Software-Blitter

Ein neues Wort ist in aller Munde, der Blitter. Der Amiga hat ihn, der ST bekommt ihn, aber was hat es eigentlich damit auf sich?

Es handelt sich um einen hochspezialisierten Chip, der nur zum Kopieren von Speicherbereichen konstruiert wurde. Besonders geeignet erweist sich ein Blitter zum Bewegen von Objekten auf einem hochauflösenden Bildschirm; daher auch der Name, der für „Block Image Transferer" steht. In der Assemblerecke wollen wir Ihnen ein Programm vorstellen, das die Arbeit eines Blitters simuliert und gleichzeitig über zwei neue Basic-Befehle verfügt.

Im Gegensatz zur Player Missile Grafik oder Sprites, die über einen separaten Speicher für die Bilddaten verfügen, werden Bobs (Blitter Objects) direkt in den Bildspeicher kopiert. Dies hat den Vorteil, dass Objekte in fast beliebiger Größe und Anzahl bewegt werden können. Die einzige Einschränkung liegt in der vorhandenen Rechenzeit, denn je mehr bzw. je größere Bobs bewegt werden, desto größer ist auch die Datenmenge. Und diese Flut kann gewaltig sein. Während zur Bewegung eines Objektes mit PM-Grafik im günstigsten Fall die Veränderung eines einzigen Registers genügt, müssen bei Bobs häufig mehrere hundert oder tausend Bytes bewegt werden. Einem echten Blitter fällt das jedoch nicht schwer. Er kann per DMA sehr schnell auf den Speicher zugreifen, d.h. praktisch in jedem Arbeitstakt ein Byte lesen oder schreiben. Der 6502 benötigt dazu drei bis sieben Taktzyklen.

GET und PUT

Da unser Atari bekanntlich keinen Blitter hat, müssen wir uns mit einer Software-Lösung behelfen. Damit auch etwas praktisch Verwertbares dabei herauskommt, werden wir dem Atari-Basic zwei neue Befehle (GET und PUT) hinzufügen. Sie kennen die beiden bereits? Irrtum, wer sich schon mit Basic-Dialekten für 16-Bit-Rechner befasst hat, weiß, dass diese Befehle einen ganz neuen Sinn bekommen haben.

Mit GET kann man einem hochauflösenden Bildschirm einen rechteckigen Bereich entnehmen und seine Daten in einem String ablegen. PUT erledigt das Gegenteil; die Daten aus dem String werden in den HiRes-Screen zurückgeschrieben. Allerdings kann man den Bildausschnitt (nichts anderes als ein Bob) nun an jede beliebige Stelle des Bildschirms kopieren.

Diese beiden Befehle eröffnen dem Programmierer eine ganze Fülle neuer Möglichkeiten. Sie können nicht nur Objekte über den Bildschirm bewegen, sondern auch bei Windows behilflich sein und Symbole jeder Art anzeigen und wieder löschen.

Moderne Benutzeroberflächen wie das GEM des Atari ST (oder Intuition beim Amiga) arbeiten vorwiegend mit Symbolen (Icons) statt Texten. Wenn es Ihnen Spaß macht, können Sie mit GET und PUT ähnliche Programme schreiben. Sie benötigen dazu mehrere Symbole, die am Bildschirm gezeichnet und mit GET-Befehlen in Strings abgelegt werden. Durch PUT können diese Icons wieder sichtbar gemacht werden und so z.B. als Menü dienen. Auch Windows kann man mit unserem „Software-Blitter" verwalten. Bevor ein Window Teile des Bildes überschreibt, wird der Ausschnitt durch GET in einem String zwischengespeichert . Später lässt sich dieser per PUT an die alte Stelle zurückschreiben.

Zur Anwendung

Zunächst sollten Sie sich das Listing 1 näher ansehen. Es enthält im ersten Teil mehrere Demos mit Beispielen zu GET und PUT. Der zweite Teil (ab Zeile 30000) enthält das Maschinenprogramm des „Software-Blitter" als Data-Zeilen. Sollten Sie GET und PUT in eigenen Programmen einsetzen, brauchen Sie nur diesen Teil zu übernehmen, der das Maschinenprogramm in Page 6 ablegt. Die neuen Befehle können dann über USR angesprochen werden. Sinnvoll ist es auch, die Einsprungadressen des in Page 6 liegenden Programms wie im Beispiel als Variablen GGET=1536 (für Grafik-GET) und GPUT=1539 zu definieren. Ein GET-Befehl wird dann so ausgeführt :

A=USR(GGET,X1,Y1,X2,Y2,ADR (A$))

Die Koordinaten X1, Y1 geben die linke obere Ecke des Bildausschnitts an, X2 und Y2 die rechte untere . Als letzter Parameter muss die Adresse eines Strings eingesetzt werden (natürlich kann man statt A$ beliebige Namen einsetzen). Achten Sie nur darauf, dass in der DIM-Anweisung des Strings ausreichend Speicherplatz reserviert wurde. Den Platzbedarf können Sie wie folgt berechnen:

Bytes = (X2-X1+1) * (Y2-Y1+1)+2

Der PUT-Befehl wird ganz ähnlich aufgerufen (GPUT=1539):

A=USR (GPUT,X,Y,MOD,ADR(A$))

Bei PUT müssen Sie nur die linke obere Ecke angeben, da Höhe und Breite des Ausschnitts im String bereits enthalten sind. Der Parameter MOD bestimmt, wie PUT den Ausschnitt einsetzt. Dies geschieht durch folgende Codes:

0: OR-Modus. Der Ausschnitt wird dem bereits vorhandenen Bild durch eine Oder-Verknüpfung überlagert, z.B. ein Bob mit dem Hintergrund.

64:EXOR-Modus. Der Ausschnitt wird Exklusiv-Oder mit dem vorhandenen Bild verknüpft. Schreibt man den Ausschnitt ein zweites Mal an die gleiche Stelle, so verschwindet er wieder. Bobs können damit zerstörungsfrei über den Hintergrund bewegt werden.

128: PSET-Modus. Hier wird der Bildteil vom Ausschnitt einfach überschrieben. Diese Methode eignet sich zum Zurückkopieren von Hintergrundinformationen (s. Beispiel).

Noch ein Wort zu den Koordinaten. Der „Software-Blitter" ist auf GRAPHICS 8 zugeschnitten. Y kann daher Werte von 0 bis 191 annehmen, während für X anstatt 0 bis 319 nur der Bereich von 0 bis 39 zulässig ist. Diese Einschränkung ergibt sich aus dem Aufbau des Video-Speichers bei hochauflösender Grafik. Je acht Bildpunkte sind in einem Byte zusammengefasst, und dies ist die kleinste Einheit, die unser "Blitter" verarbeiten kann. Die gewohnten Koordinaten von 0 bis 319 müssen daher für GET und PUT jeweils durch acht geteilt werden.

Man könnte den „Software-Blitter" natürlich auch so auslegen, dass er die Byte-Grenzen ignoriert und alle 320 möglichen X-Koordinaten zulässt. Tatsächlich war die erste Version des Maschinenprogramms so ausgelegt , aber das Ergebnis war fatal. Das Programm erreichte nur noch ein Schneckentempo da nun zusätzlich die Bit-Positionen innerhalb der Bytes verschoben und eine Menge von Randbedingungen beachtet werden mussten. Es war für jegliche Bewegungseffekte viel zu langsam.

Hier hat nun ein Hardware-Blitter die Nase weit vorne, da er über sogenannte Barrel-Shifters verfügt. Stellen Sie sich dazu bitte folgendes vor: Wenn ein Byte um vier Bitstellen verschoben werden soll, muss der Prozessor vier Schiebebefehle (z.B. ROL) nacheinander ausführen. Ein Barrel-Shifter liefert das gleiche Ergebnis in einem einzigen Arbeitstakt, eine Zeitspanne, in der der Prozessor nicht einmal einen einzigen Befehl ausführen könnte.

Trotz dieser Einschränkung ist der „Soft-Blitter" sehr nützlich. Wenn Sie Listing 1 eingetippt und die Demos angeschaut haben, werden Ihnen bestimmt eine Menge neuer Einsatzgebiete einfallen.

Bewegung

Im Demo wird auch Bewegung mit dem "Soft-Blitter" erzeugt. Das Ergebnis ist zwar nicht ganz so ruck- und flimmerfrei, wie man es sich wünschen würde, aber immerhin wird ein 40 mal 30 Punkte großes Objekt (ein Flugzeug) zerstörungsfrei über den Hintergrund bewegt - und das in Basic !

Zu Beginn jedes Bewegungsschritts wird ein Ausschnitt des Hintergrunds in den String HG$ gerettet. Anschließend wird das Flugzeug aus GR$ mit einem PUT-Befehl (im Oder-Modus) dem Bild an dieser Stelle überlagert. Nun beginnt eine kleine Warteschleife, die dafür sorgt, dass das Flugzeug eine Weile sichtbar ist. Der in HG$ gespeicherte Hintergrund überschreibt dann mit einem erneuten PUT-Befehl (diesmal im PSET-Modus) das Bild mit dem zuvor geretteten Hintergrund. Damit ist der ursprüngliche Zustand des Bildes wiederhergestellt. Im nächsten Schleifendurchlauf kann das Flugzeug ein Stück entfernt neu gezeichnet werden. Da die Zeitspanne vom Löschen bis zum erneuten Erscheinen dank dem "Soft-Blitter" sehr kurz ist, entsteht der Eindruck einer Bewegung.

Wesentlich einfacher wäre die Bewegung mit dem EXOR-Modus zu erzielen, da hier der Buffer für den Hintergrund entfällt. Allerdings wird bei dieser Methode das Objekt ziemlich unansehnlich, wenn der Hintergrund dicht beschrieben ist. Probieren Sie es aber ruhig mal aus!

Auch das lästige Flackern des Objekts könnte man beseitigen. Man müsste nur einen zweiten GRAPHICS-8-Bildschirm anlegen und dann per Page-Flipping zwischen beiden hin- und herschalten. Während der "Blitter" im einen Bild arbeitet, wird das andere angezeigt, und sobald ein neues fertiggestellt ist, werden die Rollen getauscht. Diese Technik ist sehr wirkungsvoll und wäre vielleicht ein Thema für eine zukünftige Assemblerecke.

Blitter intern

Wie immer werfen wir auch diesmal einen Blick hinter die Kulissen des Maschinenprogramms (Listing 2). Nach einigen Definitionen von Variablen, die zur Beschleunigung alle in die Zero-Page gelegt wurden, findet sich eine kleine Sprungtabelle zu den Routinen GET und PUT.

Die GET-Routine beginnt ihre Arbeit, indem sie alle Parameter vom Stack nimmt und in die zugehörigen Variablen einträgt. Danach wird die Größe des Ausschnitts berechnet und gleich geprüft, ob die Koordinaten richtig eingegeben wurden, d.h. der linke Rand nicht größer als der rechte ist . Die berechneten Größen werden in den String als erstes und zweites Byte eingetragen, damit die PUT-Routine später weiß, wie groß der Ausschnitt ist.

Im Unterprogramm BERZEIG (berechne Zeiger) wird die Anfangsadresse der Zeile im Video-Speicher berechnet, wo der Ausschnitt beginnt. Mit Hilfe dieses Zeigers kann die folgende Schleife (ab Label GETZEIL) Byte für Byte aus dem Video-Speicher entnommen und per Unterprogramm PUTSTR in den gewünschten String eingetragen werden.

Entscheidend ist, dass man die Bytes aus dem Video-Speicher nicht kontinuierlich entnehmen darf. Sehen Sie sich dazu Bild 1 an, das einen Teil des Video-Speichers darstellen soll. Beginnend am linken Rand des gewählten Ausschnitts müssen soviel Bytes entnommen werden, bis der rechte Rand angetroffen wird. Nun muss ein Sprung zum linken Rand der nächsten Zeile erfolgen. Die Größe der Lücke hängt somit von der Breite des Ausschnitts und der Anzahl der Bytes pro Zeile ab. Man nennt diese Lücke gewöhnlich das Modulo.

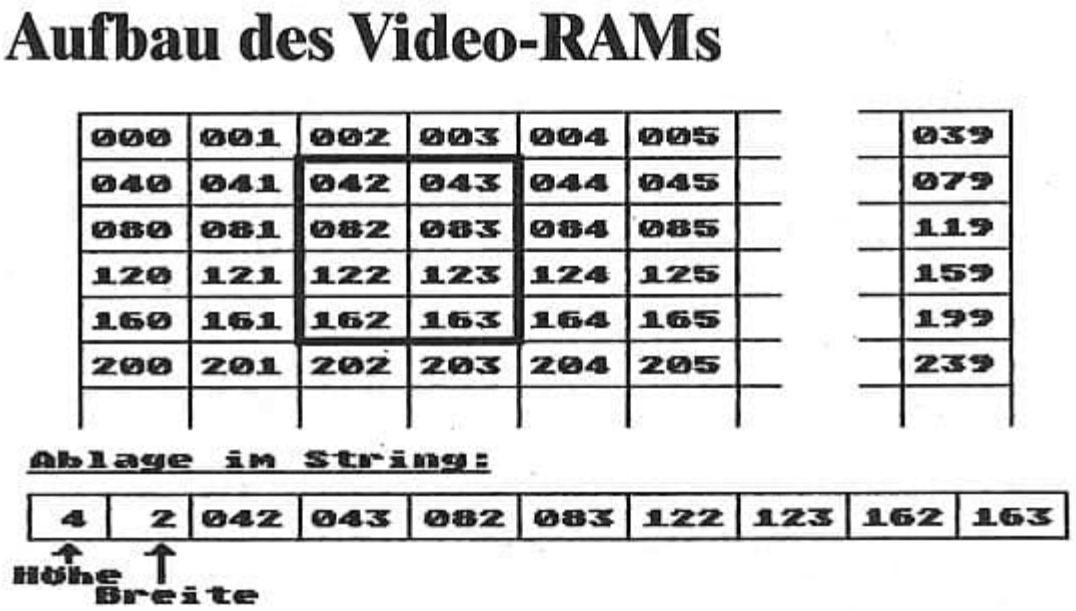

Bild1

Diese Situation wäre eine echte Anwendung für einen Hardware-Blitter. Ein solcher Chip hat Register für die Breite des Ausschnitts und das Modulo, und zwar sowohl für die Quelle der Daten (in unserem Fall das Video-RAM) als auch für das Ziel (hier der Basic-String). Da die Daten im String kontinuierlich abgelegt werden, wäre im vorliegenden Fall das Modulo eben null. Sie sehen, der Name ”Soft-Blitter” ist nicht aus dem Ärmel geschüttelt; die vorliegende Routine imitiert tatsächlich die Funktion eines Hardware-Blitter, wenn auch in stark vereinfachter Form.

In ihrem Aufbau entspricht die PUT-Routine der gerade besprochenen GET-Funktion, nur ist die Richtung des Datenflusses umgekehrt. Die Quelle (String) besitzt nun einen Modulo von null, das Ziel (Video-RAM) einen Modulo von Zeilenlänge minus Breite des Ausschnitts. Die Modulo-Werte sind in beiden Routinen fest einprogrammiert und können daher nicht frei gewählt werden. Durch diese Maßnahme wird das Programm kürzer und schneller.

Interessant ist vielleicht noch die Abfrage des Modus durch den BIT-Befehl. Dies ist eine kritische Stelle im Programm. Je länger die Abfrage dauert, desto langsamer wird das Programm, da sie für jedes bearbeitete Byte erfolgen muss. Ein Befehl mehr oder weniger kann sich dabei ganz hübsch auswirken. Die beste Lösung wäre sicherlich, für jeden Modus eine eigene ”Blitter”-Routine zu schreiben. Nur leider wird dann das Programm für Page 6 viel zu lang.

Ein wenig Aufmerksamkeit verdient auch die Berechnung der Zeilenadresse im Unterprogramm BERZEIG. Es handelt sich nicht um eine allgemeine Routine zur Multiplikation. Vielmehr werden hier trickreich Bit- Verschiebungen sowie Additionen eingesetzt, um eine Multiplikation mit 40 zu erreichen. Aufmerksame Leser der Assemblerecke kennen dieses Prinzip bereits aus früheren Folgen. Warum nun gerade mal 40? Ganz einfach, das ist die Anzahl der Bytes pro Zeile in GRAPHICS 8.

Andere Grafikstufen

Obwohl der ”Soft-Blitter” nur für den Einsatz mit GRAPHICS 8 konzipiert ist, kann man ihn auch mit einigen anderen Grafikstufen verwenden. Denkbar wären z.B. GRAPHICS 15 oder auch GRAPHICS 0. Im letzteren Fall könnte man recht effektiv mit Windows arbeiten, denn dem ”Blitter” ist es egal, ob die bearbeiteten Daten aus einem Text- oder HiRes-Bildschirm stammen. Aufpassen muss man jedoch mit den Y-Koordinaten, die dann nur noch von 0 bis 23 zulässig sind (es erfolgt keine Prüfung !) ; außerdem fänden die Modi OR bzw. EXOR des PUT-Befehls keine vernünftige Verwendung mehr. Ähnliche Einschränkungen ergeben sich mit den Farbgrafik-Stufen wie 7 oder 15. Da nun zwei nebeneinander liegende Bits die Farbinformation bestimmen, kommt es beim OR- bzw. EXOR-Modus zur Verfälschung der Farben. GET und PUT im PSET-Modus bleiben jedoch durchaus brauchbar. Es lohnt sich bestimmt, damit zu experimentieren.

Peter Finzel

```
100 REM ******************************
110 REM BLITTER-OBJECTS-(BOBS)-DEMO
120 REM
130 REM P. FINZEL 1986
140 REM ******************************
150 REM
200 DIM GR$(300),HG$(300)
210 GGET=1536:GPUT=1539
220 GRAPHICS 2+16
230 POSITION 2,4:PRINT #6;"SOFT-BLITTER DEMO"
240 POSITION 1,6:PRINT #6;"EINEN MOMENT BITTE"
250 GOSUB 30000
300 REM * Zeichne Flugzeug
305 GRAPHICS 8:COLOR 1
310 PLOT 55,20:DRAWTO 63,0:POSITION 47,12:GOSUB 700
320 DRAWTO 15,12:POSITION 0,20:GOSUB 700
330 PLOT 43,30:DRAWTO 38,20:PLOT 25,20:POSITION 35,30:GOSUB 700
335 REM * Bild in String
340 X=USR(GGET,0,0,8,30,ADR(GR$))
390 REM * viele Flugzeuge...
400 GRAPHICS 8+16: SETCOLOR 2,1,0
410 FOR X=0 TO 32 STEP 8
420 FOR Y=0 TO 140 STEP 35
430 A=USR(GPUT,X,Y,0,ADR(GR$))
440 NEXT Y:NEXT X
450 FOR Y=0 TO 140 STEP 35
460 FOR X=0 TO 32 STEP 8:SOUND 0,Y,10,10
470 A=USR(GPUT,X,Y,64,ADR(GR$))
480 SOUND 0,0,0,0:FOR I=0 TO 7:GOSUB 800:NEXT I
490 NEXT X:NEXT Y
500 REM * BEWEGUNG
510 M=69:GOSUB 900:SOUND 0,15,0,10
515 Y=100:FOR X=32 TO 0 STEP -1:Y=Y-2
520 A=USR(GGET,X,Y,X+7,Y+30,ADR (HG$))
530 A=USR(GPUT,X,Y,0,ADR(GR$))
540 FOR I=0 TO 4:GOSUB 800:NEXT I
560 A=USR(GPUT,X,Y,128,ADR(HG$))
570 NEXT X:SOUND 0,0,0,0
590 GOTO 400
700 REM * XIO-FILLROUTINE
710 POKE 765,1:XIO 18,#6,0,0,"S:"
790 RETURN
800 REM * 1/50 SEC. WARTEN
810 T=PEEK(20)
820 IF PEEK(20)=T THEN 820
830 RETURN
900 REM * BILD AUFBAUEN
910 GRAPHICS 8:SETCOLOR 2,8,0
920 FOR I=0 TO 319 STEP 31:PLOT 159,M:DRAWTO I,M+90:NEXT I
930 I=0
940 Q=I*I+M:IF Q<=159 THEN PLOT 0,Q:DRAWTO 319,Q:I=I+1:GOTO 940
990 RETURN
30000 REM * Soft-Blitter einrichten
30010 S=0:RESTORE 30100
30020 FOR A=1536 TO 1785:READ D:POKE A,D:S=S+D:NEXT A
30030 IF S<>34981 THEN ? "DATEN-FEHLER!":STOP
30090 RETURN
30100 DATA 76,6,6,76,98,6,216,104,104,104,133,218,104,104,133,217
30110 DATA 104,104,133,220,104,104,133,219,104,133,215,104,133,214
30120 DATA 56,165,220,229,218,176,1,96,133,221,230,221,56,165,219
30130 DATA 229,217,176,1,96,24,105,1,133,219,32,239,6,165,221,32,239
30140 DATA 6,32,175,6,164,218,165,221,133,216,177,212,32,239,6,200
30150 DATA 198,216,208,246,24,165,212,105,40,133,212,144,2,230,213
30160 DATA 198,219,208,225,96,216,104,104,104,133,218,104,104,133
30170 DATA 217,104,104,133,222,104,133,215,104,133,214,32,228,6,133
30180 DATA 219,32,228,6,133,221,32,175,6,165,221,133,216,164,218,32
```

```
30190 DATA 228,6,36,222,48,8,112,4,17,212,80,2,81,212,145,212,200
30200 DATA 198,216,208,234,24,165,212,105,40,133,212,144,2,230,213
30210 DATA 198,219,208,213,96,165,217,133,212,169,0,133,213,6,212
30220 DATA 38,213,6,212,38,213,24,165,212,101,217,133,212,144,2,230
30230 DATA 213,6,212,38,213,6,212,38,213,6,212,38,213,24,165,212,101
30240 DATA 88,133,212,165,213,101,89,133,213,96,162,0,161,214,230
30250 DATA 214,208,2,230,215,96,162,0,129,214,230,214,208,2,230,215,96
```

```
**************************************
*          SOFTWARE-BLITTER    V1.02
*
*        PUT/GET fuer Graphik
*         (Byte-Aufloesung)
*
*  Peter Finzel                 1996
**************************************
*
ZLAENGE EQU 40          Zeilenlaenge in BYTE
*
SAVMSC  EQU $58         Zeiger auf Video-RAM
*
* Variable in der Zero-Page
*
ZEIGER  EQU $D4         Pointer in Screen
STRING  EQU $D6         Pointer in String
BZAEHL  EQU $D8         Zaehler
ORAND   EQU $D9         Koordinaten: oberer Rand
LRAND   EQU $DA         linker Rand
URAND   EQU $DB         unterer Rand
RRAND   EQU $DC         rechter Rand
BLAENGE EQU $DD         Laenge fuer Blitter
MODUS   EQU $DE         Modus fuer PUT 0: Oder
*                       64:Exor      128: Pset
*
        ORG $0600
*
* Sprungleiste
*
        JMP GET
        JMP PUT
*
* GET-Routine fuer Bildausschnitte
* Aufruf:
* A=USR(1536,X1,Y1,X2,Y2,ADR(A$))
*
GET     CLD
        PLA             Anz. der Parameter
        PLA             Linker Rand MSB
        PLA             Linker Rand LSB
        STA LRAND
        PLA             Oberer Rand MSB
        PLA             Oberer Rand LSB
        STA ORAND
        PLA             Rechter Rand MSB
        PLA             Rechter Rand LSB
        STA RRAND
        PLA             Unterer Rand MSB
        PLA             Unterer Rand LSB
        STA URAND
        PLA             Adresse des Strings
        STA STRING+1
        PLA
        STA STRING
*
        SEC             Anz. der horizontalen
        LDA RRAND       Bytes ermitteln
        SBC LRAND
        BCS LENOK       ueberhaupt moeglich?
        RTS             Linker > Rechter!
*
LENOK   STA BLAENGE     Laenge merken
        INC BLAENGE     Laenge= R-L+1
        SEC             Anzahl der Zeilen
        LDA URAND       ermitteln
        SBC ORAND
        BCS HOEHOK      moeglich?
        RTS             Oben > Unten!
*
HOEHOK  CLC             Zeilenanzahl
        ADC #1          korrigieren
```

```
         STA URAND        und in URAND eintragen
         JSR PUTSTR       Hoehe und Breite des
         LDA BLAENGE      Ausschnittes im
         JSR PUTSTR       String eintragen
*
MSB1     JSR BERZEIG      Zeiger in Video-Ram
*
* Soft-Blitter fuer GET
*
GETZEIL  LDY LRAND        linker Rand
         LDA BLAENGE      Byte-Zaehler
         STA BZAEHL       einrichten
GETBYTE  LDA (ZEIGER),Y   Wert aus Video/RAM
         JSR PUTSTR       in String ablegen
         INY              naechstes Byte
         DEC BZAEHL       schon ganze Zeile
         BNE GETBYTE      nein ->
         CLC              Zeiger auf naechste
         LDA ZEIGER       Zeile im Video-RAM
         ADC #ZLAENGE     richten (plus 40)
         STA ZEIGER
         BCC GET2 jetzt MSB
         INC ZEIGER+1
*
GET2     DEC URAND        schon alle Zeilen?
         BNE GETZEIL      nein -->
         RTS
*
* PUT-ROUTINE fuer Bildausschnitte
* Aufruf:
* A=USR(1539,X,Y,Mod,Adr(A$))
*
PUT      CLD
         PLA              Anz. der Parameter
         PLA              Linker Rand MSB
         PLA              Linker Rand LSB
         STA LRAND
         PLA              Oberer Rand MSB
         PLA              Oberer Rand LSB
         STA ORAND
         PLA              Modus-Byte MSB
         PLA              Modus-Byte LSB
         STA MODUS
         PLA              Adresse des Strings
         STA STRING+1
         PLA
         STA STRING
*
         JSR GETSTR       Anzahl d. Zeilen
         STA URAND        des Ausschnittes
         JSR GETSTR       Hor. Laenge des
         STA BLAENGE      Ausschnittes
*
ERHZ2    JSR BERZEIG      Zeiger in Video-RAM
*
* Soft-Blitter fuer PUT
*
PUTZEIL  LDA BLAENGE
         STA BZAEHL       Zaehler Zeilenlaenge
         LDY LRAND        linker Rand
PUTBYTE  JSR GETSTR       Zeichen aus String
         BIT MODUS        welcher Modus?
         BMI PSET-> 'PSET'
         BVS EXOR> Ex-Oder
         ORA (ZEIGER),Y   'Oder'
         BVC PSETverzweigt immer
EXOR     EOR (ZEIGER),Y   Exor-Verkn.
PSET     STA (ZEIGER),Y   in Video-RAM
         INY              naechst. Spalte
         DEC BZAEHL       Zeile fertig?
```

```
         BNE PUTBYTE      nein - >
         CLC              naechste Zeile
         LDA ZEIGER       Zeiger=Zeiger+40
         ADC #ZLAENGE
         STA ZEIGER
         BCC PUT2
         INC ZEIGER+1     MSB
*
PUT2     DEC URAND        alle Zeilen fertig?
         BNE PUTZEIL      nein - >
         RTS
*
* Zeiger auf erstes Byte berechnen
*         ZEIGER = ORAND * 40
*
BERZEIG  LDA ORAND        LSB Zeiger=ORAND
         STA ZEIGER
         LDA #0           MSB Zeiger=0
         STA ZEIGER+1
         ASL ZEIGER       Zeiger=Zeiger*2
         ROL ZEIGER+1
         ASL ZEIGER       Zeiger=Zeiger*2
         ROL ZEIGER+1     ergibt ORAND*4
         CLC
         LDA ZEIGER       Zeiger=Zeiger+ORAND
         ADC ORAND        bisher berechnet:
         STA ZEIGER       Zeiger=ORAND*5
         BCC SCRADR1
         INC ZEIGER+1
SCRADR1  ASL ZEIGER       Zeiger=Zeiger*2
         ROL ZEIGER+1
         ASL ZEIGER       Zeiger=Zeiger*2
         ROL ZEIGER+1     ergibt ORAND*4
         ASL ZEIGER       Zeiger=Zeiger*2
         ROL ZEIGER+1     damit:
         CLC              ZEIGER=ORAND*40
         LDA ZEIGER       nun noch die
         ADC SAVMSC       Basisadresse des
         STA ZEIGER       Video-Rams addieren
         LDA ZEIGER+1
         ADC SAVMSC+1
         STA ZEIGER+1
         RTS
*
* Routine zur String-Verwaltung
*
GETSTR   LDX #0           Ein Byte aus
         LDA (STRING,X)   dem String holen
         INC STRING       und Stringzeiger
         BNE GS1          weiterschalten
         INC STRING+1
GS1      RTS
*
PUTSTR   LDX #0           Ein Byte in
         STA (STRING,X)   String eintragen
         INC STRING       und Stringzeiger
         BNE PS1          weiterschalten
         INC STRING+1
PS1      RTS
```