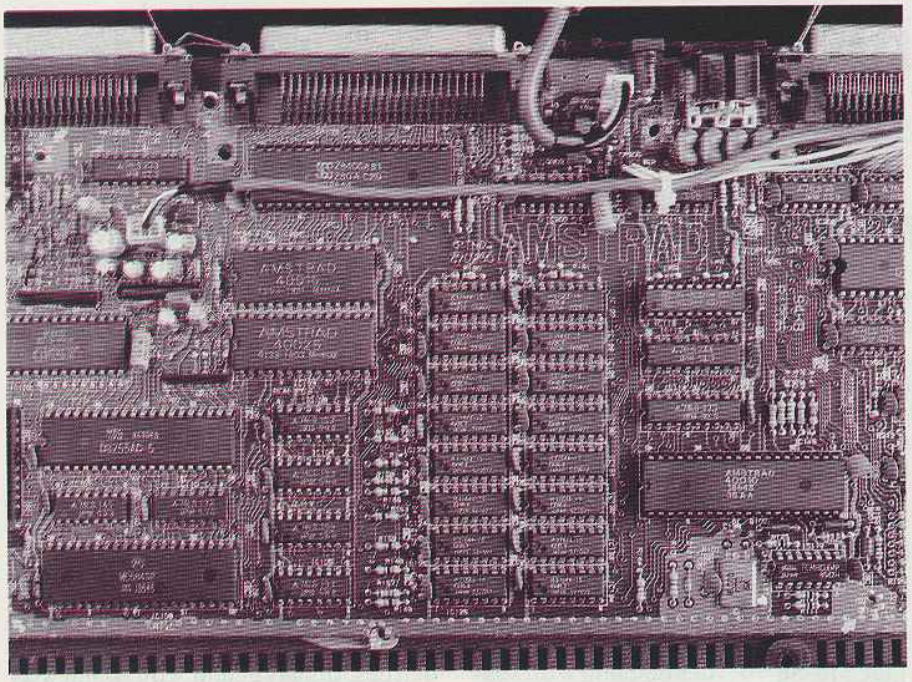


Teamwork im CPC

Der Videocontroller und das Gate-Array im CPC

Der Videocontroller und das Gate-Array sind die Bauteile im CPC, von denen wenig bekannt ist. Durch Änderung von Werten in deren Registern sind aber äußerst interessante Effekte zu erzielen. So ist es beispielsweise möglich, Bilder über den gesamten Bildschirm hinweg darzustellen.

Der Videocontroller und das Gate-Array warten schon seit 1984 geduldig, daß man ihnen die Last ihrer großen Geheimnisse von den zarten Drahtfüßchen nehme. Ersterer wurde höchstens einmal zum Umschalten in den 60-Hertz-Modus mit einem OUT-Befehl beehrt. Das Gate-Array dürfte bestenfalls als Zwischenhändler bei der schnellen BANK-Umschaltung gedient haben. Im Jahre 9 der CPC-Zeitrechnung ist eine Enthüllung dieser ungeahnten Fähigkeiten, denen ein Großteil der 16-Bit- und 32-Bit-Computer auch heute nichts Vergleichbares entgegenzusetzen hat, schon längst überfällig.

All jene, die bei Schlagworten wie "Hardwarenahe Assembler-Programmierung" lieber weiterblättern, sei hier gesagt: Verdrängt einmal für zehn Minuten Eure Ängste, und laßt Euch von den folgenden Seiten überraschen.

Wer sich aktiv an der Reise durch den CPC beteiligen möchte, sollte über Grundkenntnisse der Maschinensprache verfügen. Doch auch für alle anderen werden sich viele effektvolle Tricks zum Einbau in BASIC-Programme finden.

Der Schwerpunkt der folgenden Seiten liegt auf dem Grundverständnis der Vorgänge im Videocontroller und im Gate-Array, denn für hardwarenahes Programmieren ist das Wissen um das Innenleben dieser CPC-Bausteine unerläßlich. Den, der sich tapfer durch die Theorie geschlagen hat, erwartet ein mundgerecht zubereiteter Happen Praxis.

Beginnen wir mit dem Videocontroller. Zur Einstimmung empfehlen wir einen Blick auf die Auflistung aller CRTC-Register. Nach dem Einschalten sind die Register leer, der Bildschirm ist also dunkel. Der Prozessor beginnt seine Arbeit im ROM, und eine seiner ersten Tätigkeiten ist das Füllen sämtlicher CRTC-Register. (Besitzer eines Disassemblers können den Weg ab Adresse 0 im LOWER ROM verfolgen.) Vorher testet er noch das Vorhandensein einer Lötbrücke, um den CRTC gegebenenfalls mit den Daten für den 60-Hertz-Modus zu füttern. Ein Listing zum Umschalten in den 60-Hertz-Modus finden Sie in der Listingbox.

Der CRTC 6845(XX) von Motorola befindet sich schon seit Urzeiten am Markt und hat eine Verbreitung gefunden, die für einen Spezialchip einzigartig ist. So hat er zum Beispiel in den MDA-, Hercules- und CGA-Karten der PCs seinen festen Platz. Insgesamt bietet er vier Darstellungsmodi:

1) Textmodus mit 80*25 Zeichen und freidefinierbarem Hardware-Cursor,
2) Grafikmodus 0 (160*200 Punkte, 16 Farben),
3) Grafikmodus 1 (320*200 Punkte, vier Farben),
4) Grafikmodus 2 (640*200 Punkte, zwei Farben).

Auf den PCs der ersten Tage fand nur der Textmodus Verwendung, vor allem wegen der damals noch sehr hohen Preise für RAM-Bausteine. Die PCs hatten damals, wie unser CPC, 64 kByte Speicher. Von MBytes wagte man noch nicht einmal zu träumen. Im Textmodus verbraucht jedes Zeichen gerade zwei Bytes (ASCII und Farbinformation), das sind bei 80*25 Zeichen nicht einmal 4 kByte Bildschirmspeicher. Da galten die Grafikmodi 0 bis 2 mit ihren satten 16 kByte Video-RAM als Luxus.

Grafik – früher einmal ein Luxus

Und eben dieser Textmodus wurde von den Entwicklern des CPC nicht berücksichtigt, so daß die Z80A-CPU weiterhin jedes Zeichen auf unserem Bildschirm pixelweise aufbauen muß. Da sich ein Hardware-Cursor aber nur im Textmodus darstellen läßt, kann man die Register 10, 11, 14 und 15 als Relikt aus der PC-Welt getrost vergessen.

Dank Amstrads Preisbewußtsein ist jetzt also der aufgrund seiner weiten Verbreitung billigste CRTC für den visuellen Kontakt mit dem CPC zuständig. Und weil Amstrads Einkäufer noch nie besonders wählerisch waren, haben sie auch gleich über zehn verschiedene Typen des CRTC 6845 für die CPCs besorgt. Deren kleine Unterschiede werden uns noch mit einigen zusätzlichen Problemen belasten. Glücklicherweise sind diese Differenzen oft so minimal, daß sich die zehn Typen auf drei Hauptvertreter zurückführen lassen. Im folgenden werden wir diese mit Typ 0 bis 2

bezeichnen, wobei Typ 0 am besten für Tricks jeder Art geeignet ist und Typ 2 oft ziemlich böse reagiert.
Jetzt wäre es für jeden an der Zeit zu erfahren, welcher CRTC sich eigentlich in seinem eigenen CPC befindet.
Lösung 1: Aufschrauben, Staub absaugen und Typenbezeichnung entziffern: 6845 SP = Typ 0, 6845 R = Typ 1, 6845 P = Typ 2.
Lösung 2: Listing 2 abtippen und starten. Diesen CRTC-Test haben wir der französischen Demogruppe "LOGON SYSTEM" zu verdanken, die sich wohl einige Nächte mit öden Zeit- und Registervergleichen um die Ohren geschlagen haben muß.
Jetzt gilt es noch die Frage zu klären, wie man ein Register mit einem bestimmten Wert lädt. Der Z80-Prozessor tritt mit den Hardwarekomponenten über die I/O-Kanäle (INPUT/OUTPUT-Kanäle) in Verbindung. Diese kann man sich als Schläuche von der CPU zum jeweiligen Gerät vorstellen, in die man mit OUT Daten hineinschicken und mit IN (INP in BASIC) herauslesen kann. Jetzt könnte man natürlich 18 Schläuche zu den einzelnen CRTC-Registern verlegen, was aber in einem teuren und komplizierten Schlauchsalat enden würde.

I/O-Kanäle und Schlauchsalat

Man verwendet daher in der Praxis nur drei Leitungen, wobei man über Leitung 1 (&BC00) ein CRTC-Register auswählt, über Leitung 2 (&BD00) einen Wert hineinschreibt und über Leitung 3 (&BF00) den Inhalt eines Registers abfragt. Dieses Abfragen ist mit Vorsicht zu genießen. Bei CRTC-Typ 0 funktioniert es immerhin noch bei den

Portadressen: &BCXX (Auswahl), &BDXX (Schreiben), &BFXX (Lesen)

Reg.	Inhalt	Funktion
0	63	Gesamtbreite des Bildschirms inklusive Border und horizontaler Rücklauf
1	40	Zahl der Zeichen/Zeile (Wortmodus, d.h. 40 für 80 Bytes)
2	46	Beginn des horizontalen Rücklaufs (6 Zeichen nach rechtem Rand)
3	142	Feineinstellung des horizontalen Rücklaufs
4	38	Gesamthöhe des Bildschirms inklusive Border und vertikaler Rücklauf
5	0	Feineinstellung der Gesamthöhe in Pixelzeilen
6	25	Zahl der angezeigten Textzeilen
7	30	Beginn des vertikalen Rücklaufs (5 Textzeilen nach dem unteren Rand)
8	0	Interlace-Modus (25 Hz Bildwiederholfrequenz, 2*Auflösung)
9	7	Zahl der Pixelzeilen –1 pro Textzeile
10	0	Aussehen des nichtexistenten Hardware-Cursors
11	0	Aussehen des nichtexistenten Hardware-Cursors
12	48	Startadresse Bildschirmspeicher (HIGH = Bit 8–13)
13	0	Startadresse Bildschirmspeicher (LOW = Bit 0–7)
14	0	Position des nichtexistenten Hardware-Cursors (HIGH)
15	0	Position des nichtexistenten Hardware-Cursors (LOW)
16	??	Bildschirmadresse, bei der Lightpen-Impuls auftraf (HIGH)
17	??	Bildschirmadresse, bei der Lightpen-Impuls auftraf (LOW)

Die Register des CRTC 6845

Registern 12, 13, 16 und 17, während CRTC-Typ 2 nur bei 16 und 17 Werte ausspuckt. In der Praxis sieht das dann so aus:

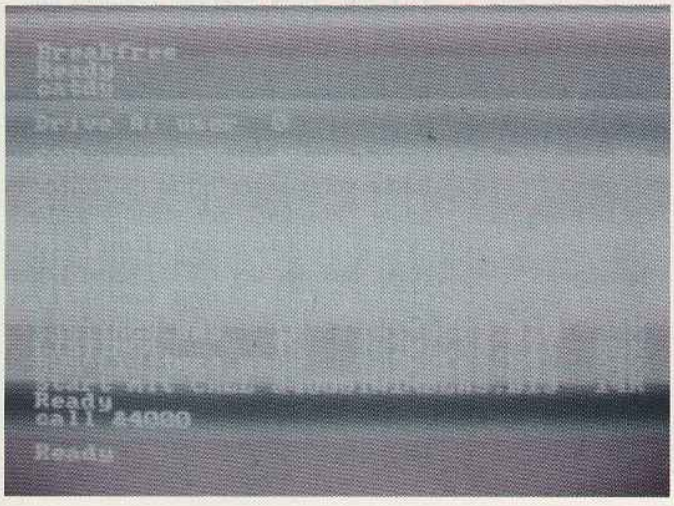
```
10 OUT &BC00, zu ladendes
   Register   0-13)
20 OUT &BD00,Wert (0-255)
```

Wichtig ist, daß es sich bei &BC00 bis &BF00 um Pseudo-16-Bit-Adressen handelt, im Gegensatz zu echten 16-Bit-I/O-Adressen. Um Register 1 auf den Wert 9 zu setzen, gibt man ein:

```
OUT &BC00,1: OUT &BD00,9
```

Das gleiche Ergebnis erzielt man aber auch mit:

```
OUT &BC28,1:OUT &BDAE,9
```



Die Rasterbalken, eine der ältesten Errungenschaften der Demoprogrammierung

Es kommt also nur auf das HIGHBYTE (= &BC,&BD) an. In Assembler wird die Sache schon etwas komplizierter, gibt es doch drei verschiedene OUT-Befehle: "OUT (XX),A", "OUT (C),Reg" sowie die Blockbefehle "OUTI", "OTIR", "OUTD" und "OTDR". Grundsätzlich steht in der Klammer die anzusprechende Portadresse und danach das CPU-Register, dessen Wert an eben diese I/O-Adresse geschickt werden soll.

Bei der Nutzung der IN- beziehungsweise OUT-Befehle sollte folgendes beachtet werden:
1) Der Befehl "OUT (XX),A" funktioniert nur bei 8-Bit-Portadressen zu denen &BCXX-&BFXX wohl nicht gehört. Fazit: In Verbindung mit dem CRTC schnellstens vergessen.
2) Der Befehl "OUT (C),Reg" krankt an seiner mißverständlichen Mnemonic (Name des Befehls), wird doch das C-Register nur bei "echten" 16-Bit-Adressen (beispielsweise Diskettencontroller &FB7E+&FB7F) zur Adressierung herangezogen. Sonst enthält es meistens sogar den Wert, der ausgegeben werden soll. Die komplette Verwirrung schafft da besonders der Befehl "OUT (C),C", unter dem sich Anfänger selten etwas Konkretes vorstellen können. Zum besseren Verständnis empfehlen wir, das C vor dem geistigen Auge durch ein BC zu ersetzen. Um wie oben Register 1 auf 9 zu setzen, könnte man schreiben:

```
LD A,1
LD BC,&BC00
OUT (BC),A
LD A,9
LD BC,&BD00
OUT (BC),A
```

So wäre es wohl jedem verständlich, nur dem Assembler leider nicht. Der beharrt auf besagtem "(C)", und weil obige Methode langsam und speicherplatzfressend ist, benutzen wir eine kürzere und damit schnellere Methode. (Wie bei BASIC gesehen, ist der Wert des C-Registers oder LOWBYTES bei der Adressierung unerheblich.)

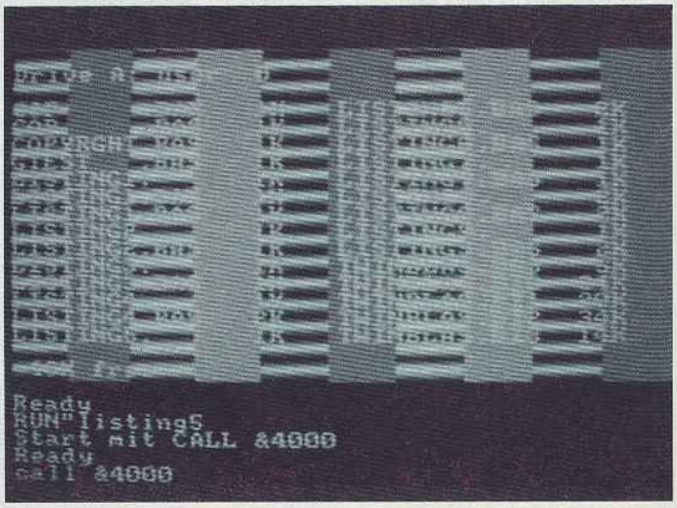
```
LD BC,&BC01
OUT (C),C
LD BC,&BD09
OUT (C),C
```

3) Die Blockout-Befehle OTIR und OTDR kann man wegen der bei Portadressen sinnlosen Wiederholung gleich abschreiben, bleiben noch OUTI und OUTD, mit denen absolute Geschwindigkeitsfanatiker noch ein paar Mikrosekunden bei längeren OUT-Ketten herausschinden können. Da wir sie aber in Verbindung mit dem Gate-Array brauchen, seien sie erwähnt. OUTI läßt sich am einfachsten so beschreiben:

```
LD A,(HL)
DEC B
OUT (C),A
INC HL
```

Für OUTD gilt analog:

```
LD A,(HL)
DEC B
OUT (C),A
DEC HL
```



Splitt-Raster – Rastertechnik verschärft

Ihnen wird schon aufgefallen sein: Das "DEC B" steht vor dem eigentlichen Befehl OUT; ein Umstand, der vielen unbekannt sein dürfte. (Wer den Befehl OUTI schon einmal zusammen mit dem Gate-Array verwendete, hat seine Daten unbewußt über Adresse &7E geschickt, was glücklicherweise aber auch funktioniert.)
Auf das obige Beispiel angewandt:

```
LD HL,DATA
LD B,&BD
OUTI
LD B,&BE
OUTI
RET
DATA: DB 1, 9
```

Nun zurück zur CPU, die noch immer im ROM beschäftigt ist (Teile des Betriebssystems ins RAM kopieren, Einschaltmeldung ausgeben, Bildschirmmodus initialisieren und so weiter).
Der CRTC wurde also endlich mit der oben besprochenen Methode "OUT (C),Reg" gefüttert und beginnt nun sofort mit der Darstellung des Bildes, dessen Daten er direkt aus dem standardmäßig 16 kByte großen Bildschirmspeicher holt. Dieser liegt beim CPC normalerweise im Bereich &C000 bis &FFFF. Intern verwaltet der CRTC mehrere Zähler, auf die man leider keinen Zugriff hat, die er aber ständig mit den Werten in seinen Registern vergleicht, wobei er bei Übereinstimmung eine bestimmte Aktion ausführt. Man kann also durch die Register indirekten Einfluß auf die Abläufe im CRTC nehmen.
Bitte "BORDER 26" eingeben, wir folgen jetzt dem Elektronenstrahl auf seinem Weg zum Monitor, wobei wir in der linken oberen Ecke des sichtbaren Bildschirmausschnittes beginnen (wo BORDER in PAPER übergeht.) Genau an diesem Punkt setzt der CRTC folgende interne Zähler auf 0:
1) den Textzeilenzähler,
2) den Pixelzeilenzähler und
3) den Zeichenzähler.
Des weiteren kopiert er den Inhalt der Register 12 und 13, die ja die Startadresse des Bildschirmspeichers in leicht veränderter Form enthalten, in einen "Pointer", der ständig aktualisiert wird und so immer auf die Adresse zeigt, aus der der CRTC seine Daten nimmt. Warum Register 11 nicht &C0, sondern &30 enthält, obwohl der Bildschirmspeicher bei &C000 beginnt, ist aus der Abbildung ersichtlich.
Der CRTC beginnt also mit der Darstellung der ersten Pixelzeile, wobei er im Wortmodus arbeitet, also immer zwei

Endwert	Wert	CPC	Farbe	Endwert	Wert	CPC	Farbe
&54	20	00	Schwarz	&5F	31	14	Pastellblau
&44	04	01	Blau	&4E	14	15	Orange
&55	21	02	Hellblau	&47	07	16	Rosa
&5C	28	03	Rot	&4F	15	17	Pastellmagenta
&58	24	04	Magenta	&52	18	18	Hellgrün
&5D	29	05	Hellviolett	&42	02	19	Seegrün
&4C	12	06	Hellrot	&53	19	20	Helles Blaugrün
&45	05	07	Purpur	&5A	26	21	Limonengrün
&4D	13	08	Hellmagenta	&59	25	22	Pastellgrün
&56	22	09	Grün	&5B	27	23	Pastellblaugrün
&46	06	10	Blaugrün	&4A	10	24	Hellgelb
&57	23	11	Himmelblau	&43	03	25	Pastellgelb
&5E	30	12	Gelb	&4B	11	26	Leuchtendweiß
&40	00	13	Weiß				

Die Hardware-Farben des CPC

Bytes auf einmal aus dem Speicher liest (und den Pointer auch um 2 erhöht) und in die Bildinformation umwandelt. Dann vermindert er den Zeichenzähler um 1. Jetzt ist die Zeit für einen ersten Registervergleich gekommen. Ist der Stand des Zeichenzählers gleich dem Inhalt von
– Register 1 (Anzahl Zeichen/Zeile), so wird das Auslesen aus dem Video-RAM beendet und für den Rest der Zeile der BORDER gezeichnet;
– Register 2 (Beginn des horizontalen Rücklaufs), wird der Elektronenstrahl dunkelgeschaltet und vom rechten Bildschirmrand an den Anfang der nächsten Pixelzeile am linken Rand geführt. Hat er diesen erreicht, wird wieder die Farbe des BORDERS erzeugt;
– Register 0 (Gesamtbreite des Bildschirms inklusive BORDER und horizontalem Strahlrücklauf), so ist die Ausgabe einer Pixelzeile beendet, und der CRTC führt folgende Aktionen aus:
a) Der Zeichenzähler wird wieder auf 0 gesetzt.
b) Der Pixelzeilenzähler wird um 1 erhöht und mit dem Inhalt von Register 9 (Pixelzeilen –1 pro Textzeile) verglichen. Stimmen beide überein, wird der Pixelzeilenzähler mit 0 geladen, aber dafür der Textzeilenzähler um 1 erhöht.
c) Die Änderung des Pixelzeilenzählers wird im "Pointer" berücksichtigt.

Register werden abgefragt

Die gleichen Abfragen wie oben für den horizontalen Zeichenzähler nun auch für den vertikalen Textzeilenzähler. Ist der Wert des Textzeilenzählers gleich dem Inhalt von
– Register 6 (Anzahl der dargestellten Textzeilen (25)), wird das Auslesen aus dem Video-RAM für diesen Bildaufbau beendet und nur noch der BORDER angezeigt;
– Register 7 (Start des vertikalen Rücklaufs), so wird der Elektronenstrahl wieder dunkelgeschaltet und von rechts unten nach links oben quer über den Bildschirm zurückbewegt;
– Register 4 (Gesamthöhe des Bildschirms inklusive BORDER und vertikalem Strahlrücklauf), beschäftigt sich der CRTC wieder mit dem Zurücksetzen sämtlicher Register und dem Neuladen des Pointers. Die unendliche Geschichte hat begonnen ...
Zur Entspannung empfehlen wir, mit den gutmütigen Registern 1 und 6 zu experimentieren. Wer obiges Sezieren des CRTCs verstanden hat, sollte auch in der Lage sein, den Grund der verwirrenden Bildverschiebung bei Änderung von Register 1 zu erkennen.
Von den Registern des CRTC nun zum Gate-Array, das den CRTC mit den für den Bildaufbau nötigen Farbdaten versorgt. Wie der Name schon andeutet: ein wichtiges Tor zur Hardwarewelt des CPC. Insgesamt verfügt das Gate-Array über vier Register, die über beliebige Portadressen von &40 bis &7F erreichbar sind. Bei Werten unter &7E fühlt sich der Floppycontroller angesprochen, deshalb sollte man trotzdem nach dem Befehl OUTI ein "INC B" setzen. Im Gegensatz zum CRTC schickt man aber nicht den Index (Registerauswahl) und den eigentlichen Wert getrennt, sondern in einem einzigen Byte vereinigt. Bit 6 und 7 legen dabei das gewünschte der vier Register fest, Bit 0 bis 5 enthalten den zu sendenden Wert. Die einzelnen Funktionen sind aus der Abbildung der Gate-Array-Register ersichtlich.

Von Pixeln und Farben

Wir werden das Gate-Array zum schnellen Ändern von Bildschirmfarben verwenden, da die Systemroutinen (&BC32 und so weiter) für komplexe Farbeffekte absolut ungeeignet sind.
Um beispielsweise den BASIC-Befehl "INK 0,6" direkt über das Gate-Array auszuführen, benötigt man zwei OUT-Befehle.
Zuerst wird Register 0 mit der Pen-Nummer (= 0) geladen, dann Register 1 mit der Farbe (= 6).
Nun nehmen wir die letzte Hürde: Das Betriebssystem versetzt uns vorsätzlich in die Scheinwelt einer nach der Helligkeit am Grünmonitor geordneten Farbpalette. Die Hardware verwendet allerdings ein anderes Codierungssystem. Ein Blick in die Tabelle offenbart: Der Farbe 6 (Hellrot) entspricht die Nummer 12. Man schreibt also:

```
OUT &7F00,0
```

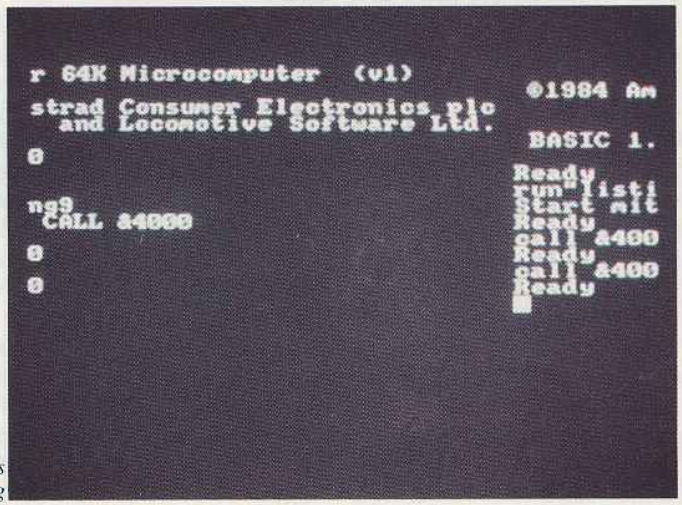
(Register 0, Wert 0)

```
OUT &7F00,&4C
```

(Register 1, das heißt Bit 6=1, Wert 12)
Ein kurzes Aufflackern von INK 0 bestätigt: Es hat funktioniert! Allerdings maximal nur 1/50 Sekunde, denn beim nächsten vertikalen Strahlrücklauf (VSR) setzt das Betriebssystem wieder die alten Farben. In Assembler geht man so vor:

```
DI ;Betriebssystem aus, Farbe
    bleibt
LD BC,&7F00
OUT (C),C
LD C,&4C
OUT (C),C ;Farbe setzen
LOOP: JP LOOP ;Endlosschleife
```

Zum Abschluß noch ein kurzer Blick auf den Portbaustein 8255, ähnlich wie das Gate-Array ein "Mädchen für alles" ist, ist er für Soundchip, Tastaturabfrage, Kassettenrekorder und Weiterleitung verschiedener Signale zuständig. Eine Übersicht der verwirrenden Funktionen würde diesen Rahmen sprengen, zumal wir nur an einem einzigen Bit interessiert sind: dem VSync-Signal. Der CRTC besitzt nämlich die angenehme Eigenschaft, bei jedem horizontalen und vertikalen Strahlrücklauf ein Synchronisationssignal auszugeben. Erste-



Screen-Splitting plus Hardware-Scrolling

res ließen die CPC-Entwickler schamlos auf der Platine versickern, das VSync-Signal (vertikale Synchronisation) wurde aber dankenswerterweise an besagten Baustein 8255 weitergeleitet, wo es sich ständig über Bit 0 der Portadresse &F5 abfragen läßt.

Das VSync-Signal am Parallelbaustein

Wegen der notwendigen Geschwindigkeit, bedingt durch die kurze Dauer des vertikalen Rücklaufs, funktioniert eine Abfrage nur in Assembler, unter BASIC verwendet man CALL &BD19 oder den Befehl FRAME (664/6128), die beide die nun folgende Routine im ROM aufrufen:

```
LD B,&F5          ;Portadresse laden
LOOP IN A,(C)     ;Wie OUT (C),A - nur INput statt OUTput über &F5
RRCA              ;Bit 0 des empfangenen Wertes ins Carry-Flag
JR NC,LOOP        ;War es 0, dann noch kein vertikaler Strahlrücklauf - weiter warten.
```

Und damit wäre sie auch endlich geschafft – die Theorie. Jetzt erwartet uns ein Sprung in die wogenden Gewässer der Praxis, was manchen rauchenden Köpfen vielleicht die ersehnte Abkühlung bringen wird.
Mittlerweile gibt es eine Vielzahl von "Special Effects", die wir hier unmöglich alle bis ins kleinste Detail durchleuchten können. Vielmehr sollte jeder, den der "Forscherdrang" gepackt hat, mit den nun folgenden Informationen gerüstet, in der Lage sein, die Feinheiten selbst zu entdecken.
Beginnen wir mit den Rasterbalken. Eine der ältesten Errungenschaften der neuzeitlichen Demoprogrammierung wurde bereits um 1987 entdeckt. Fast allen dürften die wandernden Farbbalken bekannt sein, die auch Mode-2-Programme in ungeahnter Farbenpracht erstrahlen lassen.
Der Name Raster geht auf den Begriff Rasterzeile zurück, was eigentlich Pixelzeile bedeutet. Die Idee: Während der Elektronenstrahl von oben nach unten über den Bildschirm rast, ändert man über das Gate-Array in dem Moment, wenn er von rechts nach links zurückgelenkt wird, ein bis zwei Farben. Im Speicher befindet sich dann eine Tabelle, die für jede Pixelzeile einen Farbwert enthält. Bei 200 Pixelzeilen wären so auch in Mode 2 200 Farben möglich. Der CPC kennt derer aber nur 27, womit die maximale Farbanzahl festgelegt ist.

Bedingt durch diverse Wartezyklen, sollten diese Schleifen rein rechnerisch trotzdem nur 62 ms (Millisekunden) dauern, was allerdings wieder von der Anzahl der OUT-Befehle abhängt. Es empfiehlt sich,mit Werten von 61 bis 64 zu experimentieren.

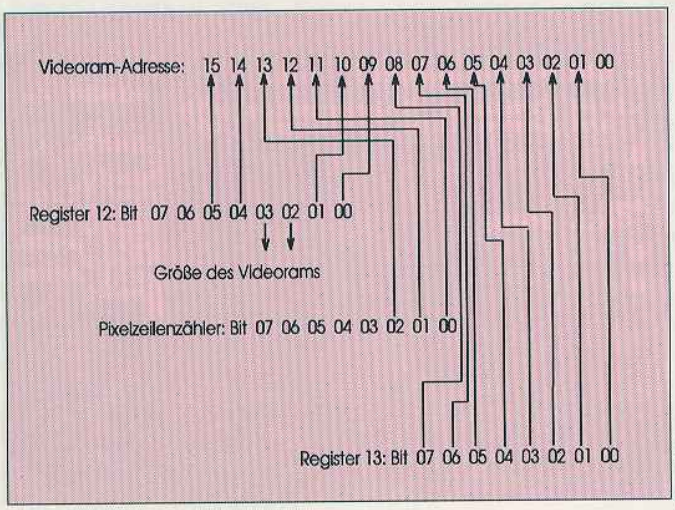
```
        LD HL,TABELLE   ;Farbdaten
        LD A,100        ;Anzahl Durchläufe
        LD BC,&7F00     ;Portadresse Gate Array
        OUT (C),C       ;PEN 0 anwählen
LOP:    LD C,(HL)       ;1.75 ms
        OUT (C),C       ;4.00 ms (durch Wartezyklen)
        INC HL          ;1.50 ms
        LD R,A          ;2.25 ms sinnlose Verzögerung
        DS 49           ;49.0 ms sinnlose Verzögerung (49 NOPs à 1 ms)
        DEC A           ;1.00 ms
        JP NZ,LOP       ;2.50 ms
                        62.00 ms

        RET TABELLE:    DB &54,&4B,&59,&5A,&53,.....(100x)
```

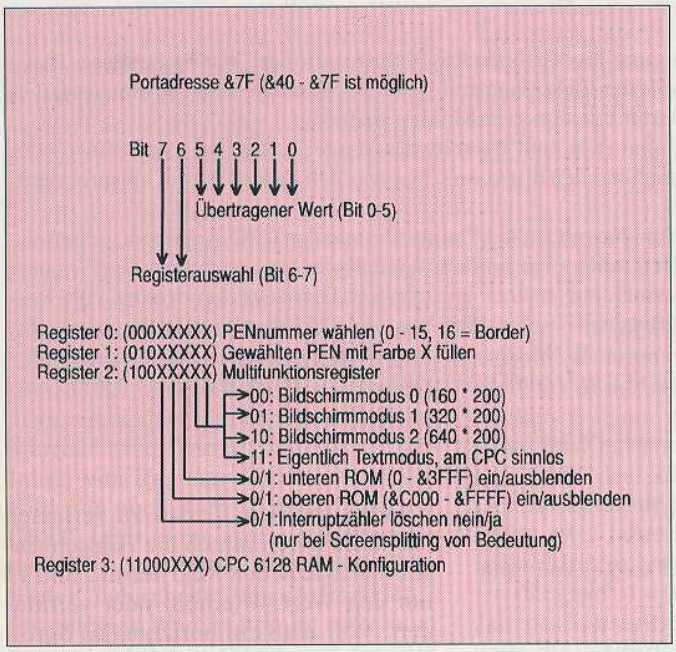
Beispiel für eine typische 64-Millisekunden-Schleife

Volle Farbenpracht auch in Mode 2

Das Besondere besteht nun darin, daß man in besagter Tabelle nach jedem Bildschirmaufbau mit dem Befehl "LDIR" Bereiche verschiebt, neu aufbaut und so weiter. Hier sind nur durch die Phantasie des Programmierers Grenzen gesetzt.
Das Problem: Wie erkennen wir, daß der Elektronenstrahl gerade nach links zur nächsten Pixelzeile zurückgleitet wird und daß es daher an der Zeit wäre, wieder eine Farbänderung vorzunehmen?
Das vom CRTC zu diesem Zwecke generierte HSync-Signal (horizontale Synchronisation) fällt aus oben nachzulesendem Grunde aus. Die Antwort: Man agiert blind und vertraut darauf, daß eine Pixelzeile (auch Horizontalzyklus genannt), wie in Register 0 festgelegt, 64 (0 bis 63) Zeichen (davon 24 Border) breit ist.
Der CRTC arbeitet mit einer Taktfrequenz von 1 MHz, ein Zyklus dauert also eine Mikrosekunde. Für den Aufbau einer Pixelzeile benötigt der CRTC 64 Taktzyklen (64 Mikrosekunden). Bedingt durch den Wortmodus liest er in dieser Zeit 128 Byte aus dem Bildschirmspeicher. Mit diesem Wissen ge-



Bildung der effektiven Video-RAM-Adresse (Pointer)



Die Gate-Array-Register

wappnet, geht man wie folgt vor:
1. Man wartet auf den vertikalen Strahlrücklauf (mit obiger Routine) oder auf einen Interrupt (mit HALT).
2. Warteschleife, um den Farbwechsel in die horizontaler Strahlrücklauf-Region zu lenken.
3. Man konstruiert eine Schleife, die für eine Farbänderung genau 64 Mikrosekunden benötigt, und führt sie zum Beispiel 100mal aus.
4. Jetzt wird die Farbdatentabelle verändert.
5. Zurück zu Punkt 1.

Einen typischen Vertreter einer 64-Millisekunden-Schleife finden Sie in der Abbildung. Ein Beispiel für die Rasterbalkentechnik finden Sie in Listing 4.

Direkter Nachfolger der Rasterbalken ist das Splitraster, aber mit einem kleinen Unterschied: Die Farbe wird nicht einmal, sondern mehrmals pro Pixel beziehungsweise Rasterzeile geändert. Beispielsweise lädt man die Register H, L, D, E, C und A mit beliebigen Farbwerten und gibt sie dann der Reihe nach aus, während sich der Elektronenstrahl im sichtbaren Bereich befindet:

```
OUT (C),H
OUT (C),L
OUT (C),D
```

und so weiter. Einziges Problem: Das VSync-Signal läßt sich nur noch unter großem Programmieraufwand zur Synchronisation verwenden, weil ein Durchlauf der Abfrageschleife über 7 ms dauert, wodurch man das Splitraster-Programm nur auf 7 ms genau mit dem Elektronenstrahl gleichschalten kann, was schließlich ein flackerndes Hin- und Herrutschen der Splitraster verursacht. Der Geistesblitz: Man bedient sich des CPC-Interruptsystems: Jede 300stel Sekunde (auf die für uns so wichtige Millisekunde genau) unterbricht die CPU ihre Arbeit und springt (im Interruptmodus 0) zur Adresse &38, wo normalerweise ein Sprung zum Interrupt-Handler des Betriebssystems steht. An besagte Adresse schreibt man jetzt eine einfache Rückkehranweisung (EI: RET) oder einen Sprung zur Splitraster-Routine. Das Hauptprogramm läßt man nun mit einem HALT-Befehl auf den Interrupt warten. Ein Beispiel finden Sie in Listing 5.

Nutzung des Interruptsystems

Ein zwar schon leicht angegrauter, aber oft sehr wirkungsvoller Trick: Man lädt das CRTC-Register 1 mit einem Wert größer als der in Register 0, meistens also &40. Dadurch muß der CRTC den Horizontalzyklus beenden, bevor er die geforderte Zeichenanzahl (64,&40) geschrieben hat. Die Folge: Der Pointer bleibt hängen, und der CRTC gibt immer wieder dieselbe Textzeile aus. Eine auf diesem Trick aufgebaute Laufschrift wartet in Listing 6. Ein Problem ergibt sich aber bei CRTC-Typ 2: Der Pointer wird eingefroren, er wird auch am Beginn des Vertikalzyklus nicht mehr neu geladen. Es ist also die Textzeile sichtbar,in der sich der Elektronenstrahl im Moment der Register-1-Manipulation befand. Dieser Umstand wird von Listing 6 berücksichtigt.

Dippel-, Trippel- oder Quadruppelmodus eignen sich gut für Leute, die wenig Bildschirmdaten schaufeln und trotzdem viel Bewegung sehen wollen. Die Lösung liegt im machtvollen Register 9: Lädt man es mit den Werten 15 (Dippel), 23 (Trippel) oder 31 (Quadruppel), so wird jede Textzeile zweimal, dreimal oder viermal hintereinander angezeigt. Der Pixelzeilenzähler durchläuft Werte bis 31. Es finden aber nur die untersten drei Bits Verwendung, dadurch kommt es zur Wiederholung. Da aber eine Textzeile jetzt zweimal, dreimal oder viermal so groß ist, muß man auch die Werte in den Registern 4, 6 und 7 halbieren, dritteln oder vierteln.

Dippelmodus:

```
OUT &BC00,4
OUT &BD00,19
OUT &BC00,6
OUT &BD00,12
OUT &BC00,7
OUT &BD00,15
OUT &BC00,9
OUT &BD00,15
```

Quadruppel:

```
OUT &BC00,4
OUT &BD00,9
OUT &BC00,6
OUT &BD00,6
OUT &BC00,7
OUT &BD00,8
OUT &BC00,9
OUT &BD00,31
```

Versuchen Sie doch nun einmal selbst, den Trippelmodus austüfteln.

Der Screen-Squasher läßt sich zum effektvollen Bildschirmauf- oder -abbau nutzen. Wie beim Rasterbalken wird hier in jeder Pixelzeile ein Register geändert, nur ist diesmal nicht das Gate-Array, sondern der CRTC das Opfer, genauer: Register 1. Es wird am Anfang einer Rasterzeile entweder mit 0 oder 40 geladen. Im ersten Fall wird nur Border gezeichnet und der Rest des Bildes nach unten gedrückt, im zweiten Fall findet die Darstellung wie gewohnt statt. Der Effekt selbst läßt sich schwer beschreiben, Listing 7 bringt ihn aber auf den Monitor.

Der Horizontal-Waggler, in deutschen Landen auch "Schwabbler" genannt, funktioniert nach demselben Prinzip wie der Screen-Squasher. Allerdings ist Register 2 das Ziel. Man verändert kon-

tinuierlich die Position des horizontalen Rücklaufs, der durch den Bildfang zwar an der gleichen Stelle bleibt, wodurch sich aber das Bild selbst verschiebt. Da der CRTC diesen Eingriff in den empfindlichen Horizontalzyklus schwer verdaut, kommt es zu einer weichen Bildverschiebung. Für Register 2 sind Werte von &2D bis &34 sinnvoll. CRTC-Typ-2-Besitzer müssen vorher Register 3 mit einer 8 laden, da er sonst Werte über &31 in Register 2 nicht verkraftet. Ein entsprechendes Beispiel hierfür ist der Effekt des Bildschirmverzerrers, wie er als 1-kByte-Programm im Ausgabe 2/3'92 abgedruckt war.

Der Effekt des Vertikal-Wagglers zählt zu den neuesten Errungenschaften und ließ auch abgebrühte Demoschreiber beim Erstkontakt erstaunen. Das Prinzip ist das gleiche wie bei den obigen Beispielen, diesmal wird aber Register 9 verändert. Je größer der Standardwert in Register 9 (normal 7) ist, desto unsauberer der Effekt. Listing 8 verwendet trotzdem den Wert 7. Die Routine kannn auch längst nicht alle Möglichkeiten ausschöpfen, die der Vertikal-Waggler bietet. Durch das ständige Abändern von Register 9 werden immer nur die ersten eins bis acht Pixelzeilen einer Textzeile angezeigt, bevor der CRTC zur nächsten Textzeile weitergeht. So entsteht der Staucheffekt, der aber nur bei reinen Grafiken besonders gut wirkt.

Tolle Effekte

Overscan ist einer der wenigen Effekte, der auch von Spieleprogrammierern verwendet wird. Man erinnere sich an die Titelbilder der französischen Firma Titus (Crazy Cars II, Wild Streets), die den ganzen Bildschirm mit Border einnehmen. Natürlich werden keine Pixel gedehnt, sondern der Bildausschnitt wird vergrößert.

Zuerst wird das Bild in die linke obere Ecke geschoben:

```
OUT &BC00,3
OUT &BD00,8 ;für CRTC-Typ-2-Kom-
patibilität,
OUT &BC00,2
OUT &BD00,&32 ;Bild nach links
schieben
OUT &BC00,7
OUT &BD00,&23 ;Bild nach oben
schieben
OUT &BC00,1
OUT &BD00,48 ;Bild in X-Richtung
dehnen (48 Zeichen)
OUT &BC00,6
OUT &BD00,34 ;Bild in Y-Richtung
dehnen (34 Zeilen)
```

Statt 40x25 mißt der Bildschirm jetzt also 48x34 Zeichen. Der Nachteil liegt auf der Hand: Da der Bildschirmspeicher nur 16 kByte groß ist, 48x34 Zeichen aber (48x34x16=26112) fast 26 kByte benötigen, erscheint ein beträchtlicher Teil des Bildes doppelt. Als unser Retter aus dem Sumpf der Ratlosigkeit tritt diesmal Register 12 auf.

Die Bits 2 und 3 sind für die Größe des Video-RAM verantwortlich, lädt man beide mit 1, adressiert der CRTC fortan 32 kByte Bildschirmspeicher. Man opfert also bereits die Hälfte des kostbaren Speicherplatzes:

```
OUT &BC00,12
OUT &BD00,&3C
```

Im unteren Bildbereich ist jetzt die Bank 0 (&0000-&3FFF) eingeblendet, man kann also sogar einem BASIC-Programm (ab &170) bei der Arbeit zuschauen. Zum guten Schluß reißen wir die zwei kompliziertesten, dafür aber auch effektvollsten Spezialeffekte Screen-Splitting und Hardware-Scrolling kurz an. Kein modernes Demo wäre ohne diese beiden Alleskönner denkbar. Eine bitgenaue Erklärung würde wohl weitere acht Seiten verschlingen, deshalb konzentrieren wir uns am besten auf die Grundzüge:

1. Hardware-Scrolling: Dieser Trick wird immer dann verwendet, wenn ein bestimmter Bildbereich verschoben werden muß, die Software-Befehle wie LDIR oder LDI aber einfach zu langsam sind. Auch das Betriebssystem bedient sich dieser Technik, um den Bildschirm vertikal zu verschieben, wenn man ihn mit dem Cursor zu verlassen sucht. Es wird einfach die Bildschirmstartadresse in den Registern 12 und 13 um den Wert 40 erhöht oder vermindert. Will man ein horizontales Scrolling erreichen, genügt ein einfaches Inkrementieren, zum Beispiel:

```
10 FOR A=0 to 255
20 CALL &BD19
30 OUT &BC00,13
40 OUT &BD00,A
50 NEXT
```

Programmiert man zum Beispiel eine Laufschrift, beschränkt sich die Arbeit der CPU auf Buchstabenschreiben am (weiterwandernden) rechten Rand. Durch Hardware-Scrolling verkompliziert sich allerdings auch die Berechnung von Video-RAM-Adressen: So folgt auf die Adresse &C7FF nicht &C800, sondern &C000. Wer das nicht glaubt, kann es überprüfen:
"MODE 2" eingeben, mit dem Cursor 28 Zeilen hinunterfahren, "POKE &C7FF, 255" tippen und die folgende Adresse mit POKE-Befehlen suchen. Im Zweifelsfall helfen für den Anfang die Systemroutinen (&BC20 bis &BC29). Der Nachteil des Hardware-Scrollings zeigt sich deutlich: Will man mehr als eine Laufschrift, beispielsweise noch ein Bild darüber, wird auch dieses gnadenlos mitgescrollt. Hier eilt uns

2. Screen-Splitting zu Hilfe. Der Trick: Man ändert während des Bildaufbaus die Video-RAM-Adresse. Nach dem vertikalen Strahlrücklauf blendet man zum Beispiel mit

```
OUT &BC00,12
OUT &BD00,&10
OUT &BC00,13
OUT &BD00,0
```

Begriffserklärung

CRTC ist eine Abkürzung für "Cathode Ray Tube Controller", das heißt, dieser Baustein ist für die Kontrolle des Elektronenstrahls, der das sichtbare Bild von hinten auf die Leuchtschicht der Bildröhre zeichnet, im Monitor zuständig.

Register des CRTC sind (wie zum Beispiel das H oder L Register der Z80A-CPU) nichts anderes als Speicherzellen direkt im Chip, was den Zugriff darauf entsprechend beschleunigt.

Lötbrücken sind Verbindungen auf der Platine, die von den CPC-Anbietern (zum Beispiel AMSTRAD in Deutschland) angebracht werden, um den entsprechenden Firmennamen in der Einschaltmeldung erscheinen zu lassen. Diese Brücken werden vom Betriebssystem im ROM über den Parallel-Port-Baustein 8255 abgefragt. Dadurch braucht nicht in jedem Land ein neuer ROM-Baustein eingesetzt zu werden. Je nach Kombination erscheinen dann ARNOLD, AMSTRAD, ORION, SCHNEIDER oder SAISHO am Monitor.

die Speicherbank 1 (&4000-&7FFF) ein, die das Bild enthält. Hat der Elektronenstrahl den Bildschirm zur Hälfte gezeichnet, lädt man die Adresse, die auf die Laufschrift zeigt, in die Register 12 und 13. Schön wäre es, ginge der CRTC jetzt gleich daran, die Laufschrift darzustellen. Doch die erscheint erst beim nächsten Bildaufbau. Der Grund: Der CRTC beachtet die Register 12 und 13 nur in dem Moment, wenn er daraus den Pointer neu lädt. Wie weiter oben beschrieben, geschieht das nur dann, wenn der interne Textzeilenzähler den Wert von Register 4 erreicht hat. Wenn man einfach den Inhalt von Register 4 halbiert, ist ein Bild nur noch halb so hoch, und der CRTC stellt derer zwei untereinander dar. Allerdings erreicht der Textzeilenzähler dadurch nie den in Register 7 festgelegten Wert, es kommt zu keinem vertikalen Strahlrücklauf, und das Bild läuft haltlos durch. Halbiert man jetzt aber auch noch Register 7, hat man es geschafft. Es gibt sogar zwei vertikale Strahlrückläufe, von denen einer den Bildschirm genau in der Mitte teilt.

Hier ist nun die Screen-Splitting-Routine in BASIC:

```
10 OUT &BC00,4
20 OUT &BD00,19
30 OUT &BC00,7
40 OUT &BD00,15
```

Listing 9 beinhaltet eine einfache Assembler-Routine, die den Bildschirm sechsmal teilt, die entsprechenden Bereiche scrollt und gleichzeitig die Vertikalstrahl-Rücklaufbalken verdrängt.

Ein Ende des Rüstungswettlaufs der Demoprogrammierer ist in Sicht. Galt es vor zwei Jahren noch als Kunst, den Bildschirm in jeder Textzeile zu splitten (Register 4 = 0), und war vor einem Jahr das Screen-Splitting in jeder Pixelzeile die letzte Errungenschaft (Register 4, 9 = 0), so schafft man heute bereits über drei Splits pro Pixelzeile, hat also endlich das horizontale Screen-Splitting erfunden. Doch hier zeigt sich: Die CPU verbraucht die ganze Rechenzeit, nur um die Video-RAM-Adresse jede Pixelzeile dreimal zu ändern, die dadurch erreichten Effekte sind auch wenig spektakulär, so daß man sich mit der Zeit mehr der Softwareseite zuwenden wird: Vektorgrafik und schnelle Sprite-Routinen (siehe ZAP'T'BALLS auf DATABOX 4/5'92) werden die Demos der Zukunft sein.

Die Assembler-Quelltexte zu den BASIC-Listings finden Sie auf der DATABOX zu dieser Ausgabe.

Elmar Krieger/jg

Literatur:
– Multiface-II-Benutzerhandblatt
– Programmierung der EGA- und VGA-Karten, Addison Wesley

```
10 'SPECIAL EFFECTS #1                                  [1143]
20 '* 50Hz/60Hz SWAP *                                  [1117]
25 '(c)1992 Elmar Krieger & CPC Internatio              [2144]
nal
30 INK 0,26:INK 1,0:BORDER 26:RESTORE                   [2728]
40 GOSUB 80:PRINT"50Hz Bildwiederholfreque              [17977]
nz- Betrachtet man das Bild nicht direkt,s
ondern aus denAugenwinkeln, so wird man vo
n erbarmungslosem Flimmern getroffen!":CAL
L &BB18
50 GOSUB 80:PRINT"60Hz Bildwiederholfreque              [11778]
nz- Das Flimmern hat sich gebessert...
            Gegebenenfalls den Bildfang
regulieren!":CALL &BB18
60 GOSUB 80:PRINT"70Hz - Das Maximum. Das               [10889]
Bild laesst sich allerdings nur auf Gruenm
onitoren      einfangen... (V-HOLD)":CALL
&BB18
70 GOTO 30                                              [340]
80 MODE 2:FOR a=4 TO 7:OUT &BC00,a:READ i:              [3204]
OUT &BD00,i:NEXT:RETURN
90 DATA &26,&00,&19,&1E:'50Hz                           [697]
100 DATA &1F,&06,&19,&1B:'60Hz                          [958]
110 DATA &1B,&02,&19,&19:'70Hz                          [1081]
```

```
10 'SPECIAL EFFECTS #2                                  [1138]
20 '****CRTC CHECK****                                  [249]
25 '(c)1992 Elmar Krieger & CPC Internatio              [2144]
nal
30 MODE 2:INK 0,0:INK 1,26:BORDER 0                     [3485]
40 PRINT"CRTC-Test aktiviert,";                         [3128]
50 c$(0)="SP=Typ 0":c$(1)="R=Typ 1":c$(2)=              [2605]
"P=Typ 2"
60 FOR a=&A000 TO &A066:READ a$:b=VAL("&"+              [2309]
a$)
70 POKE a,b:c=c+b:NEXT                                  [726]
80 IF c<>11215 THEN PRINT"-> Fehler in Dat              [4595]
azeilen!":END
90 CALL &A000                                           [637]
100 PRINT "CRTC 6845";c$(PEEK(&AF00));" ge              [4050]
funden."
110 END                                                 [110]
120 DATA F3,2A,38,00,22,60,A0,21,FB,C9                  [1719]
130 DATA 22,38,00,06,F5,ED,78,1F,D2,0F                  [1994]
140 DATA A0,ED,78,1F,D2,15,A0,FB,76,21                  [642]
```

```
150 DATA 4B,00,2B,7C,B5,C2,20,A0,ED,78                  [2340]
160 DATA 1F,DA,2F,A0,AF,18,2C,76,76,76                  [2012]
170 DATA F3,ED,78,1F,D2,33,A0,01,02,BC                  [1442]
180 DATA ED,49,01,32,BD,ED,49,FB,76,76                  [1238]
190 DATA 76,76,76,76,76,06,F5,ED,78,1F                  [2093]
200 DATA 3E,02,30,02,3E,01,01,2E,BD,ED                  [1464]
210 DATA 49,F3,32,00,AF,21,00,00,22,38                  [1331]
220 DATA 00,FB,C9                                       [468]
```

```
10 'SPECIAL EFFECTS#3                                   [1141]
20 'TESTBILD FUER 4-9                                   [667]
25 '(c)1992 Elmar Krieger & CPC Internatio              [2144]
nal
30 'RUN "LISTINGX"                                      [972]
40 'RUN "LISTING3"                                      [991]
50 MODE 2:INK 0,0:INK 1,20:BORDER 0:FOR A=              [15664]
0 TO 320 STEP 20:MOVE A,0:DRAW 640,400:NEX
T:FOR A=0 TO 16:MOVE 0,400-A^2.2:DRAW 640,
400-A^2.2:NEXT:FOR A=1 TO 25:LOCATE A*2,A:
PRINT"CPC AMSTRAD INTERNATIONAL";:NEXT
60 CALL &BB18:CALL &4000:GOTO 60                        [1583]
```

```
10 'SPECIAL EFFECTS #4                                  [1152]
20 '***RASTERBALKEN***                                  [1235]
25 '(c)1992 Elmar Krieger & CPC Internatio              [2144]
nal
30 MEMORY &2FFF:FOR A=&4000 TO &40B3                    [1702]
40 READ B$:B=VAL("&"+B$):POKE A,B                       [1468]
50 C=C+B:NEXT:IF C<>17483 THEN 70                       [2443]
60 PRINT"Start mit CALL &4000":END                      [2598]
70 PRINT"Fehler in Datazeilen":END                      [2574]
100 DATA F3,21,A6,40,11,00,30,D5,3E,0A                  [2042]
110 DATA E5,46,62,6B,70,13,01,19,00,ED                  [1297]
120 DATA B0,E1,23,3D,20,F0,E1,11,00,31                  [1519]
130 DATA 01,00,01,ED,B0,3E,C9,32,20,50                  [1600]
140 DATA 21,E8,03,E5,06,F5,ED,78,0F,30                  [1189]
150 DATA FB,21,95,01,2B,7C,B5,20,FB,21                  [1147]
160 DATA 00,30,3E,FA,01,00,7F,1E,10,56                  [1610]
170 DATA ED,49,ED,51,ED,59,ED,51,23,CD                  [870]
180 DATA 00,50,3D,C2,45,40,3E,54,ED,59                  [1882]
190 DATA ED,79,DD,21,B0,40,06,04,C5,DD                  [1572]
200 DATA 5E,00,16,31,1A,15,12,1C,DD,73                  [1656]
```

```
210 DATA 00,21,84,40,01,22,00,ED,B0,DD      [2022]
220 DATA 23,C1,10,E6,E1,2B,7C,B5,20,A9      [1316]
230 DATA FB,C9,44,44,55,44,55,55,57,55      [1258]
240 DATA 57,57,5F,57,5F,5F,4B,5F,4B,4B      [1692]
250 DATA 5F,4B,5F,5F,57,5F,57,57,55,57      [1985]
260 DATA 55,55,44,55,44,44,5C,4C,4E,4A      [1445]
270 DATA 43,4A,4E,4C,5C,54,00,40,80,C0      [1598]
```

```
10 'SPECIAL EFFECTS #5                      [1155]
20 '***SPLIT-RASTER***                      [1404]
25 '(c)1992 Elmar Krieger & CPC Internatio  [2144]
nal
30 MEMORY &2FFF:FOR A=&4000 TO &40C1        [1706]
40 READ B$:B=VAL("&"+B$):POKE A,B           [1468]
50 C=C+B:NEXT:IF C<>19206 THEN 70           [1473]
60 PRINT"Start mit CALL &4000":END          [2598]
70 PRINT"Fehler in Datazeilen":END          [2574]
100 DATA 21,9A,40,11,00,30,01,28,00,ED      [2086]
110 DATA B0,21,00,30,11,28,30,01,E8,03      [844]
120 DATA ED,B0,2A,38,00,22,94,40,21,FB      [1804]
130 DATA C9,22,38,00,21,E8,03,06,F5,ED      [2038]
140 DATA 78,0F,30,FB,76,E5,76,F3,06,07      [1889]
150 DATA 10,FE,00,00,ED,73,6D,40,31,00      [2169]
160 DATA 30,3E,AA,01,00,7F,ED,49,E1,D1      [1211]
170 DATA ED,61,ED,69,ED,51,ED,59,ED,61      [1268]
180 DATA ED,69,ED,51,ED,59,ED,61,ED,69      [907]
190 DATA FD,BE,00,FD,BE,00,00,00,00,00      [1976]
200 DATA 3D,C2,44,40,3E,54,ED,79,31,00      [1867]
210 DATA 00,FB,3E,06,F5,26,30,6F,4E,06      [1658]
220 DATA AA,54,5D,7D,C6,04,6F,D2,83,40      [1827]
230 DATA 24,7E,12,10,F2,71,F1,D6,02,20      [1487]
240 DATA E5,E1,2B,7C,B5,20,9A,21,00,00      [2087]
250 DATA 22,38,00,C9,44,5C,58,56,55,4C      [1743]
260 DATA 4D,52,57,4E,4F,59,5F,4A,4F,4A      [1778]
270 DATA 4B,43,4B,4B,5F,4A,4F,4A,57,4E      [1867]
280 DATA 4F,59,55,4C,4D,52,44,5C,58,56      [902]
290 DATA 54,54,54,54                        [631]
```

```
10 'SPECIAL EFFECTS#6                       [1150]
20 '**ONE LINE ONLY**                       [1023]
25 '(c)1992 Elmar Krieger & CPC Internatio  [2144]
nal
30 MEMORY &2FFF:FOR A=&4000 TO &4096        [1721]
40 READ B$:B=VAL("&"+B$):POKE A,B           [1468]
50 C=C+B:NEXT:IF C<>15900 THEN 70           [2039]
60 PRINT"Start mit CALL &4000":END          [2598]
70 PRINT"Fehler in Datazeilen":END          [2574]
100 DATA F3,3E,01,CD,0E,BC,01,02,BC,ED      [1729]
110 DATA 49,01,31,BD,ED,49,01,07,BC,ED      [588]
120 DATA 49,01,1C,BD,ED,49,01,00,7F,ED      [766]
130 DATA 49,0E,54,ED,49,21,E8,03,06,F5      [1784]
140 DATA ED,78,0F,30,FB,E5,21,01,C0,E5      [1535]
150 DATA 54,5D,1D,01,64,00,ED,B0,E1,7C      [2204]
160 DATA C6,08,67,30,F0,06,64,10,FE,01      [1475]
170 DATA 01,BC,ED,49,01,40,BD,ED,49,3E      [632]
180 DATA 01,EE,01,32,50,40,28,18,21,02      [1298]
190 DATA 09,CD,75,BB,3E,20,3C,32,5F,40      [1858]
200 DATA FE,7E,20,05,3E,20,32,5F,40,CD      [1461]
210 DATA 5A,BB,F3,E1,2B,7C,B5,20,AF,01      [2991]
220 DATA 01,BC,ED,49,01,28,BD,ED,49,01      [1567]
230 DATA 02,BC,ED,49,01,2F,BD,ED,49,01      [850]
240 DATA 07,BC,ED,49,01,1E,BD,ED,49,FB      [1762]
250 DATA C9                                 [276]
```

```
10 'SPECIAL EFFECTS#7                       [1153]
20 '*SCREEN SQUASHER*                       [584]
25 '(c)1992 Elmar Krieger & CPC Internatio  [2144]
nal
30 MEMORY &2FFF:FOR A=&4000 TO &40A8        [1703]
40 READ B$:B=VAL("&"+B$):POKE A,B           [1468]
50 C=C+B:NEXT:IF C<>16235 THEN 70           [1365]
60 PRINT"Start mit CALL &4000":END          [2598]
70 PRINT"Fehler in Datazeilen":END          [2574]
100 DATA F3,01,10,7F,ED,49,0E,4C,ED,49      [2066]
110 DATA 21,00,30,11,01,30,36,28,01,FA      [1697]
120 DATA 00,ED,B0,3E,C9,32,28,50,21,AB      [1788]
130 DATA 40,3E,C0,77,23,D6,08,30,FA,21      [1653]
```

```
140 DATA 01,05,22,A9,40,21,96,00,E5,06      [1167]
150 DATA F5,ED,78,0F,30,FB,21,8C,02,2B      [2360]
160 DATA 7C,B5,20,FB,3E,40,ED,79,01,01      [2140]
170 DATA BC,ED,49,04,21,00,30,16,D2,7E      [2085]
180 DATA ED,79,23,7E,7E,CD,00,50,15,C2      [1282]
190 DATA 4F,40,3E,28,ED,79,DD,21,AB,40      [1301]
200 DATA 3A,A9,40,47,C5,DD,6E,00,7D,FE      [1502]
210 DATA F0,28,1A,C6,08,DD,77,00,26,30      [1347]
220 DATA 54,5D,1C,36,00,01,07,00,ED,B0      [1832]
230 DATA 2C,1C,36,28,01,07,00,ED,B0,C1      [1077]
240 DATA DD,23,10,D8,21,AA,40,35,20,0A      [780]
250 DATA 36,05,2B,34,7E,FE,1A,20,01,35      [1635]
260 DATA E1,2B,7C,B5,C2,30,40,FB,C9         [1571]
```

```
10 'SPECIAL EFFECTS #8                      [1164]
20 '*VERTICAL WAGGLER*                      [949]
25 '(c)1992 Elmar Krieger & CPC Internatio  [2144]
nal
30 MEMORY &2FFF:FOR A=&4000 TO &40F4        [1689]
40 READ B$:B=VAL("&"+B$):POKE A,B           [1468]
50 C=C+B:NEXT:IF C<>22720 THEN 70           [1913]
60 PRINT"Start mit CALL &4000":END          [2598]
70 PRINT"Fehler in Datazeilen":END          [2574]
100 DATA F3,01,00,7F,ED,49,0E,54,ED,49      [1493]
110 DATA 21,00,30,E5,11,01,30,06,00,DD      [1293]
120 DATA 21,DC,40,3E,19,DD,4E,00,71,0C      [1102]
130 DATA ED,B0,DD,23,3D,20,F4,21,FF,09      [2296]
140 DATA 22,F5,40,3E,C9,32,28,50,3E,05      [1476]
150 DATA 32,F7,40,E1,11,00,31,01,FF,00      [1359]
160 DATA ED,B0,21,C8,00,E5,06,F5,ED,78      [2514]
170 DATA 0F,30,FB,01,10,7F,ED,49,0E,40      [1747]
180 DATA ED,49,21,8A,02,2B,7C,B5,20,FB      [1802]
190 DATA 01,04,BC,ED,49,01,65,BD,ED,49      [1633]
200 DATA 01,07,BC,ED,49,01,4E,BD,ED,49      [1870]
210 DATA 01,09,BC,ED,49,04,21,00,30,16      [1844]
220 DATA 4B,7E,ED,79,23,7E,7E,CD,00,50      [1195]
230 DATA 15,C2,79,40,0E,02,ED,49,21,F7      [1342]
240 DATA 40,35,C2,42,40,36,03,2A,F5,40      [1407]
250 DATA 7C,85,CC,D6,40,FE,0A,CC,D9,40      [1220]
260 DATA 67,22,F5,40,5F,16,30,21,09,31      [1925]
270 DATA 01,40,00,ED,B0,E1,2B,7C,B5,C2      [1725]
280 DATA 41,40,01,04,BC,ED,49,01,26,BD      [1742]
290 DATA ED,49,01,07,BC,ED,49,01,1E,BD      [1953]
300 DATA ED,49,01,09,BC,ED,49,01,07,BD      [1146]
310 DATA ED,49,FB,C9,2E,01,C9,2E,FF,C9      [1153]
320 DATA 00,00,00,00,00,00,00,00,00,00      [822]
330 DATA 01,02,03,04,05,06,07,06,05,04      [1592]
340 DATA 03,02,01,00,07                     [792]
```

```
10 'SPECIAL EFFECTS #9                      [1167]
20 '*SCREEN SPLITTING*                      [687]
25 '(c)1992 Elmar Krieger & CPC Internatio  [2144]
nal
30 MEMORY &2FFF:FOR A=&4000 TO &40BC        [1750]
40 READ B$:B=VAL("&"+B$):POKE A,B           [1468]
50 C=C+B:NEXT:IF C<>19309 THEN 70           [2168]
60 PRINT"Start mit CALL &4000":END          [2598]
70 PRINT"Fehler in Datazeilen":END          [2574]
100 DATA 01,00,7F,ED,49,0E,54,ED,49,2A      [1205]
110 DATA 38,00,22,9A,40,21,FB,C9,22,38      [1370]
120 DATA 00,21,E8,03,E5,06,F5,ED,78,0F      [1435]
130 DATA 30,FB,01,10,7F,ED,49,3E,40,D3      [1434]
140 DATA 7F,01,04,BC,ED,49,01,05,BD,ED      [1842]
150 DATA 49,01,07,BC,ED,49,01,06,BD,ED      [1771]
160 DATA 49,CD,A0,40,76,CD,A5,40,76,CD      [2094]
170 DATA A0,40,76,CD,A5,40,76,CD,A0,40      [2697]
180 DATA 76,CD,A5,40,76,01,04,BC,ED,49      [1750]
190 DATA 01,03,BD,ED,49,01,07,BC,ED,49      [1135]
200 DATA 01,03,BD,ED,49,2A,B8,40,23,CB      [959]
210 DATA 94,22,B8,40,2A,BA,40,11,FF,03      [1110]
220 DATA 19,CB,94,22,BA,40,E1,2B,7C,B5      [751]
230 DATA C2,18,40,01,04,BC,ED,49,01,26      [1222]
240 DATA BD,ED,49,01,07,BC,ED,49,01,1E      [890]
250 DATA BD,ED,49,21,00,00,22,38,00,C9      [1373]
260 DATA 2A,B8,40,18,03,2A,BA,40,01,0C      [1641]
270 DATA BC,ED,49,04,ED,61,05,0C,ED,49      [1737]
280 DATA 04,ED,69,C9,00,30,00,30,00         [1631]
```