

Wie wär's mit:

Ιδ ιόξε ιρη - Γραφή

Hätten Sie gewußt, daß die Hieroglyphen in der Überschrift »Zeichen selber machen« bedeuten? Wir zeigen Ihnen, wie Sie solche oder ähnliche Zeichen selbst gestalten können und was dabei zu berücksichtigen ist.

Es gibt viele Anwendungen, die das Arbeiten mit einem abgeänderten Zeichensatz erforderlich machen. Vielleicht wollen Sie mathematische Sonderzeichen darstellen, ein griechisches Textverarbeitungs-Programm schreiben oder Spiele programmieren, die einen abgeänderten Zeichensatz benutzen. Wie auch immer, zuerst müssen wir uns die Frage stellen: »Was ist bei der Gestaltung eines Zeichensatzes zu beachten?«

Wir müssen uns überlegen, wo der neue Zeichensatz liegen soll. Möchte man kein vom Basic aus nutzbares RAM verwenden, bleibt nur der Bereich von \$A000 bis \$FFFF. Aber genau das ist für viele Programmierer die Schwierigkeit. Der Grund dafür ist der, daß der im C 64 eingebaute Videocontroller immer nur eine 16-KByte-Bank »sieht«. Das heißt, daß der Bildschirm, also das Video-RAM und der Zeichensatz in der gleichen 16-KByte-Bank liegen müssen. Eine Ausnahme bildet dabei die Einschaltkonfiguration. Denn hier liegt der Bildschirm bekanntlich bei \$400 und der Zeichensatz bei \$D000 unter dem I/O-Bereich, in dem die Register der Peripheriebausteine liegen, zum Beispiel auch die des Videocontrollers (Bild 1). Die Einschaltkonfiguration ist aber eine hardwaremäßige Maßnahme und läßt sich von einem Programm aus nicht nachbilden. Der Videocontroller beziehungsweise das Zeichensatz-ROM ist nämlich so verdrahtet, daß der Zeichensatz vom Videocontroller aus gesehen bei \$1000 liegt. Vom Prozessor aus gesehen liegt er aber bei \$D000. Das ist übrigens auch der Grund dafür, warum sich ein beliebiger Zeichensatz nicht ins RAM nach \$1000 legen läßt. Denn die Register im Controller sind so gesetzt, als läge der Zeichensatz tatsächlich ab \$1000.

Wir wollen den neuen Zeichensatz in das RAM nach \$E000 legen und das Video-RAM nach \$C000. Das hat den Vorteil, daß kein nutzbares RAM verlorengeht. Wir müssen uns aber jetzt um zwei Dinge kümmern:

1. Wie bekomme ich den Zeichensatz vom ROM ins RAM?
2. Welche Speicherzellen müssen verändert werden, so daß der Computer den neuen Zeichensatz und das neue Video-RAM akzeptiert?

Es ist unmöglich, mit einem Basic-Programm den Originalzeichensatz auszulesen. Denn das Basic-ROM und das Betriebssystem müssen ausgeblendet werden, um auf die Bank zu schalten, in der der Zeichensatz liegt (Bild 1). Dazu existiert im Computer die Speicherzelle 1. Wir wollen uns hier kurz mit den Bits 0, 1 und 2 beschäftigen, die für das Ein- und Ausblenden der verschiedenen RAM- oder ROM-Konfigurationen verantwortlich sind.

Bit 0: Ist dieses Bit auf »0« gesetzt, wird der Bereich von \$A000 bis \$BFFF auf RAM geschaltet. Daraus folgt, daß das Basic-ROM ausgeblendet wird (Bild 2).

Bit 1: Wird dieses Bit auf »0« gesetzt, so wird der Bereich von \$A000 bis \$BFFF und zusätzlich noch der Bereich von \$E000 bis \$FFFF auf RAM und der Bereich von \$D000 bis

\$DFFF auf Zeichensatz-ROM geschaltet. Dies ist die Speicherbelegung, in der wir den Zeichensatz auslesen können. Bei dieser Konfiguration existiert aber weder das Basic noch das Betriebssystem (Bild 3).

Bit 2: Dieses Bit blendet, wenn es auf »0« gesetzt ist, das gesamte ROM aus. Hier existieren also volle 64 KByte RAM (Bild 4).

Die Routine zum Kopieren des Zeichensatzes läßt sich, wie oben erwähnt, nicht in Basic schreiben. Es bleibt also nur ein Maschinenprogramm. Um aber auch dem Anfänger das Programm verständlich zu machen, möchte ich zuerst ein Basic-Programm vorstellen, mit dem das Kopieren des Zeichensatzes denkbar wäre. Anschließend wird dieses Programm in Maschinensprache umgesetzt.

Also zuerst das Basic-Programm:

```
10 POKE1,PEEK(1)AND253:REM BIT 1 AUSBLENDEN
20 VON=53248:NACH=57344
30 FOR X=0 TO 4096: REM ANZAHL DER ZU KOPIERENDEN
  BYTES
40 POKE NACH+X,PEEK(VON+X)
50 NEXT
60 POKE1,PEEK(1)OR2:REM BIT 1 EINBLENDEN
```

Nun das entsprechende Assembler-Programm:

```
10 -.EQ VON=53248
20 -.EQ NACH=57344
30 -.EQ X=4096
40 -.BA $C000
50 - LDA # <(VON) ;»VON« WIRD IM HI/LO-BYTE
  FORMAT
60 - LDX # >(VON)
70 - STA $FA ;NACH $FA/$FB GESPEICHERT
80 - STX $FB
90 - LDA # <(NACH);DAS GLEICHE FUER »NACH«
100 - LDX # >(NACH)
110 - STA $FC
120 - STX $FD
130 - LDY #0 ;ZAEHLER MIT NULL
  VORBELEGEN
140 - STY $FE
150 - STY $FF
151 - LDA 1 ;BIT 2 AUSBLENDEN
152 - AND #253
153 - STA 1
160 -LOOP LDA ($FA),Y ;DEN AKKU MIT »VON« LADEN
170 - STA ($FC),Y ;UND NACH »NACH« SPEICHERN
180 - INC $FA ;»VON« UM 1 ERHOEHEN
190 - BNE WE
200 - INC $FB
210 -WE INC $FE ;»NACH« UM 1 ERHOEHEN
220 - BNE WE1
230 - INC $FF
240 -WE1 INC $FE ;ZAEHLER UM 1 ERHOEHEN
250 - BNE WE2
260 - INC $FF
270 -WE2 LDA $FE ;ZAEHLER-»X«
280 - SEC
290 - SBC # <(X)
300 - LDA $FF
310 - SBC # >(X)
320 - BCC LOOP ;IST DAS ERGEBNIS NEGATIV
  DANN NACH LOOP
330 - LDA 1 ;ANSONSTEN BIT 2 WIEDER
  EINBLENDEN
340 - ORA #2
350 - STA 1
360 - RTS ;UND ZURUECK INS BASIC
```

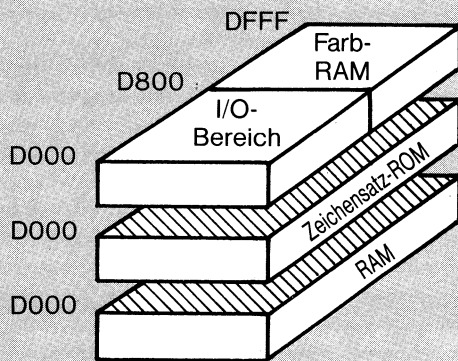


Bild 1. So sieht die Speicherbelegung im \$D-Bereich aus

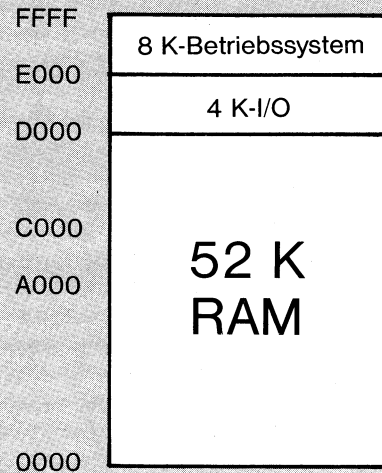


Bild 2. Ist Bit 0 auf »0« gesetzt, dann existiert diese Speicherbelegung

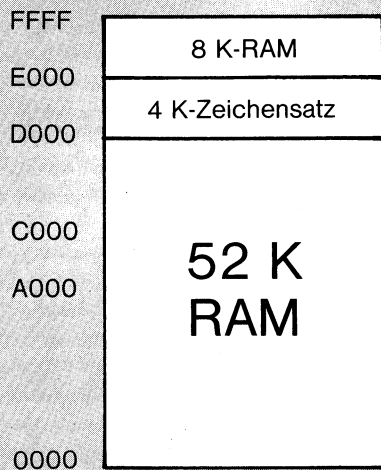


Bild 3. Speicherbelegung für den Fall, daß Bit 1 auf »0« gesetzt ist

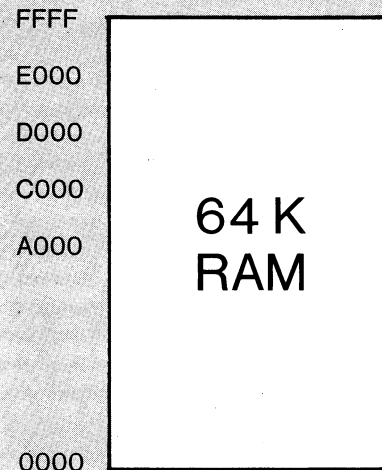


Bild 4. Ist Bit 2 auf »0« gesetzt, dann liegt diese Speicherbelegung vor

Natürlich läßt sich der Zeichensatz auch in jeden beliebigen RAM-Bereich verschieben. Dazu muß nur der Variablen »NACH« ein anderer Wert zugewiesen werden.

Wird der oben stehende Quelltext mit »Hypra-Ass« assembliert, so kopiert der Befehl SYS 49152 den Zeichensatz an die Adresse, dessen Wert in der Variablen »NACH« abgelegt wurde.

So, jetzt ist der Zeichensatz kopiert aber, noch nicht aktiviert. Dazu muß dem Videocontroller die neue Startadresse des Zeichensatzes mitgeteilt werden.

Im Videocontroller existiert das Register 24 bei Adresse \$D018 oder dezimal 53272. Dieses Register ist für die Lage des Zeichensatzes und des Video-RAMs verantwortlich. Es besteht, wie alle anderen Register auch, aus 8 Bit. Die Bits 1, 2 und 3 bestimmen die Lage des Zeichensatzes und die Bits 4, 5, 6 und 7 die Lage des Video-RAMs. Außerdem muß noch die 16 KByte-Bank, die der Videocontroller »sehen« soll, angewählt werden. Dazu dient die CIA2, und zwar das Register 0. Die entsprechende Adresse ist \$DD00 beziehungsweise dezimal 56567. Die beiden niederwertigen Bits, also Bit 0 und 1 legen nun die entsprechende 16 KByte-Bank fest. Die Lage ist wie folgt definiert:

xxxx xxx00: Bank 3 = Adresse \$C000 (49152) bis \$FFFF (65535)

xxxx xxx01: Bank 2 = Adresse \$8000 (32768) bis \$BFFF (49151)

xxxx xxx10: Bank 1 = Adresse \$4000 (16384) bis \$7FFF (32767)

xxxx xxx11: Bank 0 = Adresse 0 bis \$3FFF (16383)

Soll nun der mit dem oben stehenden Programm verschobene Zeichensatz bei \$E000 aktiviert werden, geben Sie die Zeile: POKE 56576,PEEK(56576)AND252:POKE 53272,56 ein. Der erste Teil der Zeile, bis zum Doppelpunkt, definiert die Lage der 16 KByte-Bank und der zweite Teil die Lage des Zeichensatzes und des Video-RAMs. Die Zahl, die in das Register 24 gePOKEt wird, setzt sich wie folgt zusammen:

Die Adresse des Zeichensatzes \$E000 entspricht binär 1110 0000 0000 0000

Die Bits 14 und 15 werden bedingt durch die Wahl der 16 KByte Bank auf »1« gesetzt. Die Bits 13, 12 und 11, also »100«, entsprechen nun den Bits 3, 2 und 1 im Register 24 des Videocontrollers. Das Bit 0 ist nicht belegt.

Die Adresse des Video-RAMs \$CC00 entspricht binär 1100 1100 0000 0000

Die Bits 14 und 15 werden wieder entsprechend auf 1 gesetzt. Die nun folgenden 4 Bits 13, 12, 11 und 10, also »0011« tauchen wieder als Bit 7, 6, 5, 4 im Register 24 des Videocontrollers auf.

Setzt man die Bits für den Zeichensatz und für das Video-RAM zusammen, ergibt sich die binäre Zahlenkombination 0011 100x

Das »x« steht für beliebig. Hier kann entweder eine »1« oder »0« eingesetzt werden. Daraus folgt, daß in das Register 24 entweder hexadezimal 38 oder 39 beziehungsweise dezimal 56 oder 57 gePOKEt werden muß.

Wenn Sie voller Erwartung die oben stehenden POKES eingegeben haben, waren Sie sicherlich enttäuscht; denn der Cursor war weder zu erkennen, noch ließ sich irgend etwas auf dem Bildschirm darstellen. Dem Betriebssystem fehlte nämlich noch die Information, an welcher Stelle im RAM der Bildschirm zu finden ist. Aber auch dafür existiert ein POKE. Ergänzen Sie die oben stehenden POKES zu

POKE 56576,PEEK(56576)AND252:POKE 53272,56:POKE 648,204 <RETURN>

Die Zahl, die in die Speicherzelle 648 gePOKEt wird, setzt sich aus den oberen vier Bit der Adresse zusammen, bei der das Video-RAM beginnt. Also für \$CC00 entsprechend \$CC oder dezimal 204.

Jetzt müßte es funktionieren. Das heißt, der neue Bildschirm muß noch gelöscht werden.

Der Zeichensatz ist nun verschoben und der Computer darauf abgestimmt. Wie aber lassen sich nun die Zeichen abändern, was ja schließlich der Sinn dieses Artikels sein sollte?

Dazu müssen wir uns zuerst mit dem Aufbau eines Zeichens und dem Aufbau des Zeichensatz-ROMs beschäftigen.

Jedes einzelne Zeichen besteht aus einer 8*8 Punkt-Matrix. Alle in dieser Matrix enthaltenen Punkte können entweder gesetzt oder nicht gesetzt sein; ein nichtgesetzter Punkt entspricht einer »0« und ein gesetzter entsprechend einer »1«. Jedes Zeichen ist nun zeilenweise gespeichert zu je 8 Bit beziehungsweise einem Byte. Das soll an dem Buchstaben »A« demonstriert werden:

**	0001 1000	18	024
****	0011 1100	3C	060
** **	0110 0110	66	102
*****	= 0111 1110 =	7E	= 126
** **	0110 0110	66	102
** **	0110 0110	66	102
** **	0110 0110	66	102
	0000 0000	00	000
	binär	hex	dez

Die Bytes liegen nun von oben angefangen nacheinander im Speicher, also hexadezimal:

18 3C 66 7E 66 66 66 00

Um das zu kontrollieren, versuchen Sie einmal, den Zeichensatz von \$D000 nach \$2000 zu verschieben und auch dort zu aktivieren. Für ganz Eilige schon hier die Lösung: Belegen Sie im oben stehenden Quelltext die Variable »NACH« mit \$2000 beziehungsweise dezimal 8192 (Zeile 20). Nach dem Assemblieren mit Hypra-Ass kann der Zeichensatz mit SYS 49152 nach \$2000 verschoben und mit den POKES:

POKE 56576,PEEK(56576)OR3:POKE 53272,24:POKE 648,4 <RETURN>

aktiviert werden. Hatten Sie den Computer zwischenzeitlich mal ausgeschaltet, ist nur ein POKE 53272,24 erforderlich. Denn der erste und letzte POKE stellen den Computer wieder auf die Einschaltkonfiguration.

Setzen Sie einen Monitor (zum Beispiel SMON) ein, läßt

sich die hexadezimale Darstellung des Zeichens »A« ab Adresse \$2008 mit dem Befehl »M 2008« auflisten. Ändern Sie einmal den Inhalt einer Adresse zwischen \$2008 und \$2010 ab. Sie werden feststellen, daß sich alle »A« auf dem Bildschirm entsprechend verändern.

Warum ist das Zeichen »A« ab Adresse \$2008 zu finden? Wie schon beschrieben, werden die Zeichen byteweise hintereinander im Speicher zu je 8 Byte abgelegt. Ein Blick in das Bedienungshandbuch zum C 64 wird Ihnen zeigen, daß das Zeichen »A« den Bildschirmcode »1« hat. Mit 8 multipliziert und zur Startadresse \$2000 hinzugezählt ergibt das die Startadresse \$2008 für das Zeichen »A«. Die Startadresse der anderen darstellbaren Zeichen läßt sich genauso ermitteln.

Für diejenigen, die gerne Spiele programmieren, gibt es noch eine weitere interessante Möglichkeit, den Multicolor-Modus. Um diesen einzuschalten, existiert im Videocontroller das Register 22 bei Adresse \$D016 (53270), und zwar das Bit 4. Ist dieses Bit auf 1 gesetzt, befindet sich der Videocontroller im Multicolor-Modus. Vom Basic aus läßt sich dieser Modus mit:

POKE 53270,PEEK(53270)OR16 <RETURN>

einschalten und mit:

POKE 53270,PEEK(53270)AND239 <RETURN>

wieder ausschalten.

Versuchen Sie doch gleich einmal, den Multicolor-Modus mit der ersten der beiden Zeilen einzugeben. Sie werden nur noch unleserliche Zeichen auf dem Bildschirm entdecken. Woran liegt das nun?

Nun, wie Sie vielleicht wissen, existiert neben dem Video-RAM auch ein Farb-RAM, in dem der Computer die Farbinformationen für die Bildschirmdarstellung speichert. Dieses Farb-RAM liegt im Bereich von \$D800 bis \$DBFF und kann nicht verschoben werden. Da es vom Umfang her 1 KByte groß ist, kann im normalen Modus jedem Zeichen eine andere Farbe gegeben werden. Jede Bildschirmposition hat ein korrespondierendes Byte im Farb-RAM. Auf die Farbgebung der Zeichen haben aber nur die vier unteren Bits, also Bit 3, 2, 1 und 0 eine Wirkung. Dabei hat Bit 3 eine spezielle Bedeutung. Ist es gesetzt, so wird das Zeichen im Multicolor-Modus, ist es nicht gesetzt, im Normal-Modus dargestellt. Daher lassen sich Multicolor- und normale Zeichen beliebig auf dem Bildschirm mischen. Jedem Multicolor-Zeichen werden in Abhängigkeit von den im Zeichensatz gesetzten Bits drei Farben zugeordnet. Dazu wird jedes Byte in vier Bit-Paare aufgeteilt. Nehmen wir als Beispiel wieder das Zeichen »A«, dann würde das so aussehen:

**	00 01 10 00
****	00 11 11 00
** **	01 10 01 10
*****	= 01 11 11 10
** **	01 10 01 10
** **	01 10 01 10
** **	01 10 01 10
	00 00 00 00

Die Farbzusordnung der einzelnen Bit-Paare ist wie folgt definiert:

00: Hintergrundfarbe 0 = Bildschirmfarbe = Adresse = \$D021 (53281)

01: Hintergrundfarbe 1 = Adresse \$D022 (53282)

10: Hintergrundfarbe 2 = Adresse \$D023 (53283)

11: Sind diese beiden Bits in einem Bit-Paar gesetzt, wird die Farbe gesetzt, die durch die 3 unteren Bits, also Bit 2, 1 und 0, im Farb-RAM definiert wurde.

Mit den Informationen, die Sie hier erhalten haben, dürfte die Gestaltung eines Zeichensatz-Generators kein Problem mehr sein. Lassen Sie Ihrer Kreativität freien Lauf.

(ah)