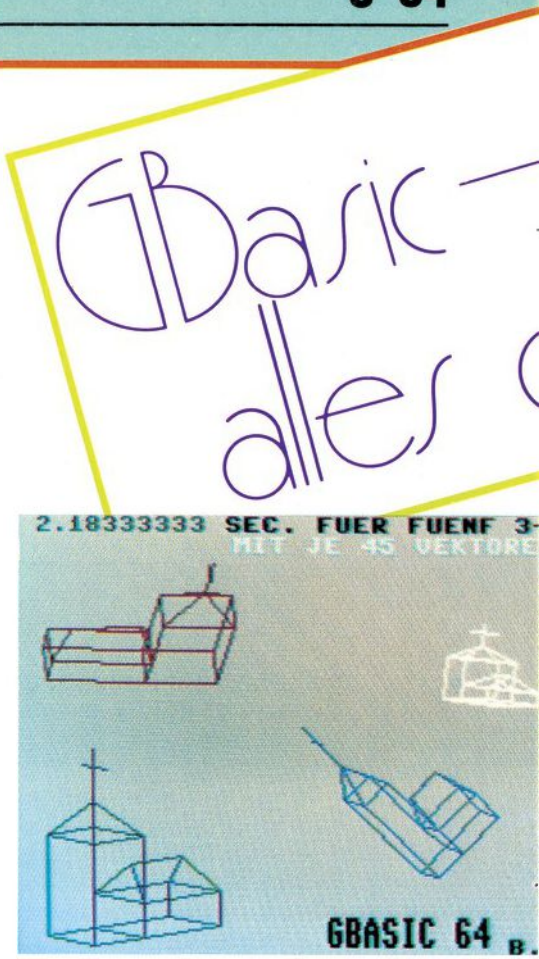
GBasic 64 ist eine relativ neue Basic-Erweiterung für den C 64. Sie ist wohl die leistungsfähigste, die es zur Zeit auf dem Markt gibt. GBasic 64 unterstützt sowohl Grafik als auch Sound. Verfechter der strukturierten Programmierung kommen ebenso zu ihrem Recht wie Anwender, die großen Wert auf Programmierhilfen legen. Dabei wird nicht nur Basic, sondern auch Assembler unterstützt.



GBasic ist noch relativ unbekannt, wahrscheinlich auch deswegen, weil es nur in einer Modul-Version vorliegt, die eine spezielle Hardware beinhaltet, und es somit praktisch kopierunfähig macht.

Das Modul macht einen sauberen soliden Eindruck. Goldkontakte, Standfüße, sowie einen eingebauten Reset-Taster sollte eigentlich jedes Modul haben. Mitgeliefert werden noch ein Handbuch, auf das ich noch zu sprechen kommen werde, sowie eine Diskette oder Kassette mit Demo-Programmen und Utilities. Nach dem Einstecken des Moduls in den Modulschacht und dem Einschalten des C 64 meldet sich dann auch sofort GBasic. Aus der Meldung wird ersichtlich, daß Sie nun 8 KByte weniger Speicherplatz für Basic-Programme zur Verfügung haben. Das Modul selbst hat allerdings 16 KByte ROM-Speicher. Daß trotzdem nur die Hälfte davon Basic-Speicherplatz belegt, liegt an einer ausgeklügelten Technik im Modul, die als »Memory-Mapping« bezeichnet wird. Modulintern werden die Speicherplätze so verwaltet, daß immer nur 8 KByte von GBasic für den Computer sichtbar sind. Sogar diese 8 KByte kann man sich freigeben lassen. Der Befehl EXIT schaltet nämlich das Modul softwareseitig aus! Nach EXIT haben sie wieder einen ganz normalen C 64 mit 38 KByte Basic-Speicher, ohne das Modul herausziehen zu müssen. Das schont sowohl Computer wie auch Modul. Einschalten läßt sich GBasic jederzeit wieder über den eingebauten Reset-Taster.

Tabelle 1 zeigt eine Befehlsübersicht, daher werden wir im folgenden nur die Besonderheiten von GBasic gegenüber anderen Erweiterungen aufzeigen.

Toolkit

GBasic besitzt einige Funktionen, die beim Programmieren sehr nützlich sind: Programmlistings können sowohl nach unten als auch nach oben gescrollt werden. Somit ist ein komfortables Editieren von Programmen möglich. Zwei nützliche und oft hilfreiche Befehle des Toolkits sind FIND- und der REN(umber).

FIND listet alle Basic-Zeilen, die eine bestimmte Zeichen- oder Befehlsfolge enthalten. Allerdings ist es nicht möglich, die Suche auf bestimmte Programmteile zu begrenzen. Bei häufigem Auftreten des gesuchten Begriffs leidet die Übersichtlichkeit etwas.

Dagegen ist der REN-Befehl komfortabel. Es werden beim Umnumerieren nicht nur alle GOTO, GOSUB, RESTORE-Befehle an die neuen Zeilennummern angepaßt, es kann auch bereichsweise umnumeriert werden. Wenn man zum Beispiel in einem Unterprogramm von Zeile 1000-2000 Änderungen vorgenommen hat, kann man es mit REN 1000,10,1000-2000 sauber in Zehnerschritten numerieren, während das restliche Programm bis auf die Anpassung der Sprungbefehle unverändert bleibt.

Positiv aufgefallen ist auch die hohe Geschwindigkeit beim TRACE-Befehl, die mit der Control-Taste zum Mitlesen herabgesetzt werden kann.

Extended Basic

Unter diese Kategorie fallen die Befehlserweiterungen, die für alle Programme nützlich sind, und sich somit nirgends richtig einordnen lassen. Benutzer von Simons oder ExBasic werden hier wohl den einen oder anderen Befehl ihrer Erweiterung wiedererkennen. Hervorstechend bei GBasic ist, daß die Befehle GOTO, GOSUB wie auch RESTORE nun auch mit Labels möglich sind. Diese Labels werden vor eine anzuspringende Zeile gesetzt. Schön wäre es noch gewesen, wenn man, wie in Simons Basic, globale und lokale Variablen zur Verfügung hätte.

Befehle wie IF..THEN..ELSE..,REPEAT.UNTIL, PRINT AT, PRINT USING, das Abfangen von ERRORs und so weiter, gehören schon fast zur »Standardausrüstung« von Basic-Erweiterungen und sind auch in GBasic enthalten.

Stringbefehle und Funktionen

Die Stringbefehle, die es in GBasic gibt, gleichen in ihrer Wirkung denen in Simons Basic. Man braucht keinen Haufen Unterprogramme, um Strings zu ersetzen, zum Einfügen und Suchen von Textteilen innerhalb eines Strings und so weiter. Neben dem EXOR-Befehl, der durch eine unübliche und unhandliche Syntax auffällt, wäre der BIT-Befehl zu erwähnen. Wenn man bestimmte Bits, zum Beispiel am Joystickport, auf gesetzt testen will, ist das mit einem Befehl möglich. BIT ist eine logische Funktion und kann somit direkt in IF-Abfragen eingesetzt werden.

Bild 1. Das alles kann der PRINT-Befehl ▼

◄ **Bild 2. 3-dimensionale Figuren lassen sich mit einfachen Befehlen drehen**

Mit BLOAD können Maschinenprogramme, Grafikbilder oder Spritedaten, kurz alles was nicht Basic ist, in den C 64 geladen werden. Wenn Sie sagen, »Das kann ich mit LOAD,X,1 auch«, dann werden Sie merken, daß im Gegensatz dazu BLOAD keinerlei Pointer verändert! Ein BLOAD-Befehl in einem Programm führt nicht zu einem Neustart nach erfolgtem Ladevorgang, sondern das Programm läuft ganz normal weiter. Auch der häßliche OUT OF MEMORY ERROR nach dem Nachladen von Maschinenprogrammen in den $C-Bereich tritt nicht mehr auf.

Noch ein paar Worte zum HCOPY-Befehl. Mit ihm kann hochauflösende Grafik ausgedruckt werden. HCOPY arbeitet mit jedem (!) grafikfähigen Acht-Nadel-Drucker zusammen. Diese Flexibilität mußte damit erkauft werden, daß Sie HCOPY erst mit einigen POKEs an ihren Drucker anpassen müssen. Eine Hardcopy der Textseite ist mit HCOPY nicht möglich.

Als sehr flexibel und nützlich erweist sich der FUNCTION-Befehl. Mit ihm kann der Inhalt eines Strings berechnet werden. So kann man mit einem INPUT-Befehl Formeln eingeben, die FUNCTION auswertet. Im normalen Basic ist das nur mit großen Schwierigkeiten möglich. Zum Beispiel ergibt PRINT FUNCTION "12/4+3" die Ausgabe 6. FUNCTION arbeitet mit allen Basic-Funktionen und auch Variablen im Funktionsstring. Erleichtert wird die Programmierarbeit auch dadurch, daß jederzeit Hexadezimal- und Binär-Zahlen verwendet werden dürfen.

Peripherie

GBasic unterstützt nicht nur Diskettenoperationen, sondern auch Joystick, Lightpen und Paddle. Sehr interessant sind dabei DEV und BLOAD. DEV stellt die Standardgeräte-Adresse um, DEV8 zum Beispiel auf die Floppy. Jedes LOAD, SAVE und VERIFY bezieht sich jetzt, so lange nicht ausdrücklich anders angegeben, auf die Floppy-Disk. Sogar die Tastenkombination SHIFT-RUN/STOP lädt das erste Programm anstelle von der Datasette von der Floppy und startet es automatisch.

Grafik

Kommen wir nun zu dem Teil von GBasic, der die meisten von Ihnen interessieren wird: die Grafik. GBasic verfügt über drei Grafikseiten, von denen zwei allerdings Basic-Speicherplatz belegen. Eine vierte kann noch als Zwischenspeicher verwendet werden. Bei ihr ist allerdings keine Farbsetzung möglich, da der Farbspeicher hier im GBasic-Modul selbst liegen müßte.

Zwischen diesen drei Seiten kann beliebig hin und her geschaltet werden, man kann sogar eine Seite bearbeiten, während eine andere oder die Textseite angezeigt wird. Es ist aber nicht möglich eine Multicolorseite anzeigen und eine »normale« Seite zu bearbeiten. Die Seiten können auch beliebig gemischt oder übereinander kopiert werden. Alle üblichen Zeichenbefehle sind implementiert: Punkte, Linien, Rechtecke und Blöcke können in sehr hoher Geschwindigkeit gesetzt, gelöscht oder invertiert werden.

Auch der CIRCLE-Befehl ist sehr vielseitig. Nicht nur Kreise, sondern auch Ellipsen, Vielecke, Kreisbögen und so weiter können gezeichnet werden. Dies allerdings in einer fast sensationellen Geschwindigkeit, vor allem, wenn man die Voreinstellung der Schrittweite ändert und erhöht. Die Kreise werden dadurch eckiger, aber sehr viel schneller gezeichnet. Hier hilft Experimentieren, bis man den optimalen Wert herausgefunden hat.

Ungewöhnlich, aber sehr nützlich ist der VECTOR-Befehl. Mit ihm kann man Linien zeichnen, die enden, sobald sie auf einen anderen gesetzten Punkt beziehungsweise Kreis, Linie oder sonstiges treffen. Schraffuren sind damit ein Kinderspiel. Der letzte gezeichnete Punkt wird in zwei Variablen gespeichert, so daß Sie volle Kontrolle über den Zeichenvorgang haben.

Nun aber zum gelungensten Befehl in GBasic. Gemeint ist PRINT. Ja, Sie haben richtig gelesen, PRINT. Sollten Sie jetzt den Kopf zweifelnd schütteln, so muß ich Ihnen sagen, daß ich in keiner Basic-Erweiterung einen solchen Komfort gefunden habe. Vergessen Sie TEXT (zum Beispiel aus Simons Basic) und seine Kollegen, mit denen Sie Texte in Grafiken eingebracht haben. Warum nehmen Sie nicht einfach PRINT? In GBasic funktioniert das, und zwar so gut, daß PRINT"shift-clr/home" tatsächlich die Grafikseite löscht. Alle Cursorbewegungen, Farbcodes, Groß/Klein-Umschaltungen und Revers-Schrift funktionieren wie auf der normalen Textseite, mit zwei kleinen Einschränkungen. Erstens ist das Ganze verständlicherweise etwas langsamer, und zweitens ist kein Scrolling nach oben möglich, wenn in die letzte Zeile geschrieben wird.

Doch damit nicht genug. Sie können mit PSN den Cursor fein, das

heißt einzelpunktweise, positionieren. Nach PSN 0,1 steht der Cursor (bitte bedenken Sie, daß er während der Programmausführung nicht zu sehen ist) um eine Einzelpunktzeile tiefer als nach PRINT"clr/home" und am linken Rand. Mit PSN ist zum Beispiel Schrägschrift oder ein Funktionenplotter mit Print-Befehlen möglich. Eine weitere Manipulationsmöglichkeit ist mit SIZE gegeben. Sie können damit die Zeichengröße in X- und Y-Richtung beliebig vervielfachen, bis aufs 38x24 = 912-fache. Das wäre dann allerdings bildschirmfüllend.

Schließlich kann noch der Zeichenabstand manipuliert werden, dies allerdings nur mit POKE-Befehlen. So läßt sich eine Engschrift, oder gar Proportionalschrift realisieren. Alle diese Möglichkeiten des PRINT-Befehls habe ich in Bild 1 ausgeschöpft. Aber das ist immer noch nicht alles. Mit der GBasic-Demo-Diskette/Kassette erhalten Sie noch eine Befehlserweiterung zu GBasic, die auf einfache Art und Weise 3D-Grafik ermöglicht. Dazu müssen Sie ersteinmal Shapes erstellen und im Speicher ablegen. Ein Shape ist, salopp gesagt, ein dreidimensionaler Linienzug beliebiger Länge und Komplexität. Haben Sie solch ein Shape erstmal definiert, können sie es mit drei zusätzlichen Befehlen beliebig um die Raumachsen drehen beziehungsweise vergrößern, um es danach im hochauflösenden Grafik-Modus zeichnen zu lassen. Gezeichnet wird aus Geschwindigkeitsgründen parallelperspektivisch im Drahtgittermodell, das heißt, eigentlich unsichtbare Linien werden mitgezeichnet. Ein Beispiel sehen Sie in Bild 2.

Sprites

Ähnlich reichhaltig wie der Grafikbefehlssatz ist der für die Sprites. Das fängt schon mit einem ebenfalls auf der Demo-Diskette/Kassette befindlichen Spriteeditor an, dessen Spritedaten Sie in GBasic direkt weiterverarbeiten können. Alle Spriteparameter sind über Befehle erreichbar. Es ist auch möglich, Sprites interruptgesteuert und unabhängig vom Basic-Programm bewegen zu lassen. Sprite-Sprite- und Sprite-Hintergrund-Kollisionen können auch jederzeit in GBasic abgefragt werden. Dies ist allerdings auch interruptgesteuert möglich, bei einer Kollision wird dann in ein Basic-Unterprogramm gesprungen. Hier kann sogar nur auf bestimmte Kollisionen, wie Sprite drei mit Sprite sieben oder ähnliches, geprüft werden.

Toolkit

AUTO	—Automatische Zeilennummern
REN	—Zeilenumnumerierung
DEL	—Löscht Programmbereiche
FIND	—Suche nach Ausdruck im Programm
DUMP	—Ausgabe aller Variablen
TRACE	—aktuelle Prg.-Zeile anzeigen
OLD	—Rückgängigmachen von NEW
KEY	—Funktionstasten belegen
DISP	—Funktionstastenbelegung anzeigen
EXIT	—Schaltet GBasic-Modul aus

Extended Basic

PAUSE	—wartet einen angegebenen Zeitraum
ELSE—	Für IF.THEN..ELSE..
LASTIF	—Letzten IF-Befehl weiterführen
REPEAT.UN-TIL	—Schleife mit Schlußbedingung ähnlich PASCAL
POP	—vorzeitiger Ausstieg aus Schleife oder Unterprogramm
LBL	—legt Label für Sprunganweisung/RESTORE fest (GOTO/GOSUB/RESTORE "TEST")
CASE ERR	
GOTO	—»Abfangen« von Fehlermeldungen
OFF	—schaltet CASE ERR GOTO ab
RESTORE	—funktioniert nun auch mit Zeilennummern
PRINT USING	—Formatierte Zahlenausgabe
VTAB	—Cursor in Bildschirmzeile setzen
SWAP	—Vertauscht zwei Variableninhalte

Strings

INSTR	—Suchen eines Strings in einem anderen
REPL$	—Teilweises Überschreiben/Ersetzen eines Strings
INST$	—Einfügen eines Strings in einen anderen
MULT$	—Vervielfachen eines Strings

Eingabe

FETCH$	—Verbessertes GET
INLINE$	—Verbesserter INPUT
BUT	—Funktionstasten abfragen

Funktionen

FUNCTION	—Berechnen eines Strings
FRAC	—Vorkommastellen einer Zahl abschneiden
MOD	—Modulo-Funktion, Rest einer Division
EXOR	—Verknüpft zwei Zahlen logisch exclusiv-oder
BIT	—Testet auf bestimmte Bits
$	—Kennzeichnet Hexadezimalzahl
%	—Kennzeichnet Binärzahl
HEX$	—wandelt Dezimal in Hexadezimal
BIN$	—wandelt Dezimal in Binär

Speicher

DOKE	—POKE für Adreßwerte im HI/LO-Format
DEEK	—PEEK für Adreßwerte im HI/LO-Format
LOMEM	—legt Untergrenze des BASIC-Speichers fest
HIMEM	—legt Obergrenze des BASIC-Speichers fest

Peripherie

DEV	—stellt die Standardgeräteadresse um
DIR	—zeigt Disketten-Directory an
DISK	—sendet ein Kommando an die Floppy-Disk
ERR	—liest den Kommandokanal aus
MERGE	—hängt ein Programm an ein anderes an
BLOAD	—LOAD ohne Veränderung von Variablen
BSAVE	—Abspeichern eines Speicherblockes
HCOPY	—Hardcopy der hochauflösenden Grafik
JOY	—Abfrage eines der beiden Joysticks
PEN	—Abfrage eines Lightpen
PDL	—Abfrage eines der vier Paddles

Grafik

HGR	—schaltet Grafik ein (normal/Multicolor)
TEXT	—schaltet auf Textbildschirm zurück
COLORG	—Farben für die hochauflösende Grafik
COLORT	—Farben für die Textseite
COL	—Zeichenfarbe ändern.
INK	—Zeichenfarbe wählen.
MODE	—Betriebsart in hochauflösender Grafik: zeichnen, löschen, invertieren, zählen
SCREEN	—Anwahl des verwendeten Grafikschirms
ADD	—überlagert oder kopiert Grafikschirme
PLOT	—zeichnet Punkt
LINE	—zeichnet Linie
VECTOR	—zeichnet Linie bis zum nächsten Punkt
CIRCLE	—zeichnet Kreise, Ellipsen, Kreisbögen etc.
BOX	—zeichnet Rechteck
BLOCK	—zeichnet Balken
FILL	—Ausfüllen einer umrandeten Fläche
PRINT	—zeichnet Text/Grafikzeichen in der HGR
SIZE	—verändert die Zeichengröße für PRINT
PSN	—Feinpositionierung der Texte in HGR
MEM	—Umschaltung Zeichensätze original/eigener

Sprites

SEDIT	—ruft den Spriteeditor auf
SPRITE	—legt das Aussehen eines Sprites fest
SCOL	—Spritefarben festlegen
SPOS	—setzt ein Sprite an eine bestimmte Position
SMOV	—Interruptgesteuerte Bewegung von Sprites
SPRX/SPRY	—Position eines Sprite feststellen
CHECK	—Test auf Kollision Sprite-Sprite/Hintergrund
COND.GOSUB	—Interruptgesteuerte, dauernde Überwachung von Kollisionen, bei Kollision GOSUB
RTN	—Rückkehr aus COND.GOSUB
SOFF	—Sprite wieder abschalten

Musik

VOL	—Lautstärke einstellen
ENVELOPE	—Hüllkurve einer Stimme einstellen
WAVE	—Wellenform, Ring., und Sync. einstellen
WIDTH	—Pulsbreite bei Rechteckschwingung
SEFILT	—Filter einstellen
FILTER	—Stimmen auf Filter leiten
SND	—Umrechnung Notenname-Frequenz
TUNE	—Ton spielen
PLAY	—Musikstück interruptgesteuert spielen
VOFF	—Stimme abschalten

Monitor

TIM	—ruft den eingebauten Monitor auf. —Möglichkeiten Load, Save, Verify Assemble, Disassemble Hex-, ASCII-, Binäry-, Dezimal-Dump & Eingabe Verschieben mit/ohne Adressenangleichung Suchen beliebiger Zeichenfolge mit Joker Trace für Maschinenprogramme Berechnungen und Variable im Monitor

Musik

Der letzte Bereich von neuen Basic-Befehlen betrifft den Sound-Chip. Auch hier zeigt sich GBasic als praktisch. Es kann auf jeden Parameter des Sound-Chips zugegriffen werden, ohne zu POKEn. Sogar der Filter ist in GBasic voll steuerbar. Aber damit nicht genug. Für Soundeffekte reichen die normalen Soundbefehle voll aus, aber sobald man längere Musikstücke spielen möchte, hört der Spaß auf. Deswegen wurde in GBasic noch eine Programmiersprache, MUSIC, integriert. MUSIC hat 14 Befehle, die das Programmieren von Musikstücken zum Kinderspiel machen. Diese sind nicht in Tabelle 1 aufgeführt! Ein in MUSIC programmiertes Lied wird irgendwo im Speicher abgelegt. Es kann dann mit dem PLAY-Befehl gestartet werden und läuft interruptgesteuert bis zum Ende oder endlos, je nach Programmierung. Während die Musik spielt, kann ein Basic-Programm unbehindert nebenher laufen. Die Eingabe von MUSIC-Programmen würde allerdings wieder zur POKEerei werden, wenn Omikron nicht auch noch einen MUSIC-Editor mitliefern würde. Der ist ebenfalls auf der Demo-Diskette/Kassette.

Monitor

Und um das Maß und die 16 KByte Speicher voll zu bekommen, ist in GBasic auch noch ein Maschinensprachemonitor implementiert. Er beherrscht so ziemlich alles, was man von einem komfortablen Monitor erwartet, den man sich einzeln, also ohne Basic-Erweiterung zulegen würde. Einziger Kritikpunkt ist die ungewöhnliche Bedienung über die Funktionstasten. Deswegen wurde auch hier keine genaue Befehlsauflistung gegeben. Alle Fähigkeiten des Monitors finden sich aber in Tabelle 1. Dieser Monitor ist wohl für jeden normalen Anwendungsfall ausreichend.

Dokumentation

Das mitgelieferte Handbuch muß hier ausdrücklich gelobt werden. Es werden nicht nur alle Befehle gut

◄ **Tabelle 1. Alle GBasic-Befehle auf einen Blick**

und genau erklärt, auch sind sehr viele Hintergrundinformationen enthalten, wie zum Beispiel eine Speicherbelegung von GBasic. Es wird auf die mitgelieferten Programme genauso stark eingegangen wie auf GBasic selber. Eine Befehlsübersicht und einige wichtige Tabellen runden das Gesamtbild ab. Ich vermisse nur noch eine Zeropagebelegung von GBasic. Warum? Nun, GBasic läßt sich vom Benutzer beliebig erweitern. Wenn Sie einen neuen Befehl in GBasic einbauen wollen, müssen Sie ihm nur ein "!" voranstellen. Findet GBasic ein "!", springt es nach $C000, wo Sie dann Ihre Befehlsauswertung vornehmen können. Ein Beispiel für eine solche Erweiterung ist die nachladbare 3D-Grafik.

Wie Sie sehen, bietet GBasic eine ganze Menge, und zwar nicht nur von jedem etwas, sondern jeder seiner Befehlsbereiche ist in sich geschlossen und vollständig. Beim Test habe ich nur einen Fehler entdecken können. Wenn Sie mit DISK ein Kommando an die Diskette senden wollen, diese aber nicht eingeschaltet ist, hängt sich GBasic auf, und läßt sich nur noch mit RUN/STOP-RESTORE wiederbeleben. Ähnliches passiert bei DIR. Es wird kein DEVICE NOT PRESENT ERROR ausgegeben, sondern einfach nur READY. Diese beiden Fehler sollen aber in einer neuen Version, die gerade bei Omikron fertiggestellt wird, beseitigt werden. Zusätzlich will man dort ein schnelleres Laden von Diskette, ähnlich Hypra-Load (64'er, 10/84), einbauen. Trotzdem soll volle Kompatibilität zu der alten GBasic-Version bestehen bleiben.

Abschließend kann ich sagen, daß mich GBasic überzeugt hat. Ausschlaggebend waren die enorme Befehlsvielfalt, wie auch die hohe Geschwindigkeit der meisten Funktionen. Ich hoffe, daß diese Erweiterung etwas bekannter, und vielleicht, ähnlich wie für Simons Basic, fertige Software dafür geschrieben wird. Auch die Erweiterbarkeit von GBasic kann ein Anreiz für andere Programmierer sein. Insgesamt ein Paket, das seine 259 Mark wert ist.

(Boris Schneider/gk)

Bezugsquelle:
Omikron Software, Erlachstr.15, 7534 Birkenfeld 2

ASSEMBLER?

Mein Weg zur Maschinensprache begann mit den vielen, vielen DATAs, die sicherlich jeder von uns einmal eingegeben hat, ohne überhaupt zu wissen, welche Bedeutung sie hatten. Diese Unmenge an Zahlen verschwand dann in der Tiefe des Rechners, wurde nicht mehr gesehen und vollbrachte wahre Wunderdinge.

Also entschloß ich mich eines Tages, einen in Basic geschriebenen Maschinensprach-Monitor abzutippen und damit zu arbeiten. Plötzlich erschienen gar seltsame Zahlen und Buchstaben auf dem Bildschirm:
4000 A0 00 B9 00 41 F0 06 20

So ungefähr muß es in grauer Computer-Urzeit auf meinem Bildschirm ausgesehen haben.

Nun, Sie müssen zugeben, daß diese »Hieroglyphen« sicherlich nicht gerade aufschlußreich erscheinen. Mit einer Befehlsliste für den 6502 ging ich nun daran, das »Zahlenrätsel« zu lösen. Zum ersten Mal konnte ich meinem Computer direkt in das Innerste — also geradezu in die Eingeweide — schauen. Allerdings war dieses Verfahren der Übersetzung auf die Dauer sehr eintönig und sehr, sehr zeitraubend. Also wie geschaffen für einen Computer.

Sehen wir uns noch einmal das obengenannte Beispiel genauer an:

Bei der ersten Angabe (4000) handelt es sich um die Adresse, in der der folgende Wert (A0) steht. Dies ist ein Befehl oder »Operationscode«, der eine Operation in der CPU des Rechners bewirkt: Er lädt das Y-Register (im Englischen abgekürzt — LOAD Y) mit der folgenden Zahl (00).

Da »unser« C 64 einen Befehlsvorrat von 50 Befehlen kennt, können Sie sich vorstellen, daß es sehr anstrengend ist, sich diese Anzahl an Befehlen in Form von Hex-Zahlen zu merken. Hinzu kommt noch, daß durch die verschiedenen Adressierungsarten die Gesamtzahl auf 150 Befehle ansteigt, so daß es selbst für einen Geistesakrobaten schwierig wird. Deshalb kamen findige Tüftler auf die Idee, Mnemonics (= Gedächtnishilfen) zu »erfinden«, die in 3 Buchstaben das Entscheidende des Befehls ausdrücken:
Aus A0 (LOAD Y) wird als Mnemonic LDY. Aus B9 (LOAD AKKU) wird LDA!

Wenn wir uns jetzt diese Erleichterung zu Nutze machen, können wir unser Maschinen-Programm auch folgendermaßen schreiben:

```
LDY   #00
LDA   4100,Y
BEQ   400D
JSR   FFD2
```

Nun fällt uns auch die Übersetzung des Programmes viel leichter:
1. lade das Y-Register mit dem Wert 00,
2. lade den Akku mit dem Wert, der in Adresse 4100+Y steht,
3. springe, wenn dies eine Null war, nach 400D,
4. springe ansonsten in das Unterprogramm (JSR = JUMP SUBROUTINE), das bei FFD2 beginnt.

Sie werden bemerkt haben, daß wir uns immer mehr von der »reinen« Maschinensprache entfernen, da wir bemüht sind, das Programmieren für uns Menschen verständlicher zu machen.

Wir benötigen also ein Programm, welches in der Lage ist, einen so eingegebenen Text zu übersetzen. Solch ein Programm, das einen in Mnemonics (zum Beispiel LDA..., STA...) vorliegenden Text in Maschinencode übersetzt, nennt man einen **Assembler.**

Entsprechend wird ein umgekehrt arbeitendes Programm als **Disassembler** bezeichnet. Dieser hat also den Vorteil, daß er unsere Programmzeile nicht in reinen Hexadezimalzahlen (A0, B9, ...), sondern in Mnemonics (LDY #00, LDA 4100, Y ...) ausdruckt.

Leider heißt auch die Programmiersprache, bei der Mnemonics benutzt werden, **Assembler(-Sprache),** so daß hierbei leicht Verwechslungen auftreten.

Neben dieser reinen Übersetzungstätigkeit besitzen fast alle Assembler(-Programme) aber noch weitere Möglichkeiten, die Arbeit zu erleichtern. Alle guten Assembler erlauben den Einsatz von **Labels**. Dabei handelt es sich um freigewählte Namen, die anstelle der absoluten Werte gesetzt werden. In unserem Beispiel könnten wir die absoluten Adressen hinter LDA und JSR durch einen beliebigen Namen ersetzen. Unser Beispiel könnte also etwa so aussehen:

```
        LDY   #
LOOP    LDA   TEXT,Y
        BEQ   ENDE
        JSR   AUSGEBEN
        INY
        BNE   LOOP
ENDE    ...   ...
```

Ein Vorteil dieser Prozedur liegt vor allem darin, daß diese Namen