Reise durch das Wunderland der Grafik

Begleiten Sie uns auf einer abenteuerlichen, aber ungefährlichen Reise durch das Wunderland der Grafik. Entdecken Sie die (fast) grenzenlosen Grafikmöglichkeiten des Commodore 64.

Ist es Ihnen auch so ergangen: Sie lesen Testberichte über den Commodore 64, seine hervorragenden Grafikmöglichkeiten, eine Auflösung von 320 x 200 Punkten, und Ihnen schweben die phantastischen Abbildungen von Computergrafiken vor, die Sie nun auch alle selbst realisieren können, wenn Sie diesen Computer in Händen halten. Dann, nach mehr oder weniger vielen Anstrengungen, sitzen Sie vor Ihrem eigenen Commodore 64, neben sich das 170 Seiten dicke Handbuch, und arbeiten sich durch alles hindurch. Aber wo ist diese schöne Grafik? Nachdem Sie frustriert ein paar Sprites über den Bildschirm ziehen ließen und die Ballspiele aus dem Handbuch anfangen, Sie zu langweilen, geht die Suche los, wie man denn nun eine hübsche dreidimensionale Grafik auf den Bildschirm zaubern kann: Im Handbuch ist nichts zu finden. Dann gibt es nur zwei Möglichkeiten: Entweder Sie lassen die Grafik Grafik sein oder Sie beginnen eine Odyssee durch Buchläden, Computerzeitschriften und auch durch die Speicherzellen Ihres Computers, um sie nach langen Irrwegen endlich zu finden: die hochauflösende Grafik.

Wenn Ihre Barschaft es erlaubt, können Sie sich natürlich einiges an Schweiß ersparen: Inzwischen wird ja eine Reihe von mehr oder weniger brauchbarer Grafik-Software angeboten. Aber was Sie nicht bezahlen können, ist eine Menge von Erkenntnissen über die Möglichkeiten, die — verborgen hinter dornigen POKE-Hecken — darauf warten, von Ihnen wie Dornröschen wachgeküßt zu werden. Glauben Sie mir, diese Küsse sind es wert, sich in das Byte-Gewirr zu stürzen, zumal ich versuchen werde, Ihnen dazu den von mir schon gebahnten Weg hier und in den kommenden Folgen zu zeigen. Bei der Gelegenheit werden Sie feststellen, daß Sie nicht nur Dornröschen (die hochauflösende Grafik) wachgeküßt haben, sondern — erinnern Sie sich an die Gebrüder Grimm — auch das ganze Volk im Schloß fing an zu leben. Mit nüchternen Worten: Sie machen sich dabei eine Menge anderer, sonst schlafender Eigenschaften Ihres Computers zunutze.

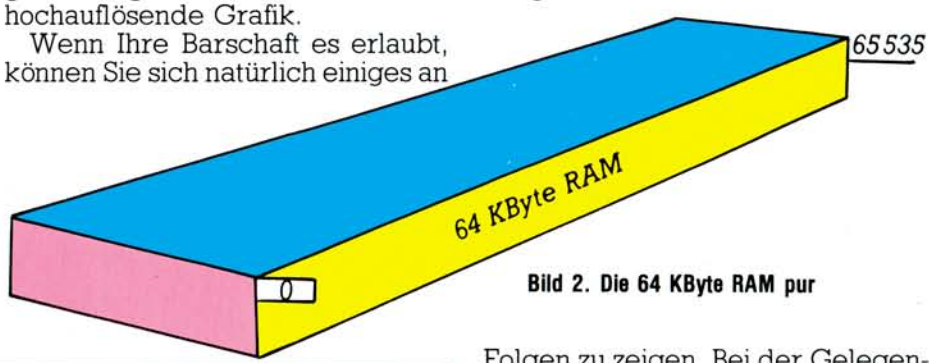


Bild 2. Die 64 KByte RAM pur

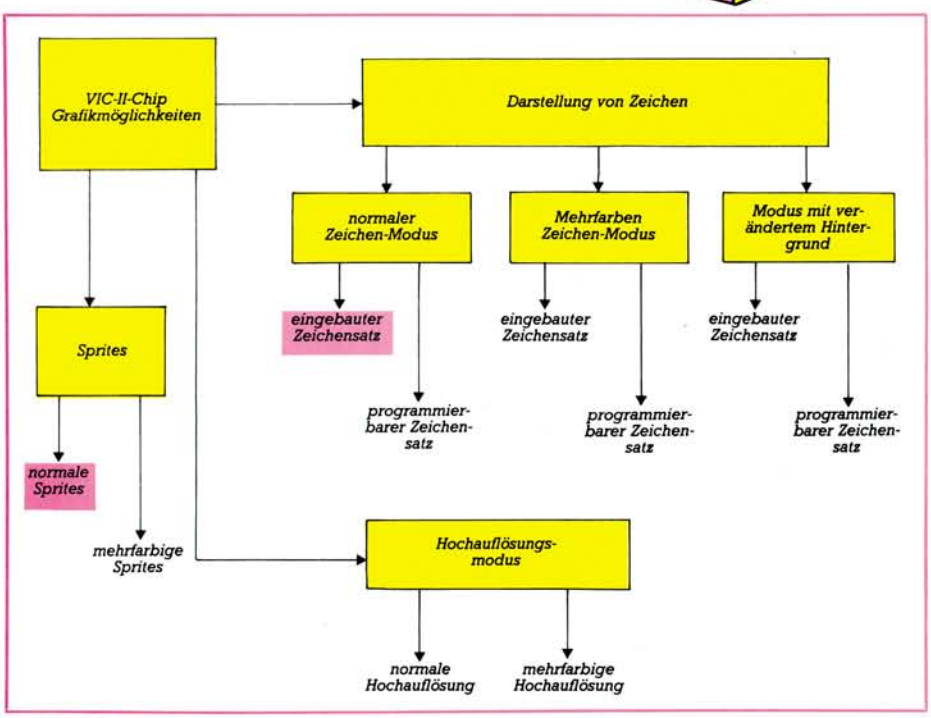


Bild 1. Die Vielfalt der Grafikmöglichkeiten des Commodore 64

Die Grafikmöglichkeiten des C 64

Noch einige Worte, bevor wir an die Arbeit gehen: Wissen Sie eigentlich, welche Grafik-Vielfalt der Commodore 64 hat? In Bild 1 ist sie aufgeführt.

Im Handbuch finden Sie davon nur zwei angegeben: Den »normalen« Zeichensatz und die »normalen« Sprites. Zu dem Schema in Bild 1 gehören eigentlich noch einige Kleinigkeiten, auf die wir noch stoßen werden. So kann beispielsweise der Bildschirm auf verschiedene Grafikarten aufgeteilt werden und so weiter. Aber um so weit zu gelangen, müssen wir uns erst eine Weile durch die Byte-Dornen gehauen haben.

Sie dürfen schon Ihren Computer anschalten, denn wir werden bei der nun folgenden Reise durch das

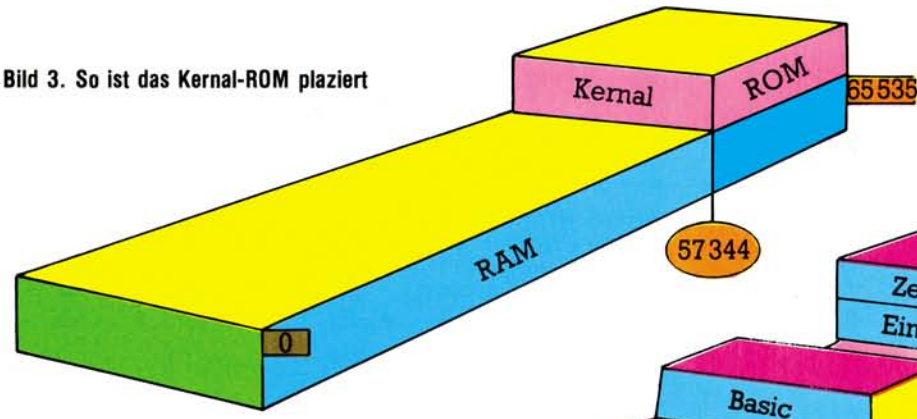


Bild 3. So ist das Kernal-ROM plaziert

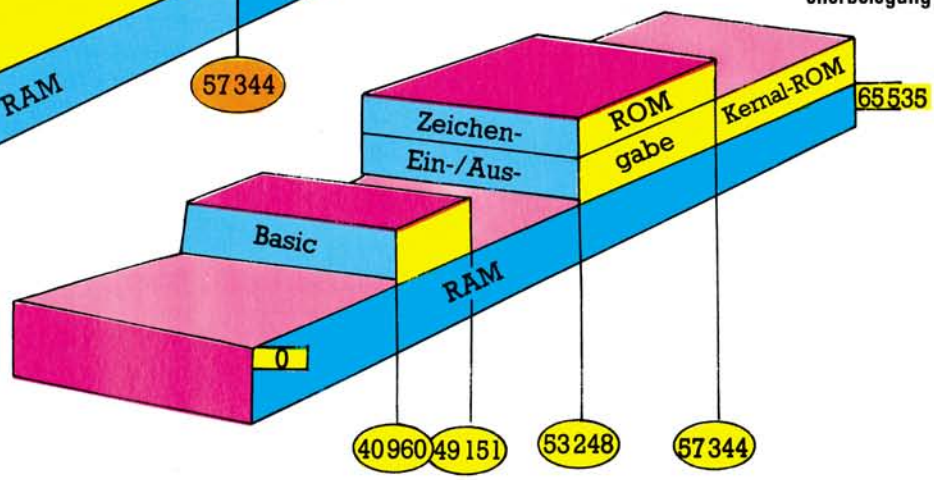


Bild 4. Die Ein-/Ausgabebausteine, das Zeichen- und Basic-ROM mit der entsprechenden Speicherbelegung

Grafik-Land einiges ausprobieren. Allerdings werden wir uns vorübergehend dabei von den Gebrüdern Grimm trennen müssen, denn die Landschaft, durch die wir uns dabei bewegen, paßt besser zum Wunderland der kleinen Alice: Ganze Landschaftsteile sind da und doch nicht da, Gebäude verschwinden und andere tauchen wieder auf, die Zeit wird gedehnt, Spiegelbilder erscheinen.

Wir fangen zunächst mal damit an, den Ast, auf dem wir sitzen, abzusägen. Damit Sie sich trotzdem keinen Schaden zufügen, sollten Sie vorher noch alle Programme, die Sie eventuell noch im Computer haben, auf Kassette oder Diskette abspeichern. Erledigt? Dann geben Sie doch jetzt mal folgendes ein:

```
POKE 1,PEEK(1) AND 252
»RETURN«
```

Jetzt ist Ihr Computer scheintot. Kein Cursor mehr, keine Reaktion auf Tastendrucke. Aber dafür haben Sie jetzt tatsächlich die 64 KBytes RAM (Worterklärungen siehe Kasten), die in der Kaltstartmeldung des C 64 angekündigt sind, zur freien Verfügung (Bild 2). Nur ist nichts damit anzufangen! Die 65536 freien Bytes Speicherkapazität liegen wie jungfräulicher Ackerboden vor uns und wir Benutzer sind ihnen völlig egal: Es muß also außer dem, was über einen Adreßbus von 16 Bit normalerweise erreichbar ist, noch etwas anderes vorhanden sein, etwas, das uns die Kommunikation mit unserem Computer erlaubt.

Natürlich ist das auch jetzt vorhanden, nur der 64 sieht es nicht. Das zwingt uns leider dazu, einige für ihn verschwundene Gebäude durch Aus- und Einschalten schlagartig wieder sichtbar zu machen. Welche Gebäude sieht der Computer jetzt wieder?

Da ist zunächst einmal das Betriebssystem (auch Kernal-ROM genannt). Alle Hausnummern unserer Byte-Straße von 57344 bis 65535 haben noch eine Etage außer dem RAM-Erdgeschoß: Im ersten Stock liegt dort das Kernal-ROM (Bild 3).

Dieses Kernal-ROM ist sozusagen der Organisator unseres Computers — nichts geht ohne ihn, wie wir ja eben, als er weggeschaltet war, gesehen haben. Allerdings braucht auch der beste Organisator noch einige andere lebenswichtige Partner. Damit wir überhaupt mit dem Computer in Verbindung treten können, sind noch einige weitere Hausnummern zumindest einstöckig (Bild 4).

53248 bis 57343. Ein- und Ausgabebausteine

40960 bis 49151. Basic-ROM

Es gibt sogar Häuser mit einem zweiten Stock.

53248 bis 57343. Zeichen-ROM

Zum Zeichen-ROM werden wir später kommen und die Ein- und Ausgabebausteine werden uns eine ganze Weile beschäftigen. Ohne Basic-ROM könnten wir nur in Maschinensprache unseren C 64 programmieren und eben nicht in Basic.

Wie kann unser Computer diese anderen Etagen nützen? Es sind ja insgesamt statt 64 KByte jetzt 88 KByte oder exakt 90112 Zimmerfluchten zu je 8 Bit, auf die man ge-

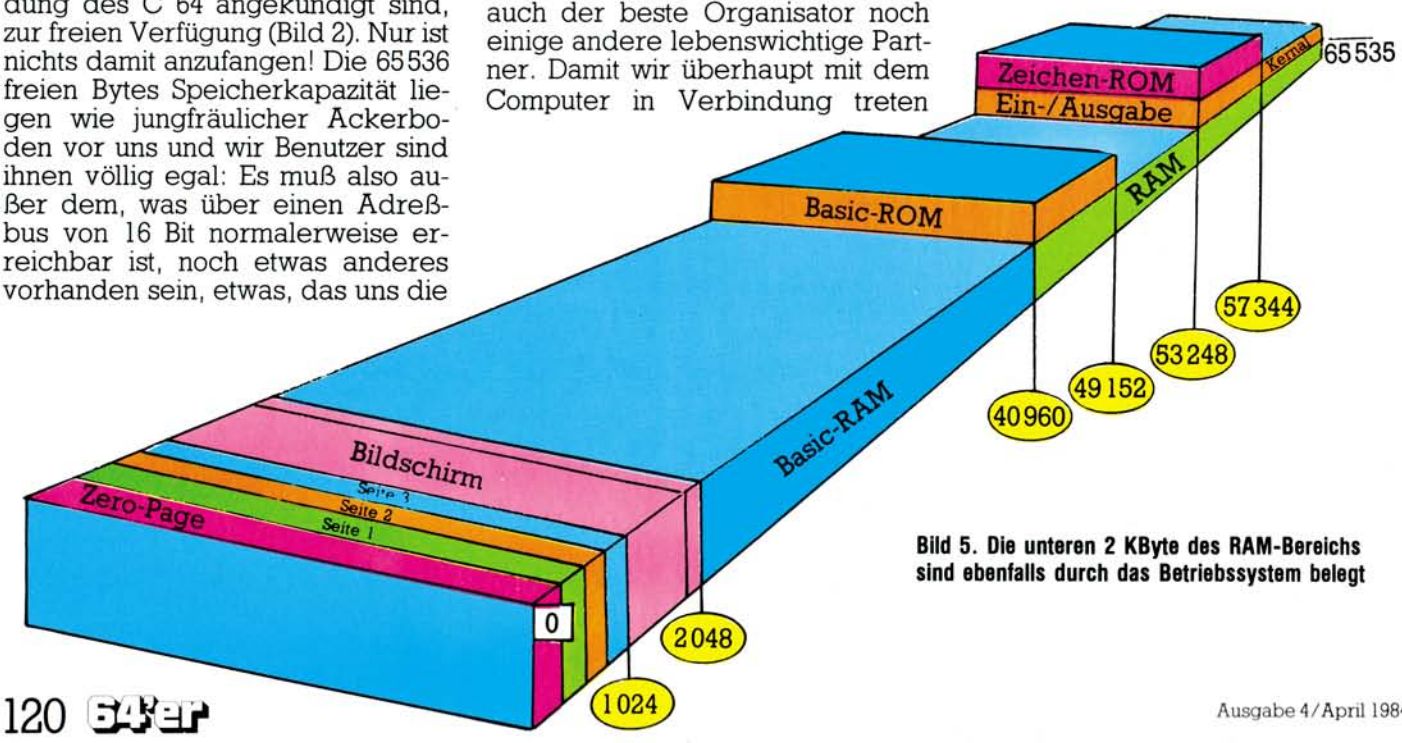
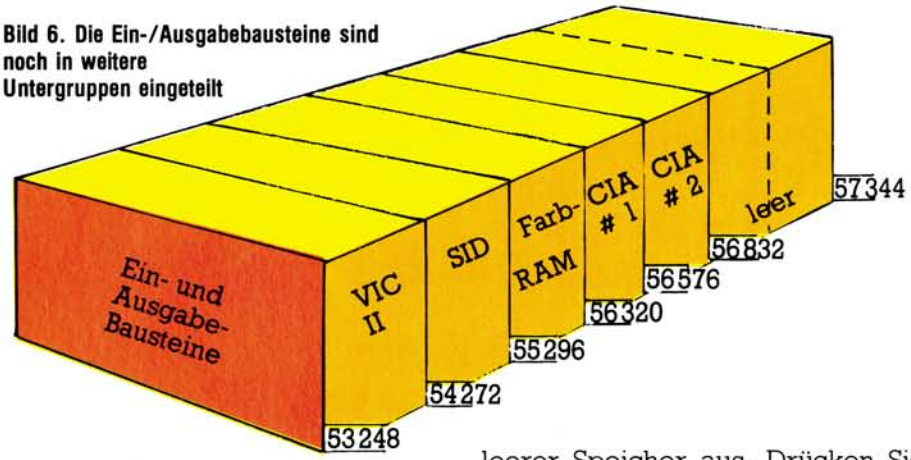


Bild 5. Die unteren 2 KByte des RAM-Bereichs sind ebenfalls durch das Betriebssystem belegt

langen können muß. Man kann sich das so vorstellen, daß zum Beispiel zwischen den Hausnummern 53248 und 57343 einen Augenblick lang die Ein- und Ausgabebausteine stehen, dann verschwinden sie und im nächsten Augenblick steht das Zeichen-ROM dort, dann wieder die Ein-/Ausgabebausteine und so weiter. Also tatsächlich ein Wunderland, das wir an dieser Byte-Straße finden. Gesteuert wird dieses Auftauchen und Verschwinden vom Betriebssystem. In Wirklichkeit bleibt alles an seinem Platz.

Bild 6. Die Ein-/Ausgabebausteine sind noch in weitere Untergruppen eingeteilt



Man sollte meinen, daß der Commodore 64 durch diese ganzen Zaubereien, denen er sich da widmen muß, wenig Zeit für uns Benutzer hat! Aber weit gefehlt, unser Computer ist so schnell, daß für uns seine Zeit gedehnt aussieht. Der Puls des Computers rast mit zirka 1 Million Schlägen pro Sekunde, während unser Puls rund einhalbmal in der Sekunde schlägt: In der Zeit also, in der unser Augenlid einmal zwinkert, hat der Computer schon tausende von Operationen vorgenommen und steht gewissermaßen mit den Fingern trommelnd bereit, unser Kommando endlich zu empfangen. Eigentlich langweilt er sich die meiste Zeit. Wie man seine Leistungsfähigkeit effektiver als mit Basic-Programmen ausnutzen kann, dazu werden wir in dieser Serie auch noch kommen.

Zunächst wollen wir uns mal ein wenig umsehen in unserem Speicher. Dazu kann das angefügte Programm »SpeiLu« benutzt werden (siehe Listing). Ziemlich primitiv, für unsere Zwecke zunächst aber schon ausreichend, ist dieses kleine Programm:

```
10  INPUT"STARTADRESSE";A
20  FOR I=A TO A+255:PRINT
    PEEK(I);:NEXT
30  GET A$:IF A$="" THEN 30
40  IF A$="←" THEN A=I:GOTO
    20
```

Geben Sie nach dem RUN jetzt mal als Startadresse 6000 ein. Es erscheinen Blöcke von Nullen und Blöcke von 255ern (meistens). Wenn Sie "←" drücken, kommen die nächsten 256 Bytes auf den Bildschirm und so weiter. So sieht ein leerer Speicher aus. Drücken Sie irgendeine Taste (außer "←") und starten Sie mit RUN erneut. Mit der Eingabe von 2048 blicken wir in die ersten 256 Bytes unseres Basic-RAMs. Der wüste Zahlensalat in der oberen Hälfte des Bildschirms ist die Computer-Version unseres Programms. Danach ist dann wieder leerer Speicher zu sehen.

Im Speicher ist einiges los

Wieso eigentlich 2048 als Start des Basic-RAMs? Warum nicht 0? Sehen wir uns doch mal mit ein bißchen Geduld den RAM-Bereich von 0 bis 2048 an, also Starten des Programms und Eingeben von 0: Wir sehen einen nahezu vollen Speicher. Das ist die sogenannte Zero-Page, zu deutsch Null-Seite. Voll ist die Seite, weil sie uns das Betriebssystem abgezwackt hat, um dort eine Reihe wichtiger Werte zu speichern. Wie wichtig das ist, haben wir gesehen, als wir den Wert 55 des ersten Byte (auf dem Bildschirm jetzt die zweite Zahl oben links) durch unser Astabsägen verändert haben. Wenn wir jetzt "←" drücken, sehen wir die nächste Seite (page 1) auf dem Bildschirm. Auch diese Seite — obwohl sie jetzt größtenteils leer ist (Nullen und 255er-Blöcke) — gehört dem Betriebssystem: Es ist der sogenannte Prozessorstapelspeicher. Drücken wir nochmal "←", dann erscheint Seite 2 (Adresse 512 bis 767). Hier ist zwar auch vieles leer (viele Nullen), aber wenn wir uns recht erinnern, sah der normale leere Speicher anders aus. Auch

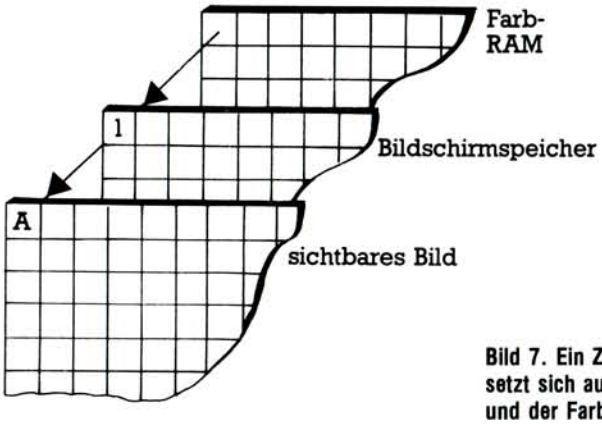


Bild 7. Ein Zeichen auf dem Bildschirm setzt sich aus der Bildschirm- und der Farbinformation zusammen.

diese Seite hat der Computer unserem Zugriff durch das normale Basic entzogen und speichert dort einige wichtige Angaben. Die Seite 3 erfüllt einen ähnlichen Zweck und außerdem befindet sich dort von 828 bis 1019 noch der Kassettenpuffer. Damit hat uns das Betriebssystem unseres Computer also schon das erste KByte des Speichers gemopst. Wenn Sie sich entsinnen, habe ich vorhin erwähnt, daß das Basic-RAM bei 2048 beginnt. Wie sieht es also im Bereich des zweiten KByte aus? Wenn wir uns mittels "←" die nächsten vier Seiten ansehen, marschieren stramme Kolonnen von Zahlen zwischen 48 und 57 (und viele 32) auf. Das sind Bildschirm-Codes von Zahlen und Leerstellen: Hier haben wir den Bildschirmspeicher mit insgesamt 1000 Bytes und dazu noch einige Bytes, die uns bei den Sprites beschäftigen werden. Jetzt sind wir wieder beim Basic-RAM ab 2048 angelangt. Haben Sie noch Lust? Dann probieren Sie noch ein bißchen weiter und sehen sich zum Beispiel das Basic-ROM zwischen 40960 und 49151 an oder das Betriebssystem oder ...

Dabei werden Sie dann nochmal einen freien RAM-Bereich zwischen 49152 und 53247 finden (Bild 5), der aber normalerweise nicht für Basic erreichbar ist. Jetzt ken-

Tabelle 1. Registerübersicht des VIC-II-Chips

Register	Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	53248	X-Position des Sprite Nr. 0. Dazu muß Register 16 beachtet werden							
1	53249	Y-Position des Sprite Nr. 0							
2	53250	X-Position des Sprite Nr. 1. Auch dazu, wie zu allen folgenden Sprites, muß Register 16 beachtet werden.							
3	53251	Y-Position des Sprite Nr. 1							
4	53252	X-Position des Sprite Nr. 2. s. o.							
5	53253	Y-Position des Sprite Nr. 2							
6	53254	X-Position des Sprite Nr. 3. s. o.							
7	53255	Y-Position des Sprite Nr. 3							
8	53256	X-Position des Sprite Nr. 4. s. o.							
9	53257	Y-Position des Sprite Nr. 4							
10	53258	X-Position des Sprite Nr. 5. s. o.							
11	53259	Y-Position des Sprite Nr. 5							
12	53260	X-Position des Sprite Nr. 6. s. o.							
13	53261	Y-Position des Sprite Nr. 6							
14	53262	X-Position des Sprite Nr. 7. s. o.							
15	53263	Y-Position des Sprite Nr. 7							
16	53264	Spr. 7, msb X-Pos.	Spr. 6, msb X-Pos.	Spr. 5, msb X-Pos.	Spr. 4, msb X-Pos.	Spr. 3, msb X-Pos.	Spr. 2, msb X-Pos.	Spr. 1, msb X-Pos.	Spr. 0, msb X-Pos.
17	53265	msb des Rasterregisters (Reg. 18)	Schaltbit für veränderten Hintergrundfarbmodus 1 = eingeschaltet	Schaltbit für Hochauflösungsmodus 1 = eingeschaltet	Schaltbit für Bildschirm »aus« 0 = normaler Bildschirm 1 = Bildschirmfarbe gleich Hintergrundfarbe	Schaltbit für Zeilenzahl 0 = 24 Zeilen 1 = 25 Zeilen	Wert der Zeilenverschiebung in Y-Richtung beim Smooth Scrolling		
18	53266	Rasterregister. Dazu kommt das msb in Bit 7, Register 17							
19	53267	Lichtgriffel X-Position							
20	53268	Lichtgriffel Y-Position							
21	53269	Ein- und Ausschalten von Sprites. 0 = Sprite aus. 1 = Sprite an							
		Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
22	53270			Reset-Bit, muß 0 sein, damit VIC-II-Chip arbeitet	Schaltbit für Mehrfarbmodus 1 = eingeschaltet	Schaltbit für Spaltenzahl 0 = 38 Spalten 1 = 40 Spalten	Wert der Spaltenverschiebung in X-Richtung beim Smooth Scrolling		

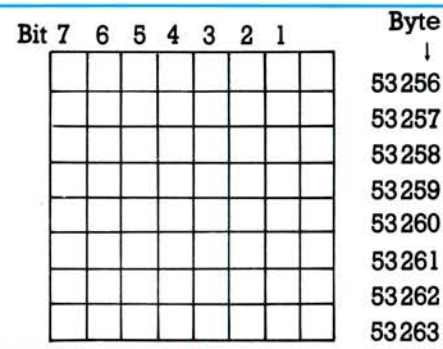


Bild 8. Das Zeichenmuster des Buchstaben A

```
CODE :? 1

ADRESSE   DEZ.   HEXA    BINAER      GRAFIK

  12296     24     18    00011000    - - - XX - - -
  12297     60     3C    00111100    - - XXXX - -
  12298    102     66    01100110    - XX - - XX -
  12299    126     7E    01111110    - XXXXXX -
  12300    102     66    01100110    - XX - - XX -
  12301    102     66    01100110    - XX - - XX -
  12302    102     66    01100110    - XX - - XX -
  12303      0     00    00000000    - - - - - - - -
```

So sieht der Buchstabe A mit den einzelnen Zahlen-Codes aus

nen wir — bis auf einige weitere Merkwürdigkeiten, zum Beispiel die versprochenen Spiegelbilder — unseren Computerspeicher schon ganz gut und können uns dem für die Grafik wichtigsten Speicherteil zuwenden: den Ein- und Ausgabebausteinen.

Der VIC-II-Chip

Verschaffen wir uns zunächst einen Überblick:

☐ Der Video-Interface-Controller 6567 (VIC-II) liegt zwischen den Hausnummern 53248 und 54271. Genaugenommen ist die letzte Register-Hausnummer allerdings schon 53294. Alle Register liegen tatsächlich nur auf 47 von den 1024 Hausnummern.

☐ An den VIC-II-Chip schließt sich

Register	Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
23	53271	Sprite-Vergrößerung in Y-Richtung. 0 = normale Größe. 1 = doppelte Größe.							
		Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
24	53272	Startadresse des Bildschirmspeichers				— Startadresse des Speicherbereichs, in dem die Zeichen als Punktmatrizen abzurufen sind. — Startadresse der Bit-Map			
25	53273	Interrupt-Flaggen-Register Interrupt				Lichtgriffel-Interrupt-Flagge	Sprite/Sprite-Kollision	Sprite/Hintergrund-Kollision	Raster-Interrupt-Flagge
26	53274	Interrupt-Masken-Register Interrupt				Lichtgriffel-Interrupt-Maske	Sprite/Sprite-Koll.-Maske	Sprite/Hintergrund-Kollision Maske	Raster-Interrupt-Maske
27	53275	Sprite/Hintergrund-Prioritätenregister. 0 = Sprite hat Priorität. 1 = Hintergrund hat Priorität							
		Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
28	53276	Sprite-Mehrfarbmodus-Register. 0 = Normaldarstellung. 1 = Mehrfarbmodus-Darstellung							
		Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
29	53277	Sprite-Vergrößerung in X-Richtung. 0 = normale Größe. 1 = doppelte Größe							
		Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
30	53278	Sprite/Sprite-Kollision. 0 = keine Berührung. 1 = Berührung							
		Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
31	53279	Sprite/Hintergrund-Kollision. 0 = keine Berührung. 1 = Berührung							
		Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
32	53280	unbenutzt				Farbe des Bildschirmrahmens			
33	53281	unbenutzt				Hintergrundfarbe Nr. 0 (normale Hintergrundfarbe)			
34	53282	unbenutzt				Hintergrundfarbe Nr. 1			
35	53283	unbenutzt				Hintergrundfarbe Nr. 2			
36	53284	unbenutzt				Hintergrundfarbe Nr. 3			
37	53285	unbenutzt				Sprite-Mehrfarben-Register Nr. 0			
38	53286	unbenutzt				Sprite-Mehrfarben-Register Nr. 1			
39	53287	unbenutzt				Sprite 0, Farbe			
40	53288	unbenutzt				Sprite 1, Farbe			
41	53289	unbenutzt				Sprite 2, Farbe			
42	53290	unbenutzt				Sprite 3, Farbe			
43	53291	unbenutzt				Sprite 4, Farbe			
44	53292	unbenutzt				Sprite 5, Farbe			
45	53293	unbenutzt				Sprite 6, Farbe			
46	53294	unbenutzt				Sprite 7, Farbe			

das Sound Interface Device 6581 (SID) an, welches von Hausnummer 54272 bis 55295 reicht. Das ist ein ebenfalls sehr verlockendes Nachbaranwesen (Musikliebhaber kommen hier auf ihre Kosten), welches wir bei dieser Gelegenheit aber nicht besuchen wollen.

☐ Von 55296 bis 56319 (genauer eigentlich nur bis 56295) liegt das Farb-RAM, Dornröschens hauseigene Malerei, die wir noch bemühen werden.

☐ Stippvisiten werden wir uns ergramm »SpeiLu« betrachten. Aus der Registerübersicht werden Sie beim Nachschlagen sehen, daß die Bit (Zimmer) 7-4 den Ort des Bildschirms im Speicher anzeigen und die Bit 3-0 im Normalfall etwas damit zu tun haben, wo die Zeichen (Buchstaben, Zahlen, Grafikzeichen etc.) abrufbar sind. Bevor wir daran gehen, dieses Byte zu verändern, wollen wir es im Urzustand erstmal unter die Lupe nehmen. Wenn Sie

PRINT PEEK(53272)

eingeben, werden Sie den Wert 21

Block	Adressenbereich	Zeichen	Muster abrufbar im Programm mit Code
0	53248 — 53759	Satz 1 von 0 bis 63 (0 bis ?)	0 — 63
	53760 — 54271	Satz 1 von 64 bis 127 (Grafikz.)	64 — 127
	54272 — 54783	Satz 1 von 0 bis 63 (Reversed)	128 — 191
	54784 — 55295	Satz 1 von 64 bis 127 (Reversed)	192 — 255
1	55296 — 55807	Satz 2 von 0 bis 63 (kleine Buchst.)	256 — 319
	55808 — 56319	Satz 2 von 64 bis 127 (Großbuchst. + Grafikz.)	320 — 383
	56320 — 56831	Satz 2 von 0 bis 63 (Reversed)	384 — 447
	56832 — 57343	Satz 2 von 64 bis 127 (Reversed)	448 — 511

Tabelle 2. Inhalt des Zeichen-ROMs

lauben beim Pförtner des Schlosses, der seine Wache bei den Hausnummern 56320 bis 56575 stehen hat, dem sogenannten Complex Interface Adapter 6526 (CIA Nr. 1). Die 1 rührt daher, daß er noch einen Kollegen hat, der das Revier von Adresse 56576 bis 56831 bewohnt und logischerweise CIA Nr. 2 heißt.

☐ Sozusagen Baugrund für Erweiterungen findet man noch zwischen den Speicheradressen 56832 und 57343. Die einzelnen Abteilungen sind in Bild 6 aufgeschlüsselt.

Von nun an wird uns Dornröschens Schloß, der VIC-II-Chip, ständig beschäftigen. Damit die Orientierung leichter fällt, ist die Tabelle 1 abgebildet, in der alle Registerinhalte wie auf einem Grundriß verzeichnet sind. Auf den ersten Blick sieht das zugegebenermaßen reichlich verwirrend aus — lassen Sie sich nicht erschrecken.

Sie stehen jetzt sozusagen mitten im Dornengestrüpp, und wenn wir gemeinsam den Weg hindurchgefunden haben, wird Ihnen alles verständlich sein, was da steht. Fangen wir mit der Hausnummer 53272 an:

Wenn sie Lust haben, können Sie sich den Inhalt der Adresse 53272 einmal mit dem beigefügten Pro-

Damit Sie nicht über Begriffe stolpern, sind sie hier erklärt:

RAM	**= Random Access Memory = Speicher für beliebigen Zugriff, also Schreiben und Lesen (POKE und PEEK) möglich.**
ROM	**= Read Only Memory = Speicher ist nur zum Lesen (PEEK)**
Speicher	**kann man sich vorstellen als lange Straße mit meist ebenerdigen Häusern und Hausnummern von 0 bis 65535**
Byte	**Ein Haus dieser Straße mit acht Zimmern. Man numeriert sie durch von 0 bis 7.**
Bit	**Ein Zimmer eines solchen Hauses. Es ist entweder etwas drin (Bit gesetzt, also = 1) oder nichts drin (Bit gelöscht, also = 0)**
Adreßbus	**Eine Art Kabinentaxi, das alle 65536 Häuser durch Angabe der Hausnummer ansteuern kann. Eine höhere Zahl als 65535 kann nicht angegeben werden.**
1 KByte	**= Einmal 1024 Bytes**
1 page	**= Eine Seite = ein Viertel von 1 KByte = 256 Bytes**

ausgedruckt bekommen. Haben Sie sich diesen Speicherplatz mittels »SpeiLu« angesehen, dann fanden Sie etwas wie

00010101

was der Binärausdruck der Dezimalzahl 21 ist. Aber dazu kommen wir noch. Das Betriebssystem setzt nach dem Einschalten das Byte 53272 automatisch auf diesen Wert, und wenn Sie sich an die Speicherreise mit dem kleinen Vierzeilen-Programm erinnern, dann wissen Sie auch noch, daß der Bildschirmspeicher damit auf die Startadresse 1024 festgelegt ist.

Wie Sie ebenfalls wissen, hat der Commodore 64 im Normalfall 40 Zeichen pro Zeile und 25 Zeilen, was insgesamt 1000 Zeichen pro Bildschirm ergibt. Deswegen hat der Bildschirmspeicher eine Ausdehnung von 1000 Bytes: von 1024 bis 2023. Die Aufteilung dieser 1000 Speicherplätze auf den Bildschirm ersehen Sie aus dem Handbuch auf Seite 138. Was ist nun drin in den Bildschirmspeicherstellen? Probieren wir es aus! Tippen Sie doch mal ein:

»shift + clear home« ABC »Return«

Natürlich taucht jetzt eine Fehlermeldung auf, die uns aber nicht kümmern soll. Jetzt steht ganz links oben (in Speicherplatz 1024) das A, dann B (1025) und C (1026). Nun wollen wir mal sehen, was der Computer sich merkt:

PRINT PEEK(1024), PEEK(1025, PEEK(1026) »Return«

Es erscheint 1 2 3 .

Wenn also auf dem Bildschirm ein A vorhanden ist, hat der Computer in der dazugehörigen Stelle seines Speichers eine 1 stehen, bei B eine 2, bei C eine 3 und so weiter. Lassen Sie uns den Bildschirm nochmal löschen mit »shift + clear home«. Dann gehen wir mit dem Cursor etwas abwärts und poken diese Kennzahlen in den Bildschirmspeicher:

POKE 1024,1:POKE 1025,2:
POKE1026,3 »Return«

Nach dem Return ist anscheinend nichts passiert. Erst wenn Sie mit dem Cursor dorthin fahren, wo eigentlich ABC stehen sollte, tauchen diese Buchstaben unter dem Cursor auf. Der Grund für dieses Verhalten liegt sicherlich darin, daß die Kennzahl 1 in der Speicherzelle 1024 alleine nicht genügt, das A sichtbar zu machen. Es hat automatisch die Farbe des Hintergrundes. Einem Zwilling des Bildschirmspeichers sind wir bei den Ein- und Ausgabebausteinen schon begegnet: dem Farb-RAM zwischen

```
1 REM*****************************************************************************
2 REM*                                                                           *
3 REM*                                                                           *
4 REM*                              S P E I L U                                  *
5 REM*                                                                           *
6 REM*                                                                           *
7 REM*   MANFRED THOMA  +  HEIMO PONNATH  2102 HAMBURG 93  VERINGSTRASSE 82     *
8 REM*                                                                           *
9 REM*****************************************************************************
10 POKE52,48:POKE56,48:POKE53280,6:POKE53281,6:POKE646,1
20 TE$=" ADRESSE  DEZ. HEXA   BINAER    GRAFIK"
100 PRINTCHR$(147)"  DARSTELLUNG VON ZEICHEN UND SPEICHER"
105 PRINT:PRINTCHR$(18);TAB(10);"====================="
110 PRINTCHR$(18):TAB(10)"THOMA/PONNATH HAMBURG"
115 PRINTCHR$(18);TAB(10);"======================"
120 PRINT:PRINT:PRINT"     (1) EINSEHEN IN EINEN SPEICHER"
130 PRINT:PRINT"     (2) DARSTELLUNG EINES ZEICHENS"
150 PRINT:PRINT:PRINT"       BITTE KENNZIFFER WAEHLEN !"
160 GETA$:IFA$=""THEN160
170 A=VAL(A$):IFA<0ORA>2THEN160
180 ONAGOSUB1000,2000
190 GOTO100
1000 PRINTCHR$(147);CHR$(18);TAB(2);"DARSTELLUNG EINES SPEICHERPLATZES:"
1010 PRINT:PRINT" SPEICHERADRESSE (0-65535) EINGEBEN"
1020 PRINT"ZURUECK MIT ZAHL AUSSERHALB 0 UND 65535":PRINT
1030 INPUT"ADRESSE :";AD:IFAD<0ORAD>65535THENRETURN
1040 DE=PEEK(AD):PRINT:PRINT:GOSUB10000:GOSUB20000:PRINTTE$:PRINT:GOSUB30000
1050 PRINT:PRINT:GOTO1010
2000 PRINTCHR$(147);CHR$(18);TAB(3);"ZEICHEN-DARSTELLUNG (CHARACTER)"
2010 IFTS=0THENGOSUB40000
2020 PRINT:PRINT"     GEBE DEN 'BILDSCHIRM CODE' DES"
2030 PRINT"      DARZUSTELLENDEN ZEICHENS EIN"
2040 PRINT"   = SIEHE HANDBUCH SEITE 133 - 134 ="
2050 PRINT:PRINT"  ZURUECK MIT ZAHL AUSSERHALB 0 UND 511":PRINT
2060 PRINT,:INPUT"CODE :";A:IFA<0ORA>511THENRETURN
2070 PRINT:PRINTTE$:PRINT
2080 FORAD=12288+8*ATO12288+8*A+7
2100 DE=PEEK(AD):GOSUB10000:GOSUB20000:GOSUB30000:NEXTAD
2110 GETA$:IFA$=""THEN2110
2120 RETURN
10000 HE$="":H$="0123456789ABCDEF":D=INT(DE/16):HE$=MID$(H$,D+1,1):D=DE-D*16
10010 HE$=HE$+MID$(H$,D+1,1):RETURN
20000 BI$="":DI=DE
20010 DI=DI/2:D$="0":IFDI<>INT(DI)THEND$="1"
20020 DI=INT(DI):BI$=D$+BI$:IFDI>0THEN20010
20030 IFLEN(BI$)<8THENBI$="0"+BI$:GOTO20030
20040 RETURN
30000 PRINTTAB(7-LEN(STR$(AD)))AD;TAB(13-LEN(STR$(DE)))DE;TAB(16)HE$;TAB(21)BI$;
30010 FORI=1TO8:W$=MID$(BI$,I,1)
30020 IFW$="1"THENPRINTTAB(30+I)CHR$(18);" ";CHR$(146);:GOTO30040
30030 PRINTTAB(30+I)".";
30040 NEXTI
30050 PRINT: RETURN
40000 PRINT:PRINT"KOPIEREN DER ZEICHEN INS RAM (AB 12288)"
40010 PRINT"          BITTE WARTEN"
40020 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
40030 FORI=0TO4095:POKE12288+I,PEEK(53248+I):NEXTI
40040 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
40050 POKE53272,(PEEK(53272)AND240)+12:TS=1:RETURN

READY.
```

Listing. Das Programm »SpeiLu« (Speicherlupe)

55296 und 56295. Wie er aufgeteilt ist (Bild 7) steht im Handbuch Seite 139 zusammen mit den Farbkennzahlen.

Wenn wir jetzt zum Beispiel noch eingeben:
POKE 55296,1:POKE 55297,3:POKE 55298,7 »Return«
dann sehen wir ein weißes A, ein cyanfarbenes B und ein gelbes C.

Übrigens, wenn Ihnen die aktuelle Farbe des Cursors oder der gerade verwendeten Zeichen nicht gefällt, dann probieren Sie doch mal
POKE 646, Farbkennzahl.

Und weil wir gerade bei den Farben sind, die Speicherzellen 53280 und 53281 steuern, mit Farbkennzahlen belegt, die Rahmen- und die Hintergrundfarbe. Mir persönlich gefällt zum Beispiel folgende Kombination sehr gut (auf Schwarzweiß-Bildschirm)
POKE 53280,11:POKE 53281,11: POKE 646,0 »Return«

Nun zu den Zeichen. Woher weiß der Computer, daß er ein A drucken muß, wenn eine 1 im Bildschirmspeicher steht? Das sagt ihm das Betriebsystem. Es teilt ihm mit, daß im Byte 53272 des VIC-II-Chips eine Kennzahl steht (Bit 1-3, Bit 0 wird nicht beachtet), die ihm wiederum sagt, wo die Muster für alle Zeichen zu finden sind. Merkwürdigerweise deutet diese Kennzahl auf eine Startadresse der Zeichenmuster von 4096! Das Zeichen-ROM, das wir bei unserer anfänglichen Speicherbegehung als 2. Stock im Bereich 53248 bis 57343 kennengelernt haben, ist davon meilenweit entfernt! 4096 liegt außerdem mitten in einem Bereich, der ständig von Basic-Programmen überschrieben wird.

Die genaue Lösung des Rätsels soll erst in einer der nächsten Folgen gegeben werden. Aufgrund einer technischen Eigenart des VIC-II-Chips werden vom Zeichen-ROM zwei Geisterbilder im Bereich 4096 bis 8191 und im Bereich 36864 bis 40959 erzeugt. Der VIC-II-Chip »meint«, er hole seine Zeichen-Muster aus diesen Bereichen. In Wirklichkeit bezieht er sie im Normalfall immer aus dem Zeichen-ROM. Das ist eine Eigenart, die so recht in Alices Wunderland paßt!
paßt!

Wie sehen diese Zeichen-Muster aus? Auch dazu können Sie das Programm »SpeiLu« benutzen. Wenn Sie sich damit beispielsweise mal das A ansehen, finden Sie ein Muster, wie es in Bild 8 dargestellt ist.

Dieses 8 x 8-Gitter (auch Matrix genannt) enthält also das Abbild des Zeichens A. Alle Zeichen sind auf diese Weise als Punktmuster gespeichert in jeweils acht Speicherzellen (hier also von 53256 bis 53263). Ein dunkles Feld bedeutet ein gesetztes Bit (=1; im Zimmer ist etwas drin), ein helles Feld ein gelöschtes Bit (=0; das Zimmer enthält nichts).

Das Zeichen-ROM hat an nullter Stelle von 53248 bis 53255 das Zeichen mit dem Bildschirmcode 0 (den Klammeraffen **@**), an erster Stelle von 53256 bis 53263 — wie wir sehen — das Zeichen mit dem Bildschirmcode 1 (also das A) und so weiter nacheinander in Form von je acht Bytes als Bit-Muster gespeichert. Wenn Sie im Handbuch die Seite 133 f. aufschlagen, dann können Sie die Tabelle 2 mit dem Inhalt des Charakter-ROMs besser verstehen.

Probieren Sie mal aus, sich die einzelnen Zeichen mit dem Programm »SpeiLu« durch Eingabe der Bildschirmcodes (im Handbuch bis 127 als Pokes bezeichnet) abbilden zu lassen.

Das Programm »SpeiLu« (der Name kommt von »Speicher-Lupe«) enthält noch einige für Sie bislang noch geheimnisvolle Einzelheiten: die Hexadezimal- und die Binärzahlen, das Interrupt-System, das Kopieren des Zeichen-ROMs. Dies alles hängt zusammen mit der Frage: Wie kann man sich eigene Zeichen bauen und verwenden? Wir werden sie in der nächsten Folge gemeinsam beantworten.

Mit dem bisher zurückgelegten Weg durch das Dornengestrüpp sind wir unserem Ziel, der hochauflösenden Grafik, schon ein ganzes Stück nähergekommen. Ich hoffe, daß Sie nach der Ruhepause bis zur nächsten Folge die zweite Etappe der Expedition zu Dornröschen zusammen mit mir durchführen werden.

(Heimo Ponnath)