.

In Zeile 120 wird die Spritenummer in die Speicherstelle 780 geschrieben, damit sie im Zuge des nächsten Befehls SYS 5606 in den Akkumulator geschrieben wird.

In Zeile 130 wird das Unterprogramm ab Adresse \$15E6 (dezimal 5606) aufgerufen, welches das jeweilige Sprite auf die neue Position transferiert.

Nun sind wir am Ende des Kapitels über Sprites angelangt. Eventuell werden Sie das Thema Kollisionserkennung vermissen, aber keine Sorge, wir werden dieses Thema wieder aufgreifen, wenn wir uns mit der Interrupt-Programmierung beschäftigen.