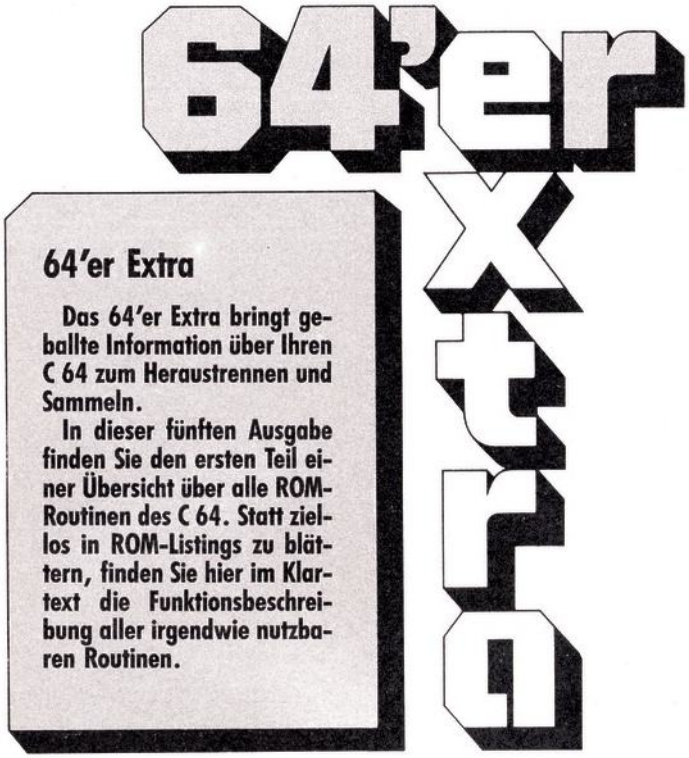

BCOLD $A000
Basic-Kaltstartvektor ($E394). Unterzieht Basic der Prozedur NEW, gibt die Meldung BYTES FREE und READY aus. Dies beendet die Reset-Sequenz; siehe die Routinen bei $E394 und $FCE2.

BWARM $A002
Basic-Warmstartvektor ($E37B). Unterzieht Basic der Prozedur CLR, gibt READY aus. Dies steht am Ende der SHIFT-RESTORE-Sequenz; siehe die Routinen bei $E37B und $FE43.

$A004
Meldung des CBM-Basic.

$A00C
Tabelle der um 1 verminderten Adressen der Basic-Befehle (END, FOR, NEXT, etc.).

$A052
Tabelle der Adressen von Zahlen- und Stringfunktionen (SGN, INT, ABS, etc.).

$A080
Tabelle der um 1 verminderten Adressen der Routinen zur Bearbeitung der Basic-Operatoren (addieren, subtrahieren, dividieren, etc.); jeder Adresse folgt ein Byte, das die Priorität des Operators angibt.

$A09E
Basic-Schlüsselwörter als Zeichenketten im CBM-ASCII mit gesetztem Bit 7 des letzten Zeichens.

$A129
Tabelle verschiedener Schlüsselwörter, die keine Aktionsadressen haben, (TAB, TO, FN, etc.); Bit 7 des letzten Zeichens ist gesetzt.

$A140
Tabelle der Schlüsselzeichen der Opeatoren; außerdem AND, OR als Zeichenketten mit Bit 7 = '1' des letzten Zeichens.

$A14D
Tabelle der Schlüsselwörter der Funktionen (SGN, INT, ABS, etc.) mit gesetztem Bit 7 des letzten Zeichens.

$A19E
Tabelle der 28 Fehlermeldungen (TOO MANY FILES, FILE OPEN etc.) mit Bit 7 des letzten Zeichens gesetzt.

$A328
Tabelle der Zeiger zu den Fehlermeldungen

$A364
Tabelle der anderen Meldungen OK, ERROR IN, READY, BREAK.

FNDFOR $A38A
Prüft den Stapelspeicher auf den Eintrag für FOR. Aufgerufen von NEXT: wird FOR nicht gefunden, erscheint die Meldung ?NEXT WITHOUT FOR. Löscht auch, wenn von RETURN aufgerufen, einen FOR-Datenblock vom Stapel.

BLTU $A3B8
Schafft eine Lücke im Basic-Text zur Einfügung einer neuen Basic-Zeile. Prüft, ob genug Platz vorhanden, dann:

BLTUC $A3BF
Verschiebt den Block zwischen ($5F und ($5A)—1 hinauf zu einem neuen Blockende bei ($58)—1.

GETSTIK $A3FB
Prüft, ob der Stapelspeicher noch 2*A Bytes faßt: wenn nicht, erscheint die Meldung ?OUT OF MEMORY.

REASON $A408
Prüft, ob der Zeiger in (A/Y) unterhalb FRETOP (aktuelle untere Grenze des Zeichenkettenbereichs) zeigt. Wenn ja, wird die Routine verlassen; wenn nein, erfolgt Garbage Collection: wenn danach immer noch nein, wird ?OUT OF MEMORY ausgegeben.

ERROR $A437
Gibt Fehlermeldung aus; X enthält die Fehlernummer (= die Hälfte der Versetzung innerhalb der Adressentabelle der Fehlermeldungen). Wird durch einen Vektor in ($0300) nach $A43A gelenkt. Dann werden die temporären Adressen für Tastatureingabe und Bildschirmausgabe gesetzt, der Steckpointer zurückgesetzt und, falls im Programm-Modus, die Fehlermeldung zusammen mit der Zeilennummer ausgegeben. Dann:

READY $A474
Startet Basic neu: Gibt READY aus, schaltet auf Direktmodus, dann:

MAIN $A480
Nimmt eine Zeile in den Eingabepuffer herein und fügt ein abschließendes Nullbyte an. Prüft, ob Programmzeile oder Befehl im Direktmodus: führt letzteren sofort aus. (MAIN wird über einen Vektor in ($0302) zu $A483 weitergeleitet.)

MAIN1 $A49C
Ist es eine Programmzeile, wird sie verschlüsselt.

INSLIN $A4A4
Ersetzt die alte Zeile durch die neue, falls die Zeilennummer bereits vorhanden; fügt ansonsten die neue Zeile ein. Die Zeilennummer befindet sich beim Eintritt in $14, $15; die Länge +4 steht in Y. Ist das erste Byte im Puffer gleich 0, handelt es sich um eine leere Zeile: die alte Zeile wird gelöscht.

FINI $A52A
Führt nach dem Einfügen einer neuen Zeile RUNC (dadurch gehen beim Editieren die Variablen verloren und anschließendes CONT ist nicht mehr möglich) und LNKPRG aus; springt anschließend zu MAIN.

LNKPRG $A533
Verkettet die Bindezeiger im Basic-Programm; verwendet dabei Nullbytes als Markierungen für das Zeilenende.

INLIN $A560
Nimmt eine Bildschirmzeile in den Basic-Puffer ab $0200 herein und fügt ein abschließendes Nullbyte an.

RDCHR
Setzt einzelnes Zeichen in A.

CRUNCH $A579
Verschlüsselt die Schlüsselwörter im Eingabepuffer. Wird über einen Vektor in ($0304) gewöhnlich nach $A57C gelenkt.

FNDLIN $A613
Sucht den Basic-Text von Anfang an nach der Zeilennummer in $14, $15 ab. Bei gefundener Zeilennummer wird das Übertragsflag gesetzt; ($5F) zeigt auf die Bindeadresse.

FNDLNC $A617
Sucht Basic-Text ab A (niederwertiges Byte) und Y (höherwertiges Byte) nach der Zeilennummer in $14, $15 ab.

SCRATH $A642
***NEW springt hierher: prüft die Syntax, dann:

SCRTCH $A644
Setzt die ersten 2 Byte des Textes (= der erste Bindezeiger) auf 0; lädt den Zeiger für den Anfang der Variablen $2D,$2E mit dem Anfang von Basic+2, dann:

RUNC $A659
Setzt mittels STXPT den Zeiger innerhalb GETCHR auf den Anfang des Basic-Texts, dann:

CLEAR $A65E
***CLR: Löscht die Variablen durch Zurücksetzen der Zeiger zum Ende des Variablenbereichs auf den Wert des Zeigers für das Programmende; eventuelle Zeiger von Zeichenkettenvariablen werden ebenfalls zurückgesetzt. Bricht I/O-Aktivitäten ab und setzt den Stackpointer zurück.

STXPT $A68E
Setzt den Zeiger innerhalb der Routine GETCHR auf den Anfang des Basic-Texts zurück. (das heißt, lädt $7A,$7B mit $2B,2C-1).

LIST $A69C
*** EINSPRUNG für den LIST-Befehl.

LIST1 $A6C9
Listet eine Zeile Basic: zuerst die Zeilennummer, dann den Text.

QPLOP $A717
Bearbeitet das zu listende Zeichen: ist es ein gewöhnliches oder ein Steuerzeichen in Anführungszeichen, wird es ausgegeben; entschlüsselt und druckt Schlüsselzeichen über einen Vektor in ($0306) nach $A71A gelenkt.

FOR $A742
*** Bearbeitung von FOR. Schiebt 18 Byte auf den Stapelspeicher. 1) Zeiger auf die nächstfolgende Anweisung, 2) momentane Zeilennummer, 3) oberer Schleifenwert, 4) Schrittweite (Standardwert = 1), 5) Name der Schleifenvariablen und 6) das Schlüsselzeichen für FOR.

NEWSTT $A7AE
Führt Basic aus: Prüft auf die STOP-Taste und auf das Nullbyte für das Zeilenende oder auf einen Doppelpunkt.

CKEOL $A7C4
Hält an, wenn Textende erreicht und setzt andernfalls den Zeiger innerhalb CHRGET auf den Anfang der nächsten Zeile.

GONE $A7E1
Bearbeitet die Basic-Anweisung in der momentanen Zeile. Über einen Vektor in ($0308) nach A7E4 gelenkt; kehrt in einer Schleife zurück zu NEWSTT.

GONE3 $A7ED
Führt ein Basic-Schlüsselwort aus. Bezieht die Adresse zum Starten der Routine aus der Tabelle bei $A00C. Nimmt "LET" an, wenn das erste Byte der Anweisung kein Schlüsselwort ist. Legt Adresse auf den Stapelspeicher, so daß der RTS-Befehl von CHRGET dorthin springt.

RESTOR $A81D
***RESTORE: Setzt den Datenzeiger bei $41, $42 auf den Anfang des Basic-Textes.

STOP $A82C
***STOP: auch END sowie Programmunterbrechung. Speichert Information für CONT (Zeiger in den Basic-Text, Zeilennummer). STOP gibt BREAK IN xxx aus, während END dies überspringt und stattdessen READY druckt. Die STOP-Taste aktiviert STOP; Ein Nullbyte für das Programmende ruft END auf.

CONT $A857
*** CONT: Setzt den Programmlauf fort. Setzt hierzu die aktuelle Zeilennummer ($39, $3A) und den Zeiger innerhalb CHRGET auf die von STOP gespeicherten Werte. ?CANT'T CONTINUE ERROR erscheint, wenn das höherwertige Byte des Zeigers bei einem Syntax-Fehler auf »0« gesetzt wurde.

RUN $A871
*** RUN: Löscht alle Variablen (CLR), setzt den Stapelspeicher zurück, setzt CHRGET auf den Anfang von Basic und beginnt mit der Ausführung des Programms. Auf RUN xxx folgt CLR aller Variablen, das Zurücksetzen des Stapelspeichers und schließlich GOTO xxx.

GOSUB $A883
*** GOSUB: Schiebt 5 Byte auf den Stapelspeicher: 1) den Zeiger innerhalb CHRGET, 2) die aktuelle Zeilennummer, 3) das Schlüsselzeichen für GOSUB; anschließend wird GOTO aufgerufen.

GOTO $A8A0
*** GOTO: Holt sich die auf die GOTO-Anweisung folgende Zeilennummer und sucht den Basic-Text nach dieser Zeile ab. Ist das höherwertige Byte der Bestimmungszeile größer als das der aktuellen Zeilennummer, erfolgt die Suche ab der momentanen Zeile aufwärts, um die Suchzeit abzukürzen; sonst beginnt die Suche von Anfang an. Setzt Zeiger auf die gefundene Zeile in CHRGET ein.

RETURN $A8D2
*** RETURN: Der Stapelspeicher wird bis zum GOSUB-Schlüsselzeichen restoriert (?RETURN WITHOUT GOSUB, wenn keines gefunden). Anschließend werden die Nummer und der Zeiger der aufrufenden Zeile wiederhergestellt und die Programmausführung fortgesetzt.

DATA $A8F8
*** DATA: Die Routine veranlaßt CHRGET, die DATA-Anweisung bis zum abschließenden Nullbyte oder einem Doppelpunkt zu überspringen.

DATAN $A906
Sucht nach der Endemarkierung der Anweisung; beim Aussprung enthält Y den Unterschied zwischen Zeilenende und dem Zeiger der Routine CHRGET.

REMN $A909
Sucht nach dem Ende einer Basic-Zeile.

IF $A928
*** IF: Wertet den Ausdruck aus; ist das Resultat »Falsch« (das heißt gleich 0), wird der THEN- oder GOTO-Teil mittels REM übergangen.
REM $A93B
*** REM: Sucht nach dem Ende der Zeile und bringt den Zeiger in CHRGET auf den neuen Stand. Dadurch wird der Inhalt der REM-Zeile übergangen.
DOCOND $A940
Setzt IF fort: Ist der Ausdruck wahr, wird der nächste Befehl, oder, wenn eine Zahl folgt, GOTO ausgeführt.
ONGOTO $A94B
*** ON: Berechnet den Ausdruck, prüft auf Schlüsselzeichen für GOTO oder GOSUB, geht die Liste der Zeilennummern nach der angegebenen Zeilennummer durch, überspringt dabei Kommata, und führt schließlich GOTO oder GOSUB dorthin aus.
LINGET $A96B
Liest aus dem Basic-Text eine Ganzzahl (gewöhnlich eine Zeilennummer) in $14, $15 ein; die Zahl muß im Bereich 0...63999 liegen.
LET $A9A5
*** LET: Sucht Zielvariable in der Variablenliste (oder erzeugt sie, falls nicht vorhanden), prüft auf das Schlüsselzeichen für "=", berechnet den Ausdruck und setzt das Ergebnis oder den Zeichenkettendescriptor in die Variablenliste.
PUTINT $A9C4
Rundet den FAC1 und sezt ihn als Ganzzahl in die Variablenliste an die aktuelle Variablenposition, auf die ($49) zeigt.
PTFLPT $A9D6
Setzt FAC1 in die Variablenliste an die durch ($49) angegebene Stelle.
PUTTIM $A9E3
Ordnet die Systemvariable TI$ zu.
ASCADD $AA27
Addiert ASCII-Ziffer zu FAC1.
GETSPT $AA2C
LET für Zeichenketten — setzt den Zeichenketten-Deskriptor, auf den (FAC1 + 3) zeigt, in die Variablenliste an die Stelle ($49).
PRINTN $AA80
PRINT: Ruft CMD auf, löscht dann die I/O-Kanäle und setzt die Standard-Geräteadressen wieder ein.

CMD $AA86
*** CMD: Setzt mittels der Kernel-Routine CHKOUT das CMD-Ausgabegerät aus der Dateitabelle und ruft PRINT auf.
STRDON $AA9A
Teil der PRINT-Routine: Gibt Zeichenkette aus und fährt mit der Abwicklung von PRINT fort.
PRINT $AAA0
*** PRINT: Identifiziert PRINT-Parameter (TAB, SPC, Komma, Strichpunkt usw.) und berechnet Ausdrücke.
VAROP $AAB8
Gibt Variable aus; Zahlen werden zunächst in eine Zeichenkette umgewandelt.
CRDO $AAD7
Gibt eine Zeilenschaltung aus, der ein Zeilenvorschub folgt, wenn die Kanaladresse 128 ist.
STROUT $AB1E
Druckt die Zeichenkette, auf die (A/Y) zeigt, bis ein Nullbyte oder Anführungszeichen angetroffen wird.
STRPRT $AB21
Druckt Zeichenkette: (FAC1+3) zeigt auf den Zeichenketten-Deskriptor.
OUTSTR $AB24
Zeichenkettenausgabe: ($22) zeigt auf die Zeichenkette; die Länge ist in A festgehalten.
OUTSPC $AB3B
Ausgabe eines Cursorschritts nach rechts (oder eines Leerzeichens, wenn die Ausgabe nicht zum Bildschirm erfolgt).
PRTSPC $AB3F
Gibt Leerzeichen aus.
OUTSKP $AB42
Gibt Cursorschritt rechts aus.

OUTQST $AB45
Gibt Fragezeichen vor Fehlermeldung aus.
OUTDO $AB47
Gibt das Zeichen in A aus.
TRMNOK $AB4D
Gibt die entsprechenden Fehlermeldungen für GET, READ und INPUT aus.
GET $AB7B
*** GET: Prüft auf Direktmodus (nicht zulässig) und holt ein Zeichen von der Tastatur oder Datei.
INPUTN $ABA5
*** INPUT: Holt die Dateinummer, schaltet das Gerät ein, ruft INPUT auf und schaltet dann das Gerät ab.
INPUT $ABBF
*** INPUT: Gibt den Hinweistext des Benutzers aus, falls vorhanden, dann:
QINLIN $ABF9
Bringt ein »?« auf den Bildschirm und nimmt die durch eine Zeilenschaltung begrenzte Zeile Text in den Eingabepuffer herein.
READ $AC06
*** READ: GET und INPUT teilen sich diese Routine, werden aber durch ein Flag in $11 unterschieden.
INPCON $AC0D
Einsprung für INPUT: Setzt das Flag und ruft READ auf; (X/Y) zeigt auf den Puffer.
INPCO1 $AC0F
Einsprung für GET: Setzt das Flag und ruft READ auf; (X/Y) zeigt auf den Puffer.
DATLOP $ACB8
Geht Text durch und liest DATA-Anweisungen
VAREND $ACDF
Prüft auf eine »0« am Ende des Eingabepuffers; gibt die Meldung? EXTRA IGNORED aus, wenn nicht gefunden.
EXINT $ACFC
Die Meldungen? EXTRA IGNORED und ?REDO FROM START beide gefolgt von CRLF (Zeilenschaltung und Zeilenvorschub).
NEXT$AD1E
*** NEXT: Prüft auf das Schlüsselzeichen für FOR und die passende Variable auf dem Stapelspeicher; gibt ?NEXT WITHOUT FOR aus, wenn nicht gefunden; berechnet nächsten Wert: wenn dieser noch gültig, werden die aktuelle Zeilennummer und der Zeiger in CHRGET zurückgesetzt und die Schleife fortgesetzt.
FRMNUM $AD8A
Berechnet einen numerischen Ausdruck von Basic aus durch Aufruf von FRMEVL, dann:
CHKNUM $AD8D
Prüft durch Testen der Flag bei $0D, ob FRMEVL eine Zahl geliefert hat: Meldung ?TYPE MISMATCH ERROR, wenn nicht.
CHKSTR $AD8F
Prüft durch Testen der Flag bei $0D, ob FRMEVL eine Zeichenkette geliefert hat: Meldung ?TYPE MISMATCH ERROR, wenn nicht.

FRMEVL $AD9E
Wertet jeglichen Basic-Formelausdruck aus und meldet alle Syntaxfehler; setzt $0D (VALTYP) im Falle einer Zahl auf $00, bei einer Zeichenkette auf $FF. Setzt bei einer Fließkommazahl $0E (INTFLG) auf $00 und legt den Wertin FAC1 ab. War der Variablentyp ganzzahlig, wird INTFLG auf $80 gesetzt, das Ergebnis jedoch im Fließkommaformat in FAC1 belassen. Komplizierte Ausdrücke können eine Vereinfachung erfordern, um Platz im Stapelspeicher zu gewinnen und den Fehler ?OUT OF MEMORY zu vermeiden.
EVAL $AE83
Wertet einzelnes Glied in einem Ausdruck aus; sucht nach Ziffernketten in ASCII, Variablen, π, NOT, arithmetischen Funktionen und so weiter.
PIVAL $AEA8
Wert für π im 5-Byte-Fließkommaformat.
PARCHK $AEF1
Wertet Klammerausdrücke innerhalb des Textes aus.
CHKCLS $AEF7
Prüft, ob CHRGET auf ")" zeigt —?SYNTAX ERROR, wenn nicht.
CHKOPN $AEFA
Prüft, ob CHRGET auf "(" zeigt —?SYNTAX ERROR, wenn nicht.
CHKCOM $AEFD
Prüft, ob CHRGET auf " , " zeigt —?SYNTAX ERROR, wenn nicht.
SYNCHR $AEFF
Prüft, ob CHRGET auf ein Byte gleich dem in A zeigt: wenn ja, wird die Routine mit dem nächsten Byte in A verlassen, sonst ?SYNTAX ERROR.
SYNERR $AF08
Gibt ?SYNTAX ERROR aus und kehrt zum READY-Status zurück.
DOMIN $AF0D
Berechnet NOT.
TSTROM $AF14
Setzt das Übertragsflag auf 1, wenn (FAC1+3) auf ROM zeigt, was auf eine der reservierten Variablen TI$, TI, ST hinweist.
ISVAR $AF28
Sucht die Variablenliste nach der in $45, $46 genannten Variablen ab; beim Verlassen der Routine enthält FAC1 einen numerischen Wert im FLPT-Format (sowohl bei Ganzzahl- wie bei Fließkommavariablen); im Falle einer Stringvariablen zeigt (FAC1+3) auf den Zeichenketten-Descriptor.
TISASC $AF48
Liest den Taktzähler und bildet eine Zeichenkette, die TI$ enthält.
ISFUN $AFA7
Stellt Funktionstyp fest und wertet ihn aus.
OROP $AFE6
*** OR-Funktion: Setzt Flag und führt mit zwei Ganzzahlen aus je 2 Bytes in FAC1 und FAC2 logisches ODER (OR) aus.
ANDOP $AFE9
*** AND-Funktion: Diese Operationen werden durch eine Routine ausgeführt; ein Flag (in Y) enthält $FF für OR, $00 für AND. Wandel FLPT in Ganzzahl um (Fehlermeldung, wenn außerhalb des zulässigen Bereichs); das Ergebnis bleibt im FLPT-Format in FAC1.
DOREL $B016
*** Vergleiche: < = > prüfen die Variablentypen, dann:
NUMREL $B01B
Führt mittels FCOMP bei $DC5B numerische Vergleiche aus, oder:
STRREL $B02E
Führt Zeichenkettenvergleich aus: Aussprung mit X=0 bei gleichen Zeichenketten mit X=1, wenn die erste > die zweite und X=255, wenn die zweite > die erste.
DIM $B081
*** DIM: Stellt mittels der Routine PRTGET alle Feldelemente auf.
PTRGET $B08B
Prüft einen Variablennamen in Basic auf Zulässigkeit; das erste Zeichen muß ein Buchstabe sein, das zweite ein Buchstabe oder eine Zahl; was danach kommt, bleibt unbeachtet. Setzt VALTYP ($0D) auf $FF; wenn ein »$« angetroffen wird, sonst auf $00; setzt INTFLG ($0E) auf $80, wenn ein »%« gefunden wird. Der Name wird in VARNAM ($45, $46) zwischengespeichert. Dabei sind die höchsten Bits so gesetzt, daß sie über den Variablentyp Auskunft geben, wie in Kapitel 5 beschrieben.
ORDVAR $B0E7
Sucht die Variablenliste nach dem in VARNAM ($45, $46) abgelegten Namen ab und läßt VARPNT ($47) darauf zeigen. Definiert neue Variable, wenn Name nicht gefunden.
ISLETC $B113
Setzt das Übertragungsflag, wenn der Akkumulator einen der Buchstaben A...Z enthält.
NOTFNS $B11D
Stellt eine neue einfache Variable (keine Feldvariable) in der Variablenliste unmittelbar vor den Feldern auf. Der Name befindet sich in VARNAM ($45, $46). Eventuelle Felder müssen um 7 Byte nach oben rücken, um Platz für die neue Variable zu schaffen. Beim Verlassen zeigt ($5F) auf die neu eingerichtete Variable.
FMAPTR $B194
Berechnet Zeigerwert in $5F,$60; wird bei der Zuweisung von Platz für Felder gebraucht.
N32768 $B1A5
Enthält — 32768 als 5-Byte-Fließkommazahl.
FACINX $B1AA
Verwandelt FAC1 in eine Ganzzahl (—32768 bis +32767) und speichert sie in A/Y.
INTIDX $B1B2
Holt einen positiven Ganzzahlausdruck vom nächsten Teil des Basic-Texts und wertet ihn aus; liegt das Resultat im Bereich 0...32767, wird es in (FAC1+4, FAC1+3) abgelegt.
AYINT $B1BF
Wandelt FAC1 in Ganzzahl des Bereichs 0...32767 um; Ergebnis in FAC+3.
ISARY $B1D1
Holt Feldparameter aus dem Basic-Text: Zahl der Dimensionen, Zahl der Elemente. Legt sie auf den Stapelspeicher.
FNDARY $B218
Sucht nach dem Feldnamen in VARNAM ($45,$46) und nach den anderen Einzelheiten auf dem Stapel.
BSERR $B245
BAD SUBSCRIPT ERROR (falscher Index). BS+3 = ILLEGAL QUANTITY ERROR (nicht zulässige Größe).
NOTFDD $B261
Ist das Feld nicht vorhanden, nimmt diese Routine die Angaben auf dem Stapelspeicher und definiert das Feld mit der Dimension 10.
INPLN2 $B30E
Macht angegebenes Feldelement ausfindig und läßt VARPTN ($47) darauf zeigen.
UMULT $B34C
Berechnet den Abstand des angegebenen Feldelements von der Feldadresse, auf die (VARPTN) ($47) zeigt; setzt das Ergebnis in X/Y.
FRE $B37D
*** FRE: Führt Garbage Collection aus und läßt (Y/A) auf die unterste (Zeichenkette minus Zeiger auf das Ende der Felder) zeigen; plaziert das Ergebnis in FAC1 durch Aufruf von:
GIVAYF $B391
Wandelt die 2-Byte-Ganzzahl in Y/A (Bereich —32768 bis +32767) in FLPT um und setzt sie in FAC1.

Fortsetzung im nächsten Extra