64'er Extra 6

Das 64'er Extra bringt geballte Information über Ihren C 64 zum Heraustrennen und Sammeln.

In dieser sechsten Ausgabe finden Sie den zweiten Teil einer Übersicht über alle ROM-Routinen des C 64. Statt ziellos in ROM-Listings zu blättern, finden Sie hier im Klartext die Funktionsbeschreibung aller irgendwie nutzbaren Routinen.

POS $B39E
*** POS: Ruft die Kernal-Routine PLOT auf, um sich die Cursorposition zu verschaffen, und lädt sie dann in FAC1 mittels:
SNGET $B3A2
Wandelt das Byte in Y in FLPT in FAC1 um (0...255).
ERRDIP $B3A6
Prüft, ob der Befehl nicht im Direktmodus eingegeben wurde; ein Wert von $FF in CURLIN+3 ($3A) zeigt Direktmodus an. Ist das der Fall, erfolgt die Meldung ?ILLEGAL DIRECT ERROR. Wird von Routinen aufgerufen, die nicht im Direktmodus zu verwenden sind, wie zum Beispiel GET.
DEF $B3B3
*** DEF: Erzeugt Funktionsdefinition; sucht die Funktionsvariable oder stellt sie auf. Ein Aufruf von FN setzt den Zeiger innerhalb CHRGET auf den Anfang der FN-Definition im Basic-Text, und der dort vorgefundene Ausdruck wird ausgewertet; anschließend wird der Zeiger wieder zurückgestellt. Die dafür notwendige Information ist mit der in GETFNM aufgestellten Funktionsvariablen gespeichert.
GETFNM $B3E1
Prüft die Syntax von FN; sucht oder stellt Variable mit dem Funktionsnamen auf und läßt (DEFPNT) ($4E) darauf zeigen (muß numerisch sein, keine Zeichenkettenvariable).
FNDOER $B3F4
Wertet Funktion aus: Berechnet den Klammerausdruck in der Anweisung mit dem Funktionsaufruf und legt das Ergebnis in FAC1 ab; anschließend erfolgt die Auswertung des Funktionsausdrucks (siehe DEF).
STRD $B465
*** STR$-Funktion: Berechnet Ausdruck und verwandelt das Ergebnis in eine ASCII-Zeichenkette.
STRINI $B475
Schafft im Zeichenkettenbereich Platz zum Einfügen einer Zeichenkette: A enthält die Länge und (FAC1+3) zeigt auf die Zeichenkette (zum Beispiel im Eingabepuffer). Beim Verlassen enthält $61 bis $63 den Descriptor der neuen Zeichenkette. CHR$, LEFT$ und so weiter arbeiten sämtlich mit dieser Routine.
STRLIT $B487
Kopiert eine Zeichenkette in den Zeichenkettenbereich am oberen Speicherende; beim Eintritt in die Routine zeigt (A/Y) auf die Zeichenkette. Sucht nach » " « oder einem Nullbyte als Endmarkierung, um die Länge zu bestimmen. Beim Verlassen enthalten $61, $62, $63 den Descriptor.
GETSPA $B4F4
Weist im dynamischen Zeichenkettenbereich am oberen Speicherende Platz für eine Zeichenkette zu; die Länge ist in A festgehalten. Führt eine Garbage Collection durch, wenn der Platz erschöpft ist. Aufgerufen von STRINI.
GARBA2 $B526
Führt Garbage Collection aus; sammelt die gültigen Zeichenketten und entfernt überflüssige aus dem Zeichenkettenbereich. Bei einer großen Zahl von Zeichenketten wird die Routine für Garbage Collection langsam.
DVARS $B606
Sucht Variablen und Felder nach der nächsten, durch die Garbage Collection zu sichernde Zeichenkette ab.
CAT $B63D
Verknüpft zwei Zeichenketten.
MOVINS $B67A
Verschiebt Zeichenkette in den Zeichenkettenbereich oben im Speicher; beim Eintritt zeigt ($6F) auf den Descriptor der betreffenden Zeichenkette.
ERESTR $B6A3
Verwirft Zeichenkette: Beim Eintritt zeigt (FAC1+3) auf den Zeichenketten-Descriptor; beim Verlassen finden sich neue Zeichenkettenlänge und Zeiger in INDEX1.
FRETMS $B6DB
Löscht den Descriptor-Stapel.
CHRD $B6EC
*** CHR$: Stellt eine Zeichenkette der Länge 1 auf.
LEFTD $B700
*** LEFT$.
RIGHTD $B72C
*** RIGHT$.
MIDD $B737
*** MID$.
PREAM $B761
Holt Zeiger für Zeichenketten-Descriptor nach $50,$51 und die Länge nach A (auch nach X).
LEN $B77C
** LEN: Fließkommawert des Parameters Zeichenkettenlänge, plaziert in FAC1.
LEN1 $B782
Ermittelt Länge der Zeichenkette, setzt das Ergebnis in Y, schaltet von Zeichenketten-Modus auf Zahlen-Modus. Aufgerufen von LEN, VAL.
ASC $B78B
*** ASC: Holt das erste Zeichen einer Zeichenkette und wandelt es in einen Fließkommawert in FAC1 um. Eine Zeichenkette der Länge 0 erzeugt den Fehler ?SYNTAX ERROR.
GTBYTC $B79B
Liest einen Ausdruck aus dem Basic-Text und wertet ihn aus; muß einen 1-Byte-Wert liefern, der dann in X und FAC1+4 abgelegt wird.
VAL $B7AD
*** VAL: Wandelt Wert in Fließkommazahl in FAC1 um.
GETNUM $B7EB
Liest die Parameter für WAIT und POKE aus dem Basic-Text; setzt den ersten (2-Byte-Ganzzahl) in $14,$15 und den zweiten in X ein.
GETADR $B7F7
Verwandelt FAC1 in 2-Byte-Ganzzahl (Bereich 0...65535) in $14,$15 und Y/A.
PEEK $B80D
** PEEK: Beim Eintritt enthält FAC1 die Adresse, die gelesen werden soll, im Fließkomma. Beim Verlassen steht der abgelesene Wert in Y.
POKE $B824
*** POKE: Holt zwei Parameter aus dem Text und führt POKE aus.
WAIT $B82D
*** WAIT: Holt zwei Parameter aus dem Text, und eventuell noch einen dritten, der als 0 betrachtet wird, wenn nicht vorhanden. Tritt in eine WAIT-Schleife ein.
FADDH $B849
Addiert 0,5 zum Inhalt des FAC1; dient zum Runden.
FSUB $B850
Fließkomma-Subtraktion: FAC1 wird ersetzt durch den MFLPT-Wert, auf den (A/Y) zeigt, minus FAC1.
FSUBT $B853
*** Fließkomma-Subtraktion: FAC1 wird ersetzt durch (FAC2 minus FAC1).
FADD $B867
Fließkomma-Addition: FAC1 wird ersetzt durch den MFLPT-Wert, auf den (A/Y) zeigt, plus FAC1.
FADDT $B86F
Fließkomma-Addition: FAC1 wird ersetzt durch (FAC2 plus FAC1). Enthält beim Eintritt den Exponenten von FAC1, zum Beispiel Inhalt von $61: addiert »0« schneller.
COMPLT $B947
Ersetzt FAC1 durch sein Zweierkomplement.
OVERR $B97E
Gibt die Meldung ?OVERFLOW ERROR und anschließend READY aus.
MULSHF $B983
Multipliziert mit einem Byte.
FONE $B9BC
Tabelle von Konstanten im MFLPT-Format: zuerst eine »1«, dann ein Byte vom Wert 3, dann Konstante zur Berechnung von LOG, dann SQR(0.5),SQR(2),—0.5 und LOG(2).
LOG $B9EA
*** LOG: Berechnet vom Inhalt des FAC1 den Logarithmus zur Basis e.
FMULT $BA28
Fließkomma-Multiplikation: FAC1 wird ersetzt durch den MFLPT-Wert, auf den (AY/Y) zeigt, mal FAC1.
FMULTT $BA30
*** Fließpunkt-Multiplikation: FAC1 wird ersetzt durch FAC1 mal FAC2.
MLTPLY $BA59
Multipliziert FAC1 mit einem Byte und speichert das Ergebnis in $26...$2A.
CONUPK $BA8C
Lädt FAC2 mit dem MFLPT-Wert bei (A/Y), isoliert das Vorzeichenbit, speichert es separat und bildet so das FLPT-Format. Beim Verlassen enthält A das erste Byte von FAC1.
MULDIV $BAB7
Prüft Akkumulatoren für Multiplikation und Division: Ist FAC2 »0«, wird FAC1 »0« gesetzt; ist die Summe der Exponenten zu groß, erfolgt die Meldung ?OVERFLOW ERROR, wenn zu klein, wird das Ergebnis ohne Unterlaufmeldung auf 0 gesetzt.
MUL10 $BAE2
Multipliziert FAC1 mit 10 und setzt das Ergebnis in FAC1.
TENC $BAF9
10 im MFLPT-Format.
DIV10 $BAFE
Dividiert FAC1 durch 10 und legt das Ergebnis in FAC1 ab.
FDIVF $BB07
Fließkomma-Division: FAC1 wird ersetzt durch FAC2 dividiert durch denjenigen MFLPT-Wert, auf den (A/Y) zeigt; beim Einsprung enthält X das Vorzeichen des Resultats.
FDIV $BB0F
Fließkomma-Division: FAC1 wird ersetzt durch den MFLPT-Wert, auf den (A/Y) zeigt, dividiert durch FAC1.
FDIVT $BB14
*** Fließkomma-Division: FAC1 wird ersetzt durch (FAC2 dividiert durch FAC1).
MOVFM $BBA2
Lädt FAC2 mit dem MFLPT-Wert bei (A/Y), holt das Vorzeichenbit heraus, speichert es separat und bildet so das FLPT-Format.
MOV2F $BBC7
Verwandelt FAC1 in MFLPT-Format und speichert das Resultat in $5C bis $60, TEMPFP2.
MOV1F $BBCA
Verwandelt FAC1 in MFLPT-Format und speichert das Resultat in $57 bis $5B, TEMPFP1.
MOVVF $BBD0
Verwandelt FAC1 in MFLPT-Format und speichert das Resultat an der Adresse, auf die ($49) zeigt.
MOVMF $BBD4
Verwandelt FAC1 in MFLPT-Format und speichert das Resultat an der Adresse, auf die (A/Y) zeigt.
MOVFA $BBFC
Kopiert FAC2 in FAC1.
MOVAF $BC0C
Rundet FAC1 durch Aufruf von ROUND und kopiert das Ergebnis in FAC2.
ROUND $BC1B
Rundet FAC1.
SIGN $BC2B
Ermittelt das Vorzeichen von FAC1: beim Verlassen ist A=0, wenn der Wert in FAC1 null ist, A=1, wenn er positiv ist und A=$FF, wenn er negativ ist.
SGN $BC39
*** SGN-Funktion: Ruft SIGN auf und verwandelt dann A in Fließkommaform in FAC1.
ABS $BC58
*** ABS-Funktion: Verwandelt FAC1 in ABS(FAC1).
FCOMP $BC5B
Vergleicht FAC1 mit dem MFLPT-Wert bei (A/Y); beim Verlassen ist A=0, wenn die Werte gleich sind, A=1, wenn FAC1 >MFLPT, und A=$FF, wenn FAC1 <MFLPT.

QINT $BC9B
Verwandelt FAC1 in 4-Byte-Ganzzahl und speichert das Ergebnis, höchstes Byte zuerst, in (FAC1+1) (FAC+4).

INT $BCCC
INT-Funktion: Rundet FAC1 ab, beläßt das Resultat jedoch in FLPT-Form in FAC1.

FIN $BCF3
Wandelt eine ASCII-Zeichenkette, zum Beispiel »—99.375« in eine Zahl in FAC1 um. Beim Eintritt zeigt TXTPTR auf den Anfang. Die Umwandlung erfolgt dann durch JSR CHRGET/JSR FIN.

AADD $BD7E
Addiert den Inhalt von A zu FAC1.

STCONS $BDB3
3 Konstante in MFLPT-Form: 99999999.9, 999999999, 1000000000. Verwendet bei Zeichenkettenumwandlungen.

INPRT $BDC2
Druckt IN gefolgt von der aktuellen Zeilennummer in CURLIN ($39, $3A).

LINPRT $BDCD
Gibt die Ganzzahl in A/Y aus; Bereich 0...65535.

FOUT $BDDD
Verwandelt den Inhalt von FAC1 in eine ASCII-Zeichenkette, die mit der Adresse $0100 beginnt und mit einem Null-Byte endet. Beim Verlassen enthält (A/Y) die Startadresse, so daß STROUT die Zeichenkette ausgeben kann.

FOUTIM $BE68
Verwandelt TI in ASCII-Zeichenkette, die mit der Adresse $0100 beginnt und mit einem Null-Byte endet.

TICONS $BF11
Konstanten zur Umwandlung von Zeichenketten und TI sowie der Wert 0,5 in MFLPT-Form, danach 15 weitere Konstanten C4-Byte-Ganzzahlen).

SQR $BF71
*** SQR: FAC1 wird durch die Quadratwurzel aus FAC1 ersetzt.

FPWRT $BF7B
** Führt Potenzberechnungen aus: FAC1 wird ersetzt durch FAC2 hoch FAC1. Beim Eintritt muß A den Inhalt von FAC2 (das heißt von $69) speichern, damit Potenzen von 0 korrekt sind.

NEGOP $BFB4
Macht FAC1 negativ.

EXCONS $BFBF
Tabelle von 8 Konstanten zur Auswertung von EXP-Reihen.

EXP $BFED
*** EXP-Funktion: FAC1 wird durch e hoch FAC1 ersetzt.

POLYX $E059
Routine zur Reihenberechnung. Beim Eintritt zeigt A/Y auf den Zähler am Anfang der Konstantentabelle, die zur Berechnung der Potenzreihe herangezogen wird.

RMULC $E08D
11879546.4 im MFLPT-Format: multiplikative Konstante zur Auswertung von RND.

RADDC $E092
3.92767778 E-8 im MFLPT-Format: additive Konstante zur Auswertung von RND.

RND $E097
*** RND: Setzt in FAC1 je nach seinem Vorzeichen auf folgende Weise eine Zahl:

RNDO $E09E
Wenn 0, wird FAC1 von den Registern der CIA-Timer geladen: eine einfache Art, einen neuen Keim für Zufallszahlen zu setzen.

QSETNR $E0BE
Wenn >0, wird die in ($88...$8C gespeicherte) durch vorhergehende Aufrufe erzeugte Zufallszahl mit RMULC multipliziert und RADDC hinzugezählt; das Ergebnis steht in FAC1.

RND1 $E0D3
Wenn <0, wird FAC1 mit vermischten Bytes von sich selbst geladen, daher ist RND(-We) Konstant und also wiederholbar. In allen diesen drei Fällen wird FAC1 in $88...$8C gespeichert.

RNDRNG $E0E5
Zwingt FAC1 in den Bereich 0...1,0 und C gespeichert.

BIOERR $E0F9
Fehlerbehandlung für bestimmte Basic-Aufrufe des Kernel (erforderlich zur Verarbeitung von CMB, LOAD, SAVE), falls bei der Rückkehr von der Kernal-Routine das Fehlerflag C gesetzt ist.

BCHOUT $E10C
Gibt Zeichen mittels CHROUT aus; Fehlermeldung bei Versagen.

BCHIN $E112
Nimmt Zeichen mittels CHRIN herein; Fehlermeldung bei Versagen.

BCKOUT $E118
Richtet mittels CHKOUT eine Ausgabedatei ein; Fehlermeldung bei Versagen.

BCKIN $E11E
Richtet mittels CHKIN eine Eingabedatei ein; Fehlermeldung bei Versagen.

BGETIN $E124
Holt Zeichen mittels GETIN; Fehlermeldung bei Versagen.

SYS $E12A
*** SYS: Lädt A, X, Y, SR aus $30C..., ruft MC-Routine an der Adresse auf, die in der Anweisung als Argument angegeben ist. Lädt bei der Rückkehr von der Routine alle Registerinhalte aus $30C... zurück.

SAVET $E156
*** SAVE: Sichert ein Basic-Programm: läßt A auf die Adresse in Seite Null zeigen, die ihrerseits auf die Startadresse zeigt; setzt (X/Y) auf $2D,$2E = Programmende. Anschließend wird über einen Vektor bei $FFD8 die Kernal-Routine SAVE aufgerufen.

VERFYT $E165
*** VERIFY: Setzt das Flag in A auf 1, um die Verify-Operation anzuzeigen; tritt in LOADT ein und prüft auf Fehler.

LOADT $E168
*** LOAD: Holt die Parameter aus dem Basic-Text und stellt sie auf; ruft die Kernal-Routine LOAD über einen Vektor bei $FFD5 auf.

LOADR $E16F
Lädt vom bereits angesprochenen Gerät ins RAM ab der Basic-Adresse in ($2B).

LDFIN $E195
Beendet das Laden. Nach Aufruf von LOAD im Direktmodus wird der Zeiger auf das obere Ende von Basic ($2D) auf die Adresse des letzten geladenen Bytes gesetzt. Nach einem Aufruf aus einem Programm heraus unterbleibt dies, so daß die Variablenliste bewahrt ist. Dann wird der Zeiger in CHRGET zurückgesetzt und ein Basic-Warmstart durchgeführt, um das neue Programm zu starten.

OPENT $E1BE
*** OPEN: Liest die Parameter aus dem Text und stellt sie durch entsprechende Kernel-Aufrufe auf. Ruft über den Vektor bei $FFC0 die Kernal-Routine OPEN auf.

CLOSET $E1C7
*** CLOSE: Liest die Parameter aus dem Text und stellt sie auf. Ruft über den Vektor bei $FFC3 die Kernel-Routine CLOSE auf.

SLPARA $E1D4
Holt die Parameter für LOAD, SAVE und VERIFY aus dem Basic-Text; setzt die Standardwerte, wenn Angaben fehlen. Richtet durch einen Aufruf von SETLFS über den Vektor bei $FFBA eine Datei ein.

COMBYT $E200
Prüft auf ein Komma, wertet den folgenden 1-Byte-Parameter aus und setzt ihn in X.

CMMERR $E20E
Prüft auf Komma, dem irgendwas außer dem Anweisungsende folgt: andernfalls ?SYNTAX ERROR.

OCPARA $E219
Holt die Parameter für OPEN/CLOSE-Aufrufe aus dem Basic-Text; setzt die Standardwerte, wenn Angaben fehlen.

COS $E26
*** COS: FAC1 wird durch COS(FAC1) ersetzt.

SIN $E26 B
*** SIN: FAC 1 wird durch SIN(FAC1) ersetzt.

TAN $E2B4
*** TAN: FAC1 wird durch TAN(FAC1) ersetzt.

$E2E0
Tabelle von Konstanten im MFLPT-Format: Pi/2, Pi*2 und Pi*0,25. Danach folgt ein Zähler (5) und 6 MFLPT-Konstanten zur Berechnung von SIN.

ATN $E30E
*** ATN: FAC1 wird durch ARCTAN(FAC1) ersetzt.

$E33E
Zähler (11) und Tabelle mit 12 Konstanten im MFLPT-Format zur Berechnung von ATN.

BASSFT $E37B
Basic-Warmstartroutine. Eintritt mit JMP ($A002): Teil (nur) der Interrupt-Sequenz, die infolge einer BRK-Instruktion oder auf eine Betätigung der Tasten STOP/RESTORE hin abläuft. Schließt alle I/O-Kanäle, restauriert den Stapelspeicher, gibt die Meldung ?BREAK ERROR aus und springt zu READY.

INIT $E394
Basic-Kaltstart. Eintritt mit JMP ($A000): Teil der RESET-Sequenz. Führt INTV, INITCZ, INITMS aus, setzt den Stapelzeiger und springt zu READY.

CHRCPY $E3A2
Routine CHRGET und Keim für RND im ROM für Verlegung ins RAM.

INITCZ $E3BF
Initialisiert Sprunginstruktion für USR und den Standardvektor sowie die Vektoren von $03...$06. Überträgt CHRGET und Keim für RND in das RAM; ruft die Kernel-Routinen MEMBOT und MEMTOP auf, um die Zeiger für Basic-Anfang und oberes Speicherende ($2B,$37) gemäß den beim Einschalten initialisierten Zeigern bei $282...$285 zu setzen. Setzt in 2048 das Nullbyte für Programmende.

INITMS $E422
Gibt die Einschaltmeldung "*** COMMODORE 64 BASIC V2 *** 64 K RAM SYSTEM" und die Zahl der freien Bytes (auf dem C64 gewöhnlich 38911) aus.

INITV $E453
Initialisiert die Vektoren für ERROR, MAIN etc. an den Adressen $0300...$030B.

CPATCH $E4DA
Korrektur, um die momentane Hintergrundfarbe in das aktuelle Nibble des Farb-RAM zu schreiben; das mindert das Flimmern des Bildschirms. Aufrufen von $EAOB (eine von CLR benutzte Routine).

IOBASK $E500
Kernel-Routine IOBASE. Gibt die Basisadresse der CIA in X/Y aus. Verwerndet von der Kernel-Routine SCNKEY (Tastaturabfrage).

SCRENK $E505
Kernel-Routine SCREEN gibt die Bildschirmeinstellung aus: die Zahl der Spalten (40) in X, die Zahl der Zeilen (25) in Y.

PLOTK $E50A
Kernel-Routine PLOT. Setzt den Cursor auf X (Zeile), Y (Spalte), oder gibt die aktuellen Werte für Zeile, Spalte aus.

CINT $E518
Allgemeine Initialisierung von Bildschirm und VIC-Chip: Stellt die Tabellen für die Bildschirmeditierung an den Adressen $D9 bis $F2 auf, initialisiert den VIC-Chip, setzt die Zeichenfarbe auf hellblau, führt CLR und HOME aus und stellt in $9A die Standardadresse der I/O-Geräte ein.

HOME $E566
Bringt den Cursor in die Grundpositionn (links oben).

INITVC $E5A0
Initialisiert den VIC-Chip mittels der Wertetabelle bei $ECB9...$ECE6.

GETKBC $E5B4
Holt ein Zeichen aus dem Tastaturpuffer und schiebt die übrigen Zeichen weiter. der Puffer muß beim Eintritt mindestens 1 Zeichen enthalten (die Länge des Pufferinhalts ist in $C6 festgehalten). Beim Verlassen enthält A das Zeichen.

INPPRO $E5CA
Liest SHIFT-STOP, RETURN etc. und verarbeitet sie.

QTSWC $E684
Kehrt das Anführungszeichen-Flag ($D4) um, wenn A beim Eintritt ein Anführungszeichen enthält.

PRT $E716
Gibt das Zeichen in A zum Bildschirm aus. Behandelt die Zeichen für Cursorsteuerung, Bildschirmeditierung, zur Einstellung der Farben etc. Besorgt außerdem den Übergang zur nächsten Zeile und das Scrollen.

CHKCOL $E8CB
Prüft A auf ein Farbcode-Zeichen: Ändert die Farbe in $0286, wenn eines gefunden.

COLTAB $E8DA
Tabelle der 16 Farbcode-Zeichen in der Anordnung Schwarz, Weiß, Rot, Cyan etc.

SCROL $E8EA
Srollt den Bildschirm. Ist die oberste Zeile länger als 40 Zeichen, wird um 2 Zeilen gescrollt, um sie vollständig zu entfernen. Verzögert, wenn die »CTRL«-Taste gedrückt ist: der Test darauf erfolgt durch direktes Abfragen des CIA-Chips.

CLRIN $E9FF
Löscht die X-te Bildschirmzeile.

DSPP $EA 13
Setzt das Zeichen in A an die Cursorposition auf den Bildschirm; keine Prüfung auf Steuerzeichen und so weiter. Die Farbe befindet sich in X.

KEY $EA31
Interrupt-Dienstroutine: Bei unverändertem Vektor in ($0314) verarbeitet diese Routine alle IRQ-Interrupts. Die Funktionen von KEY sind: Taktzähle und Speicherstelle $91 mittels der Kernel-Routine UDTIM aktualisieren; das Cursorblinken aufrechterhalten, falls der Cursor aktiviert ist (siehe $CC...$CF); den Motor des Bandgeräts gemäß der Flag bei $CO ein- oder ausschalten; die Tastatur mittels der Kernel-Routine SCNKEY auf ein neues Zeichen hin überprüfen. Schließlich wird noch das Interrupt-Register bei $DC00 im CIA gelöscht, Y,X und A werden wiederhergestellt und mit RTI erfolgt die Rückkehr zum Hauptprogramm.

SCNKEY $EA87
Kernel-Routine SCNKEY. Prüft auf einen Tastendruck; liest Spalte und Zeile der Tastatur-Matrix, nimmt die entsprechenden Änderungen vor, falls Tasten wie SHIFT, CTRL etc. gedrückt sind, wandelt den Matrixwert mittels Tabellen ab $EB81 in den CBM-ASCII-Wert um und plaziert ihn in Tastaturpuffer, wenn dort noch Platz ist.

SHFOG $EB48
Logische Behandlung der SHIFT- Taste.

KBDTBL $EB81
Tabellen zur Umwandlung der Matrixwerte in CBM-ASCII-Werte; 3 Tabellen für Normal-SHIFT- und Graphikmodus; eine vierte für die CTRL-Codes findet sich in$EC78...$ECB8. Anfangswerte für den VIC-Chip (die Sprite-Farben sind falsch gesetzt).

LDRUN $ECE)
LOAD RETURN RUN RETURN für den Tastaturpuffer.

Fortsetzung im nächsten Extra