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

ASSEMBLER!

Wer die Fähigkeiten seines Computers voll und ganz ausnutzen möchte, kommt um den Einsatz der Maschinensprache nicht herum. Schließlich ist sie die einzige, die der Computer wirklich versteht. Allerdings hat sie den entscheidenden Nachteil, daß der Mensch mit einer Anhäufung von Nullen und Einsen kaum etwas anfangen kann. Kluge Köpfe haben deshalb die Assembler-Sprache erfunden, einen Kompromiß, mit dem beide, Mensch und Computer, zufrieden sein können.

schon beim Lesen des Quelltextes erkennen lassen, welche Funktion zum Beispiel das Unterprogramm »Ausgeben« hat. Anders als in Basic, wo ein GOSUB 12600 nichts darüber aussagt, was dort geschehen soll. Ein weiterer Vorteil ist, daß wir uns beim Schreiben des Textes (auch als **Quelltext,** Quellcode beziehungsweise Sourcecode bezeichnet) nicht von vornherein über sämtliche Sprungziele im Klaren sein müssen. Wo zum Beispiel der Text steht, ist erst einmal völlig gleichgültig. Wir könnten auch noch später einige Befehle zwischen LOOP und ENDE einfügen, kurz, der Quelltext ist ohne großen Aufwand beliebig veränderbar. Wären statt dessen absolute Adressen verwendet worden, müßten diese bei jeder Änderung ebenfalls angepaßt werden. Bei längeren Programmen ist das nahezu unmöglich. Die Rechnerei übernimmt nun der Assembler: Dazu arbeitet er den Quelltext in — normalerweise — zwei Schritten **(Pass)** durch. Im ersten Pass werden alle Sprungziele etc. berechnet und in einer **Symboltabelle** abgelegt, im zweiten Pass wird das Maschinenprogramm (**Objectcode,** Maschinencode) im Speicher abgelegt.

Wenn in einem Programm verschiedene Teile mehrfach auftauchen, ist es sicherlich nicht sinnvoll, diese jedesmal neu eingeben zu müssen. Deshalb bieten verschiedene Assembler die Möglichkeit, solche Teile als **»Makro«** zu definieren. Anstelle der gesamten Befehlsfolge genügt es, nur den vorher definierten Makronamen einzusetzen. Unser Beispielprogramm ließe sich mit dem Makro-Namen »Textaus« versehen und dann an jeder Stelle im Programm aufrufen, die einen Text ausgeben soll.

Als weitere Bequemlichkeit nehmen die meisten Assembler dem Programmierer das Umrechnen der verschiedenen Zahlensysteme ab. Sie verarbeiten Binärzahlen ebenso wie Dezimal- oder Hex-Zahlen. Häufig kann man auch Texte als Buchstabenfolge und nicht in ASCII-Codes eingeben. Darüber hinaus sind beim Operanden einfache Rechen- beziehungsweise logische Operationen erlaubt, zum Beispiel:

```
LDA   #53280 AND 255
oder
STA   TEXT+7,X
```

Um dem Assembler Anweisungen beim Übersetzen zu geben, werden **Pseudo-Opcodes** eingesetzt. So bedeutet zum Beispiel:

```
.BA $C000
```

Beginne die Assemblierung bei $C000. (Bisweilen wird hierfür auch der Pseudo-Opcode »ORG« benutzt. ».BY« oder ».WORD« signalisiert dem Assembler, daß die folgenden Zeichen nicht als Befehl, sondern als Byte-Folge im Speicher abgelegt werden sollen. Ebenso gibt es Pseudo-Opcodes, die die Ausgabe eines Assemblerlistings auf dem Drucker steuern. In der Anzahl und den Möglichkeiten der Pseudo-Opcodes unterscheiden sich die verschiedenen Assembler stark voneinander.

Zur Eingabe des Quelltextes wird ein **Editor** benötigt. Im einfachsten Falle ist dies der von Basic gewohnte Editor. Gute Assembler bieten hier jedoch weitaus mehr. Befehle zum Suchen (FIND) und Ändern (EDIT) bestimmter Befehlsfolgen gehören eigentlich zum Standard. Befehle zum Blättern im Quelltext, zum Einfügen, Kopieren und Verschieben von Textteilen, wie man sie aus Textverarbeitungssystemen kennt, bieten ein Höchstmaß an Komfort. Auch hierin unterscheiden sich die Assembler ganz erheblich. Der Editor erlaubt meistens auch, Teile des Programms separat abzuspeichern und sich so eine Bibliothek von häufig benötigten Unterprogrammen anzulegen. Damit wären die wichtigsten Begriffe, die im Zusammenhang mit Assemblern auftauchen (zum Beispiel in den Testberichten in dieser Ausgabe) erläutert. Eine Zusammenfassung finden Sie in unserem »Kleinen Assemblerlexikon«.

(N. Mann/D. Weineck/gk)

Kleines Assemblerlexikon

Monitor
Programm zum Bearbeiten von Maschinenprogrammen. Enthält mindestens HEX-DUMP, DISASSEMBLER sowie LOAD- und SAVE-Befehle. Bessere Monitore bieten zum Beispiel die Möglichkeit, Programme unter Kontrolle der Register ablaufen zu lassen.

Hex-Dump
(Memorydump, Speicherdump). Auflisten eines Maschinenprogrammes in Hex-Zahlen.

Disassembler
listet ein Maschinenprogramm in Form von MNEMONICS auf.

Assembler
übersetzt einen in Assemblersprache geschriebenen QUELLTEXT in OBJECTCODE.

Mnemonic
Aus drei Buchstaben bestehende »Gedächtnishilfe«, Abkürzung eines Assemblerbefehls.

Label
(Marke), ein (sinnvoller) Name, der bei der Assemblierung in eine feste Zahl umgerechnet wird. Labels können Sprungziele, aber auch Operanden sein.

Makro
Häufig auftretende Programmteile werden unter einem Namen zusammengefaßt. Der Quelltext enthält nur den Namen, beim Assemblieren wird die Befehlsfolge eingesetzt.

Quelltext
(Sourcecode) ist der Text, den man mit Hilfe des EDITORS in Assembler-Sprache schreibt. Er wird später vom Assembler übersetzt.

Objectcode
(Maschinencode) ist das vom ASSEMBLER erzeugte, fertige Maschinenprogramm.

Pseudo-Opcode
enthält Anweisungen an den Assembler.

Symboltabelle
wird vom Assembler im ersten Pass erstellt. Sie enthält die absoluten Adressen aller LABELs.