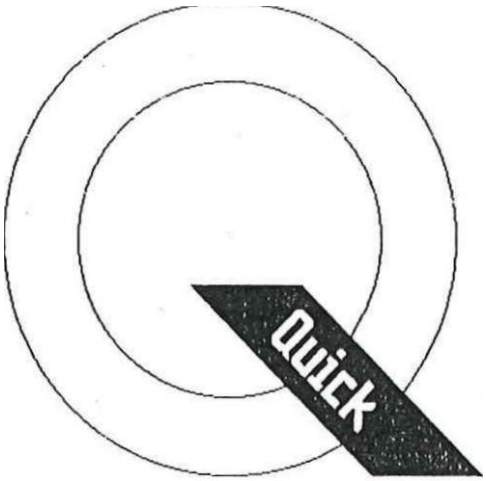
Das

QUICK

Programmierhandbuch



programmiert von Andreas Binner und Harald Schönfeld

Herzlich willkommen bei QUICK,

der neuen Programmiersprache für die ATARI XL/XE-Computer. QUICK wurde exklusiv für Sie im Auftrag des Verlag Werner Rätz von Andreas Binner und Harald Schönfeld entwickelt.

QUICK, das ist die neue Sprache, die fast so schnell und leistungsfähig wie Assembler ist, aber trotzdem auch annähernd so komfortabel und einfach wie BASIC!

In diesem Handbuch werden wir Ihnen nicht nur die Sprache vorstellen und deren Benutzung erklären, sondern wir wollen Ihnen auch mit Tipps und Tricks zur Seite stehen. Wir gehen aber davon aus, daß Sie sich bereits mit anderen Programmiersprachen auf den ATARI XL/XE-Computern beschäftigt haben, so daß es nicht nötig ist, auf völlig grundlegende Dinge einzugehen.

Viel Spaß mit Quick wünschen:

RAINDORF-SOFT
Andreas Binner und Harald Schönfeld

Verlag Werner Rätz
Postfach 1640
7518 Bretten

EINLEITUNG

In diesem Kapitel möchten wir Ihnen einen Eindruck davon vermitteln, wie QUICK auf Ihrem ATARI implementiert wurde.

Von der Idee zum Konzept

Jeder der sich mit seinem ATARI Computer einige Zeit beschäftigt, stößt bald an die Grenzen des eingebauten Basics: Leistungsfähige Programme sind mit ihm nicht zu verwirklichen, weil es viel zu langsam ist und die umfangreichen Hardwarefähigkeiten (z.B. Interrupts) der ATARIs überhaupt nicht ausgenutzt werden können. Will man seinen Computer richtig ausreizen, so blieb bisher nur der Schritt zu Assembler. Diese "Programmiersprache" ist zwar schnell und leistungsfähig, aber die Programmierung ist mit einem immensen Zeitaufwand verbunden. Denn schon die Lösung einfacher Programmieraufgaben ist in Assembler mit aufwendiger Tüftelarbeit verbunden. Der Grund liegt darin, daß Maschinensprache eben keine Hochsprache ist, und somit keine fertigen Problemlösungen vorhanden sind. Schon so selbstverständliche Dinge wie PRINT, PLOT, SOUND, LOAD bedeuten in Assembler viel Arbeit und oft langwierige Fehlersuche. Und so resignieren viele und überlassen das Programmieren den anderen...
Deshalb lag die Idee nicht fern, eine Sprache zu entwickeln, die die Vorzüge von Assembler und Basic vereinigt: Sie soll schnell und universell einsetzbar sein, sie soll alle Hardwaremöglichkeiten der ATARIs zugänglich machen, aber sie muß auch komfortabel und strukturiert zu programmieren sein. Bevor wir uns aber an die Definition der Sprache machen konnten, mußte erst deren Haupteinsatzgebiet geklärt werden. Wir entschieden uns, die Sprache auf die Programmierung von Spielen, Sounds, Grafiken und allen damit verbundenen Anwendungen (also z.B. durchaus auch Grafik- und Textverarbeitungsprogramme) zuzuschneidern. Das ist der Grund dafür, daß es in QUICK Befehle zur Verschiebung von Graphikausschnitten, zum Spielen von digitalisierten Sounds, zur Darstellung von Playern und sogar zur Abfrage einer Maus gibt.
Auf der anderen Seite sind aber auch Befehlsstrukturen wie IF-ELSE-ENDIF oder REPEAT-UNTIL nötig. Einen weiteren Beitrag zur Strukturiertheit liefert die Möglichkeit der Variablendeklaration. D.h., es ist möglich verschiedene Arten von Variablen zu verwenden: Einerseits Integervariablen, also Variablen die nur ganze Zahlen darstellen können (dabei kann noch zwischen BYTE und WORD unterschieden werden) und ARRAYs d.h. eine Art Stringvariable. Ein wichtiges Merkmal ist dabei auch die Möglichkeit lokale Variable zu verwenden und an Prozeduren zu übergeben. Diese Variablen können dann nur in dem Unterprogramm verwendet werden, in dem sie deklariert wurden. Wie Sie sehen fehlen bisher die Fließkommavariablen (die in BASIC ausschließlich verwendet werden können, was ein Grund für die niedrige Geschwindigkeit ist). Diese Art von Variablen ist aber für die meisten Anwendungsgebiete überhaupt nicht notwendig! Aber was passiert wenn man nun doch ein Programm schreiben will, in dem Fließkommazahlen notwendig sind? Nun, jetzt sind wir auf die wichtigste Forderung an Quick gestoßen: Es muß leicht erweiterbar sein, d.h. die Definition eigener "Befehle" soll einfach möglich sein.

Und dann war noch ein entscheidender Punkt zu klären: Wie soll Quick in der Lage sein, so schnelle Programme zu produzieren? Dazu mußte Quick als Compilersprache konzipiert werden. D.h. zunächst schreibt man mit Hilfe eines Texteditors einen Quickquelltext, der dann vom Compiler in ein direkt laufähiges Maschinenprogramm übersetzt werden muß. Dabei ist es natürlich von entscheidender Bedeutung, wie diese Übersetzung in Maschinensprache gelingt. Befehle wie die Addition zweier Variablen oder das Einlesen von Daten werden dabei so gut übersetzt, daß es in Assembler auch nicht schneller gehen kann. Andere Aufgaben, z.B. Vergleiche müssen dagegen so flexibel sein, daß es sicher möglich ist in Assembler (wo man sich ja nur um einen bestimmten Fall kümmern muß) schneller zu sein.

Das QUICK-Programmiersystem

Wie Sie sehen, muß man also immer zwischen 2 Programmen (Editor, Compiler) hin und her wechseln. Damit das nicht zu lange dauert, dürfen diese Programme nur einmal zu Beginn geladen werden, und müssen dann jederzeit im Speicher vorliegen.
Es fehlt also noch ein drittes Programm, das den Editor und Compiler zu Anfang lädt, und dann die Verbindung der beiden darstellt: Die Shell. Damit der Editor und der Compiler aber dann nicht den ganzen Speicher belegen, müssen sie abwechselnd, wenn sie nicht gebraucht werden, in den freien Speicher hinter dem Betriebssystem kopiert werden. Auch dafür ist die Shell zuständig.

Die Erweiterbarkeit der Sprache wird durch Libraries (Unterprogrammbibliotheken) verwirklicht. Eine Library ist ein Quick-Quelltext, der eine Anzahl verschiedener Unterprogramme enthält. Einige Standardlibraries gehören von Haus aus zu Quick:
Die Grafik-Library (enthält Routinen zum Zeichnen verschiedener geometrischer Objekte und einen FILL-Befehl), die Mathe-Library (die die Fließkommaarithmetik einbindet) und einige andere. Aber auch Sie können Routinen in Quick schreiben, und in einer Library ablegen. Wird eine solche Routine dann in einem Programm verwendet, kann man den Compiler dazu veranlassen, die entsprechende Library zu laden.

Und so sieht das Quick-Programmiersystem dann also aus:

- **Editor**, in dem die Quelltexte geschrieben werden
- **Compiler**, der die lauffähigen Programme produziert
- **Shell**, die Editor und Compiler verbindet
- **Libraries**, die eine einfache Erweiterung ermöglichen

Jetzt wollen wir Sie aber nicht länger auf die Folter spannen: Hier der erste Quickquelltext:

```
MAIN
  PRINT ( "Hallo Welt")
ENDMAIN
```

Sie sehen, das hat durchaus Ähnlichkeiten mit Basic (oder "C").

Aber sind die Quickprogramme nun wirklich soo schnell? Dazu haben wir einen kleinen Benchmarktest geschrieben: Mit Hilfe des "Sieb des Eratosthenes" sollten die ersten 1889 Primzahlen ermittelt werden. Das entsprechende Basic-Programm benötigt dafür 363 Sekunden, während das QUICK-Programm bereits nach 6 Sekunden beendet ist!

Wir haben also nicht zu viel versprochen: Quick hat die Nase weit vorne.

Die Shell

Wir empfehlen Ihnen, sich zunächst mit Hilfe des DUP eine Sicherheitskopie der Systemdisk zu erstellen. Wir möchten Sie darauf hinweisen, daß das Kopieren der Systemdisk nur für ihren persönlichen Gebrauch gestattet ist!

Zum **Starten** ihres Quick-Programmiersystems legen Sie die Systemdisk in Laufwerk D1 und schalten den Computer bei gedrückter OPTION-Taste ein.

Nun wird zunächst die Shell geladen. Sie ist ein kurzes Programm, das direkt hinter dem **DOS 2.5** in den Speicher geladen wird. Nach dem Booten der Disk befinden Sie sich in der Shell, die zunächst den Editor und Compiler nachlädt. Nun können Sie die Systemdisk aus dem Laufwerk nehmen, und eine Arbeitsdisk einlegen, auf der Sie dann Programme abspeichern können.

Auf der Systemdisk befinden sich die Shell (AUTORUNSYS), der Compiler und der Editor. Da diese Programme sofort geladen werden, müssen diese auf einer Arbeitsdiskette (auf der Sie ihre Programme abspeichern wollen) nicht vorhanden sein.
Der Compiler benötigt zum Arbeiten jedoch in jedem Fall das RUNTIME.OBJ File. Unter Umständen werden auch die auf der Diskette vorhandenen Libraries (*.LIB) gebraucht. Aus diesem Grund müssen Sie all diese Programme auf die Arbeitsdiskette kopieren wenn Sie nur mit einem Laufwerk arbeiten. Falls Sie mit mehreren Disklaufwerken arbeiten, ist es dagegen praktischer einfach die Systemdisk im Laufwerk D1 zu belassen und im anderen eine völlig leere Arbeitsdisk zu verwenden.

Durch Eingabe von Editor oder Compiler rufen Sie das entsprechende Programm auf. Die Tastenkombination CONTROL-Q löst einen Kaltstart aus. während das Drücken von RESET immer in die gerade laufende Programmierumgebung zurückführt.

Der Editor

Der erste Schritt zur Erstellung eigener QUICK-Programme ist das Schreiben eines QUICK-Quelltextes mit Hilfe des Editors. Der Quelltext wird dann vom Compiler in ein lauffähiges Programm umgewandelt.

Der Aufbau des Editors

Der Editor unterscheidet sich in der Handhabung stark von den üblichen Zellennummern-orientierten Editoren für die XL-Computer, da er völlig ohne Zeilennummern auskommt. Bei der Benutzung werden Sie feststellen, daß das viele Vorteile hat. Ein anderes wichtiges Merkmal ist, daß der Editor ständig im Insert-Modus arbeitet, d.h. alle Zeichen, die Sie eintippen, werden an der Position des Cursors eingefügt, wobei die weiter rechts stehenden Zeichen verschoben werden. Der Editor arbeitet zeilenorientiert, d.h. pro Zeile können höchstens 38 Zeichen getippt werden. Am Ende der Zeile bleibt der Cursor automatisch stehen. Zu Beginn legt der Editor automatisch einen Text-Kopf an, in dem die Länge des Textes (zu Anfang natürlich 0 Bytes) und der freie Speicherplatz in Hex-Zahlen angezeigt werden. Sobald Sie bei SAVE einen Filenamen eintippen, wird dieser ebenfalls im Text-Kopf erscheinen.
In der untersten Zeile des Bildschirms wird nun "Edit" angezeigt.

Der Edit-Modus

Wenn Sie in diesem Modus sind, können Sie drauf los tippen. Dabei können Sie sowohl Groß- als auch Kleinbuchstaben, jedoch keine inversen Zeichen oder Grafikzeichen eintippen. Falls der Editor plötzlich keine Zeichen mehr anzunehmen scheint, so liegt das einfach daran, daß Sie die INVERS- oder CONTROL-CAPS-Taste gedrückt haben. In diesem Fall, müssen Sie den entsprechenden Modus durch nochmaliges Drücken wieder ausschalten.

Mit Hilfe der RETURN-Taste erzeugen Sie ein Zeilenende. Dadurch wird der Cursor an den Anfang der nächsten Zeile gesetzt. Wenn Sie inmitten einer Zeile RETURN tippen, können Sie sie in zwei Zeilen aufspalten. so daß der Rest der Zeile In die folgende geschoben wird. Zu beachten ist, daß in der 38. Spalte nicht mehr RETURN gedrückt werden kann!

Mit den Pfeiltasten CONTROL - "+ - = *" können Sie den Cursor im Text bewegen. Dabei wird das obere und untere Ende des Textes durch 2 Pfeile markiert. Über diese Markierungen hinaus können Sie den Cursor nicht bewegen, innerhalb einer Zeile können Sie den Cursor nur bis zum RETURN-Zeichen noch rechts bewegen.

Mit der BACK SPACE Taste bewegen Sie den Cursor um ein Zeichen nach links und löschen dabei das dort stehende Zeichen. Dabei können Sie nicht über das linke Zeilenende hinaus. Die CONTROL DELETE Taste löscht das Zeichen rechts vom Cursor, wobei der Zeilenrest nach-geschoben wird. Das (unsichtbare) RETURN Zeichen am Ende der Zeile kann so jedoch nicht gelöscht werden.
TAB fügt 2 Leerzeichen ein. Damit können Sie ihren Programmen einen übersichtlicheren Aufbau verleihen.

Der Editor – Editierfunktionen

Die CONTROL-Sequenzen

Durch Drücken von CONTROL zusammen mit einer der folgenden Tasten können Sie zusätzliche Funktionen aufrufen:

-X: Löschen der Zeile, in der sich der Cursor befindet.

-V: Vereinigen der Zeile mit der folgenden. Diese Funktion ist nur ausführbar, wenn die Länge der beiden Zeilen zusammen nicht größer als 38 ist.

-H: Bewegt den Cursor an den Textanfang.

-N: Bewegt den Cursor ans Textende.

-U: Bewegt den Cursor eine Seite hoch.

-D: Bewegt den Cursor eine Seite nach unten.

-;: Bewegt den Cursor ans Zeilenende.

-Clr: Löscht den Text nach Rückfrage.

-B: Setzt den Blockanfang (zeilenweise) und löscht das Blockende.

-E: Setzt das Blockende. Die Länge des Blocks darf dabei nicht größer als 3 KBytes sein. Der Block wird dann in einen internen Buffer kopiert, so daß er unverändert bleibt, auch wenn der Block im Text weiter editiert wird. ln der Statuszeile erscheint nun "Edit - Block defined".

-C: Kopiert den Block an der Position des Cursors In den Text.

-F: Sucht eine Folge von Zeichen wobei das "?" als Joker dient. Bei der Eingabe müssen Sie bedenken, daß der Text nur bis zum ersten Leerzeichen beachtet wird.

-R: Wiederholt die Suche.

-I: Zeigt das Directory von Laufwerk 1 an. Am Ende müssen Sie eine Taste drücken.

-S: Speichert den gesamten Text ab. Dabei können Sie entweder den Filenamen "D:XXXX.QIK" eintippen, oder falls Sie das schon einmal getan haben, mit RETURN den gleichen Namen wieder verwenden. Mit dieser Funktion können Sie den Text auch auf einen Drucker ausgeben, indem Sie als Namen "P:" angeben.

-L: Lädt einen Text. Dabei wird ein lm Speicher stehender Text gelöscht. Als Filenamen wird der bei SAVE eingetippte, oder der zuletzt im Compiler verwendete benutzt.

-M: Lädt einen Text und hängt ihn an dos Ende des aktuellen Textes an. Bedenken Sie, daß Definitionen globaler Variablen dann aber an den Anfang des Textes kopiert werden müssen (näheres dazu im nächsten Kapitel).

-O: Zeigt einen Hilfstext an. der die Steuertastenbelegung angibt. Durch Drücken einer Taste gelangen Sie wieder in den Edit-Modus.

-Q: Verläßt den Editor nach der Rückfrage ARE YOU SURE? (Sind Sie sicher), auf die Sie mit "Y" (ja) oder einer anderen Taste (nein) antworten müssen. Im Fall "Y" wird zur Shell zurückgekehrt.

Wie Sie sehen, bietet der Editor, trotz seiner Kürze, eine Reihe von praktischen Funktionen, die das Schreiben von Quicktexten komfortabler als das Schreiben von Basic-Programmen machen.

Der Compiler

Bevor wir die Bedienung des Compilers erläutern, wollen wir Ihnen zunächst erklären was ein Compiler eigentlich tut. Falls Sie bisher nicht in Assembler programmiert haben, werden Ihnen vielleicht manche Erläuterungen unverständlich erscheinen, denn oft wird darauf eingegangen, wie der Compiler in Maschinensprache übersetzt. Das ist aber kein Problem, denn um in Quick zu programmieren benötigen Sie natürlich keine Assemblerkenntnisse.

Dem Compiler auf die Finger geschaut

Der Quick-Compiler hat die Aufgabe, Quick Quelltexte, die Sie mit dem Editor erstellt haben, in lauffähige Maschinenprogramme zu übersetzen. Dabei sollen die erzeugten Programme nicht zu lang werden (wie es bei compilierten Programmen oft der Fall ist) und vor allem sollen sie schnell sein. Es ist klar, daß ein Compiler diese Forderungen nicht 100%ig erfüllen kann. Es muß also ein brauchbarer Kompromiss gefunden werden.

Was tut der Compiler nun im Einzelnen?

Der Compiler Ist ein 3 Pass Compiler, d.h. ein Text wird in drei Durchgängen in ein Programm übersetzt.

Pass 1
In diesem Durchgang werden alle verwendeten Variablen in eine Variablentabelle eingetragen. Das Gleiche gilt für Unterprogramme. Dabei wird auch die korrekte Gliederung des Programms überprüft.

Pass 2
Hier geschieht die eigentliche Übersetzung in Maschinensprache. Dabei unterscheidet der Compiler 3 verschiedene Übersetzungsarten:

1. Wertzuweisung: Darunter versteht man jeden Transfer von einer Variable in eine andere.
Bsp.: `A=B` oder `A=10`
Dieser Transfer von Werten wird optimal in Maschinensprache übersetzt:

```
LDA B    LDA #10
STA A    STA A
```

Natürlich entstehen bei Verwendung von 16-Bit-Variablen kompliziertere Programmteile.

2. Macros: Einfache Befehle werden durch fertige Programmteile in Maschinensprache übersetzt. Das entspricht der Vorgehensweise eines echten Macroassemblers.
Bsp: `SETCOL(N,F,H)`
Wird zu:

```
LDX N
LDA F
ASL
ASL
ASL
ASL
ORA H
STA 708,X
```

Auch das geht in Assembler in dieser Form selten schneller.

3. Runtime-Unterprogramme: Komplizierte und aufwendige Befehle rufen ein entsprechendes Unterprogramm im Runtimeteil auf. Der Runtimeteil ist ein ca. 3 kByte umfassender Programmblock, der die quickinternen Unterprogramme enthält. Er wird während des Compilierens nachgeladen und ins Programm eingebaut. Somit hat jedes Quick-Programm eine Mindestlänge von 3 kByte.
Diese Befehle sind am "langsamsten". Zuerst müssen Variablen übergeben werden (auf verschiedene Arten). dann wird dos Unterprogramm aufgerufen und schließlich müssen eventuelle Rückgabevariablen übertragen werden. Andererseits sind die Routinen dann aber wieder l00% Maschinensprache.

Bsp.:	`DIGI(G,A,E)`	spielt digitalisierte Sounds
	`MOUSE`	fragt eine ST-Maus ab

Pass 3:
Hier setzt der Compiler dann Sprungadressen ins Programm ein.

Die Bedienung des Compilers

Zu Beginn müssen Sie den Filenamen des zu compilierenden Programms (Quelltextes) eingeben. Falls Sie vorher den Editor verlassen haben, wird der dort verwendete Filenamen gleich angezeigt. Dann wird compiliert. Dabei können Sie zur Information die gerade compilierte Zeile sehen. Das geht aber so schnell, daß man nur ab und zu eine Zelle aufblitzen sieht.
Nach erfolgreichem Compilieren haben Sie folgende Auswahl:

Another File: Noch ein File compilieren.
Exit: Springt in die Shell.
Save: Speichert das compilierte Programm unter dem gleichen Namen mit Extender ".OBJ" ab.
Run: Startet ein compiliertes Programm. Rückkehr in die Shell ist durch Drücken von RESET möglich, falls man nur Speicherbereiche hinter dem Compiler benutzt.

Die Programmadresse steht nach erfolgreichem Compilieren unter dem kleinen Auswahlmenu.

Falls beim Compilieren ein Fehler auftritt, erscheint eine Fehlermeldung mit Fehlernummer und der Anzeige der fehlerhaften Zeile. Außerdem erscheint hinter der Fehlernummer der Name des Unterprogramms. in dem der Fehler auftrat.

Bsp:

```
Error "05 in FARBEN (C)ont (E)xit
```

Diese Meldung besagt, daß im Unterprogramm “FARBEN” ein unbekannter Befehl verwendet wurde. Ist der Fehler in keinem Unterprogramm so wird "MAIN” angezeigt. Tritt der Fehler jedoch schon beim 1.Pass auf, so kann keine genaue Position, sondern nur "????" angegeben werden.
Nun können Sie mit **E**xit den Compiler verlassen oder mit **C**ont den Compiliervorgang fortsetzen. Das compilierte Programm ist dann natürlich nicht fehlerfrei, außerdem können Folgefehler auftreten.

Nun aber zum wichtigsten Teil: Wie müssen Quick-Quelltexte aussehen?

Programmieren in Quick

Der Aufbau eines Quick-Quelltextes

Jeder Quelltext muß den folgenden Aufbau besitzen:
A) - Include-Filenamen (eventuell)
B) - Variablen-Deklarationsteil
C) - Hauptprogramm (Main)
D) - Unterprogramme (Proc)

A)
Mit dem Include-Befehl können Libraries (d.h. Unterprogramme) beim Compilieren nachgeladen werden:

```
INCLUDE
[
D1:GRAPH.LIB
D2:MATH.LIB
…
]
```

Dieser Aufbau (Keyword,[,Filenamen,]) ist typisch und muß immer exakt eingehalten werden. Es muß auch immer beachtet werden, daß nur ein Befehl pro Zeile verwendet werden darf.

B)
Im Variablendeklarationsteil des Hauptprogramms müssen alle globalen Variablen deklariert werden. Globale Variable sind Variablen, die im gesamten Programm "bekannt" sind, und somit überall verwendet werden können. Mon muß deshalb dem Compiler ganz zu Anfang diese Variablen bekanntgeben.
Im Gegensatz zum BASIC gibt es zunächst keine Fließkommavariablen. Stattdessen gibt es 3 verschiedene andere Arten:
1-Byte Variablen: "BYTE"-Variablen, die nur 1 Byte im Speicher belegen.
2-Byte Variablen: "WORD"-Variablen, die 2 Byte lm Speicher belegen.
Felder: "ARRAY"-Variablen, die 1 bis 255 Byte (je nach Deklaration) belegen. Diese ARRAYs können sowohl als "Strings" als auch als "eindimensionale Felder“ benutzt werden.

Der Aufbau des Deklarationsteils muß so aussehen:

```
Typ
[
Variablenname,Variablenname, …
…
]
```

Die Typen sind BYTE, WORD, ARRAY:
Bsp:

```
[BYTE
[
A1,F3,DA-S1
INT
]
WORD
[
```

```
W1,W0
]
ARRAY
[
FELD(10),TEXT(40)
]
```
Bei Feldvariablen muß hinter dem Namen die Länge des Feldes (1 bis 255) angegeben werden.

Der Compiler weist jeder Variable einen Speicherplatz in einem dafür vorgesehenen Speicherbereich zu. Es gibt aber auch die Möglichkeit direkt anzugeben, an welche Stelle die Variable gelegt werden soll. Das kann große Vorteile bieten:

```
BYTE
[
COL1=708, COL2=709
]
```
Nun kann man später im Programm anstatt
```
SETCOL(0,1,10)
```
schreiben
```
COL1=26
```
Das geht natürlich viel schneller und braucht weniger Platz. Auf diese Art werden POKE und PEEK (die auch vorhanden sind) in vielen Fällen überflüssig.

C)
Das Hauptprogramm beginnt mit
```
MAIN
```
und endet mit
```
ENDMAIN
```
Dazwischen steht das Programm. das aus allen möglichen Quick- Befehlen bestehen kann.

D)
Unterprogramms beginnen mit
```
PROC Unterprogrammname
```
und enden normalerweise mit
```
ENDPROC
```

Der erste Teil des Unterprogramms ist die Deklaration der "lokalen" Variablen, die in 3 Gruppen gegliedert sind: IN, OUT, LOCAL.

Bei IN müssen die Variablen deklariert werden, die beim Aufruf des Unterprogramms vom rufenden Programm ans Unterprogramm übergeben werden.
Bei OUT müssen dann die Variablen deklariert werden, die ans rufende Programm zurückgegeben werden sollen.
Wichtig! Reihenfolge und Anzahl der Variablen exakt einhalten.

Bei LOCAL werden zusätzliche Variablen deklariert, die nur interne Verwendung im Unterprogramm finden sollen. Es ist so also möglich im Unterprogramm Variablen mit dem gleichen Namen wie im Hauptprogramm zu deklarieren, die dann nur in diesem Unterprogramm bekannt sind. Sie können also in jedem Unterprogramm Zählschleifen mit "i" als Zählvariable verwenden, die sich gegenseitig nicht stören. Besonders wichtig sind lokale Variablen für die

Libraries. Rekursive Aufrufe sind aber nicht möglich. Globale Variablen dürfen natürlich im Unterprogramm auch verwendet werden.
Der Befehl BEGIN beendet den Deklarationsteil und leitet den Befehlsteil des Unterprogrammes ein.

Beispiel:

```
PROC BEISPIEL
IN
BYTE
[
VAR1
]
OUT
BYTE
[
VAR2
]
WORD
[
WERTI
]
LOCAL
ARRAY
[
TEXT
]
BEGIN
…
ENDPROC
```

Beim Aufruf des Unterprogramms muß also ein Wert übergeben und 2 Variablen entgegengenommen werden. Der Aufruf erfolgt so:

```
.BEISPIEL (10,V1,W2)
oder      .BEISPIEL (V0,V1,W2)
```

Beachten Sie den "**.**"! Er macht den eigentlichen Aufruf aus. Falls Sie eine Variable 2 mal zurückbekommen wollen, so erhält sie den Wert der zweiten OUT-Variable: Beim Aufruf .BEISPIEL(10,W2,W2) hat W2 also den Wert von WERT1.

Programmieren in Quick – Wertzuweisungen

In Quick gibt es nicht die Möglichkeit Terme (also numerische Ausdrücke) einer Variablen zuzuweisen:

```
A= 5-4*3    falsch
```

Bei Wertzuweisungen in Quick darf rechts vom Gleichheitszeichen nur eine Zahl, Variable oder ein Text stehen:

```
A=5          richtig!
W=-1000
A=B
```

Man kann also auch negative Zahlen verarbeiten, die in 2er- Komplement Notation abgespeichert werden. Zulässige Wertebereiche sind also bei:

	mit Vorzeichen (signed)	ohne Vorz. (unsigned)
BYTE	-128 bis +127	0 bis 255
WORD	-32768 bis +32767	0 bis 65535

Ob eine Variable (oder Zahl) ohne Vorzeichen oder im 2er- Komplement interpretiert wird, hängt nicht von der Variablen ab. Das wird durch den Modus, in dem gearbeitet wird festgelegt. Der Befehl UNSIGN schaltet die Vorzeichen aus; SIGN schaltet sie ein. Ab SIGN wird bei allen Variablen das Vorzeichen beachtet. Das wirkt sich dann bei PRINT und bei Vergleichen aus.

Bei der Verarbeitung von ARRAYS sind einige Dinge zu beachten:

- *ARRAY* als String: Man kann ARRAYS direkt Texte zuweisen. Falls der Text länger als die Dimensionierung ist, wird er abgeschnitten

```
TEXT="Hallo"
```

Der Inhalt eines Arrays kann auch direkt in ein anderes kopiert werden:

```
FELD=TEXT
```

Mon kann auch indiziert zuweisen:

```
FELD="ABCDE"
FELD(3)=TEXT
```

Ergibt: "ABCHallo". Dabei wird nicht beachtet, ob die Dimensionierung überschritten wird! Bei indizierter Zuweisung dürfen aber keine Texte zugewiesen werden:

```
FELD(3)="ABDC" falsch!
```

Das Ende eines Strings wird durch eine Null gekennzeichnet, deshalb müssen Sie immer um 1 länger dimensioniert werden, als eigentlich nötig.

- *ARRAY* als CHR$-Ersatz:

```
Im BASIC:   PRINT CHR$(125) (Löscht den Bildschirm)
in Quick:   FELD(0)=125
            FELD(1)=0          (kennzeichnet das Ende des Strings)
            PRINT (FELD)
```

- *ARRAY* als Zahlenfeld: Sie können es als normales eindimensionales Zahlenfeld verwenden, wobei Sie sich selbst entscheiden müssen, ob Sie 1 oder 2-Byte Zahlen verwenden:

```
            FELD(0)=1000       Damit ist FELD(0) und FELD(1) belegt
            FELD(0)=100        8-Bit, nur FELD(0) belegt
Aber        FELD(0)=1100       16-Bit,
```

so daß FELD(0) und FELD(1) belegt wird. Mit dem Ausrufezeichen wird also 16 Bit erzwungen.

Wichtig: Bel negativen Zahlen gilt folgende Notation:
FELD(0)=-1100 d.h. erst das Vorzeichen

Programmieren in Quick – Die Befehle

Die Befehle

Zum festen Sprachschatz von Quick gehören ca. 60 Befehle. Einige davon dürften Ihnen vom BASIC bekannt sein. Andere erinnern eher an Assembler oder C.
Der Compiler bindet die Befehle entweder direkt in den fertigen Programmcode ein, oder er ruft ein entsprechendes Unterprogramm aus dem RUNTIME-Teil auf (bei besonders aufwendigen Befehlen). Solche Befehle sind mit einem "x" gekennzeichnet.

** Kommentar*
Der "*" kennzeichnet einen Kommentar. Er kann alleine in der Zelle oder auch rechts von einem Befehl stehen. Der Quelltext darf übrigens auch Leerzeilen enthalten.

OPEN (NR,AUX1,AUX2,NAME)
Öffnet Kanal Nummer NR mit den Parametern AUX1, AUX2 mit dem Dateinamen NAME.
Bsp: `OPEN (1,4,0,"D:TEXT.TXT")`
Wie bei den meisten anderen Befehlen dürfen die Parameter nur BYTE oder WORD-Variablen oder Zahlen sein. ARRAYs können nur als Strings beim Aufruf verwendet werden, nicht jedoch als "Integervariabie" in der Form FELD(1). D.h. eine Indizierung beim Befehlsaufruf ist nicht möglich:

```
OPEN (10,FELD(1),0,"P:")     falsch!

A=FELD(1)
OPEN (10,A,0,"P:")           richtig!
```

Zu Beginn eines Quick-Programms müssen Sie normalerweise den Bildschirm- oder den Editorkanal (in Grafik 0) öffnen:

```
CLOSE(6)
OPEN (6,12,0,"S:")     Print möglich; Input nicht möglich
```
oder
```
OPEN (6,12,0,"E:")     Print und Input möglich
```

Dieser Befehl sollte bei jedem Programm verwendet werden, es sei denn, es benutzt die GRAPHICS-Routine aus der Grafik-Library.

CLOSE (NR)
Schließt den Kanal NR.

BGET (NR,ANZ,ADR)
Liest ANZ Bytes von Kanal NR ab der Speicherstelle ADR. Dient auch oft zur Eingabe von Zeichen von der Tastatur bei nicht geöffnetem Editor und zur Umgehung von INPUT.

BPUT (NR,ANZ,ADR)
Schreibt ANZ Werte ab ADR auf Kanal NR.

INPUT (A) x
Dient zur Eingabe einer Zahl (A ist BYTE oder WORD) oder eines Textes (A ist ARRAY).
Wichtig: Der INPUT-Befehl funktioniert nur wenn man vorher einen Editorkanal geöffnet hat!

PRINT (A1,A2,A3,...) oder ? (A1,A2,A3,...) x
Schreibt auf den Bildschirm. in der Klammer können beliebig viele Parameter, durch Kommata getrennt. angegeben werden. Hier ist die Verwendung von BYTE. WORD. gesamten ARRAY: und Texten möglich:

```
PRINT (A,B,10,"Hallo, Welt",FELD)
```

Nach Zahlen wird jeweils ein Leerzeichen eingefügt. Am Ende des Befehls kann ein ";" angefügt werden. Dann wird nach dem PRINT kein RETURN ausgeführt, und der nächste PRINT beginnt direkt hinter dem vorherigen:

```
PRINT ("Hallo,");
PRINT (" Welt")
```

ergibt

```
Hallo, Welt
```

LPT (A1,A2,...) x
Schreibt Im selben Format wie PRINT auf Kanal 5 (Kanal vorher öffnen). Dieser Befehl kann beispielsweise benutzt werden. um Daten zum Drucker zu senden. Sie können aber auch auf Diskette "printen".

```
OPEN(5,8,O,"P:")
LPT("Das druckt der Printer")
```

SIGN
Bel PRINT werden Variablen mit Vorzeichen ausgegeben. Bel Vergleichen wird das Vorzeichen beachtet.

UNSIGN
Keine Vorzeichen. d.h. Variablen werden nicht im 2er-Komplement interpretiert.

POS (X,Y)
Setzt den Cursor an die Stelle X, Y.

LOCATE (WERT) x
Liest den Inhalt des Bildschirms an der Stelle des Cursors In WERT.

COLOR (A)
Wöhlt Farbregister 0 bis 4 (bzw 15) für PLOT und DRAW.

PLOT (X,Y) x
Setzt einen Punkt in der gewählten Farbe an die Stelle X,Y.

DRAW (X,Y) x
Zieht eine Linie an die Stelle X,Y

PLAYER(Z,I,L,Q)
Überträgt *L* Daten ab der Adresse *Q* in die Page *Z*, wobei *I* als Index zur Zieladresse addiert wird. Dieser Befehl eignet sich sehr gut. um Playerdaten in den Playerbereich zu kopieren, wobei *I* sozusagen als Y-Position verwendet werden kann. *I* darf nur 0 bis 255 betragen.

CLR (P,A) x
Löscht ab der Page *P A**256 Bytes. Eine Page ist ein 256 Byte langer Speicherblock, z.B. ist Page 0 der Bereich von 0 bis 255. Page 1 von 256 bis 511.

CUT (X1,Y1,X2,Y2,ADR) x
Schneidet ein Rechteck mit den Eckpositionen *X1,Y1* und *X2,Y2* aus und legt es in den Speicher ab der Adresse ADR. Funktioniert In Graphics 8, oder bei geradem *X1* und *X2* auch in Graphics 15,7 (dabei X1 und X2 einfach verdoppeln!). Quick ist also die erste Programmiersprache für die XL/XE-Computer mit "eingebautem" Blitter...

PASTE (M,X1,Y1,ADR) x
Kopiert die Paste-Daten in den Bildschirm. Die linke obere Ecke wird durch *X1,Y1* angegeben. Bei *M*=0 wird einfach überschrieben, bei *M*=1 wird im OR-Modus eingesetzt (Nur in Graphics 8 sinnvoll).

SETCOL (N,F,H)
Setzt Farbregister *N* auf Farbe *F* in Helligkeit *H*.

SOUND (K,H,V,L)
Schaltet Tonkanal *K* mit Tonhöhe *H* mit Verzerrung *V* und Lautstärke *L* ein. Bei der Verzerrung müssen Werte von 0 bis 7 angegeben werden (die Hälfte der Werte in BASIC).

DIGI (G,A,E) x
Spielt digitalisierte Sounds. Die Daten werden von Adresse *A* bis *E* vorgespielt. Bei *A* und *E* wird nur das Highbyte beachtet. Als Geschwindigkeit kann 1 bis 255 (schnell->langsam) angegeben werden. Dabei wird der Bildschirm abgeschaltet. Bei *G*=0 wird der Bildschirm nicht abgeschaltet, dafür ist nur ein konstantes Tempo möglich.
Die Sounddaten müssen im **ATARI***magazin* Soundsampler-Format vorliegen. Dabei ist ein Byte in jeweils 4 Bits aufgeteilt, die jeweils einen Soundwert (0-15) darstellen. (Näheres siehe **ATARI***magazin* 1/89 und 5/89)

MOUSE x
Liefert die Position einer in Port 2 angeschlossenen ST-Maus in den Speicherzellen 178 (MOUSEX) und 179 (MOUSEY). Dieser Befehl sollte In einer Schleife eingesetzt werden, die kaum durch andere Befehle "gebremst" wird, um auch noch schnelle Mausbewegungen richtig zu erfassen.
Durch Einschreiben von Werten in 178 und 179 kann die Maus auch gesetzt werden. Der Mauszeiger muß getrennt mit Hilfe des PLAYER- Befehls (z.B. im VBI (siehe später)) dargestellt werden.

DATA (ADR)
[
1,4,876,4563,34,...
]
Schreibt die Daten ab der Position *ADR* in den Speicher.

POKE (A,B)
Schreibt Wert von *A* in Speicherzelle *B*.

PEEK (A,B)
Schreibt Inhalt der Speicherzelle *A* (8 Bit Inhalt) in Variable *B* bzw. in Adresse *B*.

DPEEK, DPOKE
Das Gleiche mit 16-Bit.

BMOVE (Q,Z,L) x
Kopiert einen Speicherbereich ab Adresse *Q* der Länge *L* an die Adresse *Z* bis *Z+L-1*. Überlappende Blöcke werden problemlos kopiert.

FMOVE (Q,2,L) x
Wie oben, aber *L* muß kleiner 256 sein. Überlappende Blöcke werden eventuell nicht richtig kopiert. Der Befehl ist schneller als MOVE.

CALL (A,X,Y,ADR) X
Ruft ein Maschinenprogramm ob der Adresse *ADR* auf. Vorher werden *A,X,Y* in den Akku, das X- und Y- Register übertragen. Das Unterprogramm muß mit RTS enden.

INLINE
[
169,45,141,A1...
]
Schreibt die Daten innerhalb der Klammern direkt ins Programm. Als Daten können auch Variablen verwendet werden. Dabei wird deren Adresse eingesetzt. wobei normalerweise eine 2-Byte-Adresse eingesetzt wird. Es sei denn , die Variable liegt in der Page 0. Dieser Befehl dient zum einfachen Einbinden von kurzen Maschinensprache-Teilen (besonders auch für DLIs (siehe später)).

REGX (Z), REGY (Z), REGA (Z), REGP (Z)
Überträgt X, bzw Y, bzw. Akku, bzw. Statusregister in Variable Z (bei WORDs nur ins Lowbyte!)
Vorsicht: REGP verändert Akku.

```
REGP(VAR1)
REGA(VAR2)  falsch!

REGA(VAR2)
REGP(VAR1)  richtig!
```

PROCADR (Unterprogrammname)
Nach dem Aufruf enthalten die Speicherzellen $D0 und $D1 (208,209) die Adresse des entsprechenden Unterprogramms.

ADD (A,B,C)
Entspricht *C=A+B*. Überläufe werden nirgends überprüft.

A+
Erhöht 8 Bit Variable *A* um 1 (der sog. INC-Befehl).

SUB (A,B,C)
Entspricht *C=A-B*

A-
Erniedrigt 8 Bit Variable *A* um 1 (der sog. DEC-Befehl).

MULT (A.B,C) x
Entspricht *C=A*B*

DIV (A,B,C) x
Entspricht *C=A/B*

AND (A,B,C)
Entspricht *C=A* AND *B* (Bitweise). Für nicht Assemblerfreaks hier eine Erklärung: Die einzelnen Bits der Variablen/Werte *A* und *B* werden nacheinander verknüpft. Sind die beiden Bits beider Werte gleichzeitig gesetzt. so wird auch das Bit in *C* gesetzt.

Bsp:	A=6	(binär: 00000110)
	B=3	(binär: 00000011)
	AND(A,B,C)	ergibt 2 (binär: 00000010) in *C*

OR(A,B,C), EOR(A.B,C)
Ergibt die entsprechenden Verknüpfungen mit OR (ergibt jeweils 1 wenn wenigstens eines der beiden Bits gesetzt ist) und EXOR (ergibt 1 wenn die beiden Bits nicht gleich sind).

ASLB(A)
Schiebt Inhalt der Variable A (8 Bit) um ein Bit nach links. Das bedeutet. der Wert der Variable wird verdoppelt.

Bsp:	A=5	(binär: 00000101)
	ASLB(A)	ergibt 10 (binär: 00001010) in *A*.

Auf diese Art kann man also eine Zahl schnell mit 2 multiplizieren.

ASLW(A)
Schiebt Inhalt der Variable *A* (16 Bit) um ein Bit nach links.

ASRB(A)
Schiebt den Inhalt von *A* um ein Bit vorzeichenrichtig nach rechts. Auf diese Art kann man eine Variable schnell durch 2 dividieren. lm SIGN-Modus sollte dieser Befehl verwendet werden.

ASRW(A)
Schiebt Inhalt von *A* vorzeichenrichtig nach rechts. (*A* 16-Bit).

LSRW(A)
Schiebt *A* (16 Bit) um ein Bit nach rechts (ohne Vorzeichenbeachtung). Dabei wird links in jedem Fall eine 0 herein geschoben.

LSRB(A)
Schiebt *A* (8 Bit) um ein Bit nach rechts (ohne Vorzeichenbeachtung).

Im Text können Labels gesetzt werden die als Sprungziele dienen. Ein Label besteht aus einem „-„ und einer Zahl von 0 bis 384.

```
-10
```

Die Labels können dann mit JUMP angesprungen werden:

```
JUMP (10)
```

Eleganter und strukturierter geht es aber mit folgenden Befehlen:

```
REPEAT
...
UNTIL Vergleich
```

und

```
WHILE Vergleich
...
WEND
```

Ein Vergleich ist dabei ein Ausdruck der Form WERT OPERATOR WERT. Zum Beispiel:

```
A>C
D>=6
5<=100
G<>9
A=M
J<M
```

aber nicht:

```
FELD="ABCD"
A=B OR C>5
A=B-C
```

Bei WHILE-WEND handelt es sich um eine abweisende Schleife. Zu Beginn wird die Bedingung überprüft und falls sie falsch ist, gleich zum ersten Befehl hinter WEND gesprungen. Ansonsten wird der Block solange wiederholt, bis die Bedingung nicht mehr wahr ist.
REPEAT-UNTIL wird dagegen auf jeden Fall einmal durchlaufen, da die Bedingung erst am Ende überprüft wird.
Beachten Sie, daß Sie hier selbst für eine Zählvariable und deren Veränderung sorgen müssen (im Gegensatz zum FOR-NEXT des Basic).

Bsp:

```
I=0
WHILE I<10
  ?("I ist kleiner als 10)
  I+
WEND
```

Eine weitere wichtige Kontrollstruktur ist:

```
IF Vergleich
( ELSE
… )
ENDIF
```

Bsp:

```
IF A<0
  ?("A ist kleiner als Null")
ELSE
  ?(„A ist nicht kleiner als Null")
ENDIF
```

Alle Kontrollstrukturen können beliebig geschachtelt werden.

Programmieren in Quick - interrupts

Interrupt-Programmierung

In Quick kann man ein normales Unterprogramm als DLI oder VBI verwenden. Das Unterprogramm muß dann einfach mit dem Befehl INTER (anstatt PROC) beginnen und mit ENDDLI oder ENDVBI enden. Eingeschaltet wird der **VBI** mit dem Befehl VBI (Unterprogrammname), der **DLI** mit DLI (Unterprogrammname). Beim DLI muß man selber alle (CPU-)Register retten und am Ende wieder herstellen. Dafür gibt es die Befehle PUSH und PULL.

```
MAIN              Hier ist das Hauptprogramm, wo im Hintergrund
…                 die Interrupts laufen sollen.
VBI (INTERRUPT)   Aufrufen/Einschalten der
DLI (FARBEN)      Interrupts.
…
ENDMAIN

INTER INTERRUPT   Das VBI-Unterprogramm
…                 hier könnte man zunächst lokale Variablen deklarieren
BEGIN
…                 hier steht dann das eigentliche Programm
ENDVBI

INTER FARBEN      Das DLI-Unterprogramm
…
BEGIN
PUSH
…                 hier steht das Programm
PULL
ENDDLI
```

Auf diese Art und Weise hat man aber nur prozessorinterne Register gerettet. Falls Sie mit "x" gekennzeichnete Routinen im Interrupt verwenden, müssen Sie aber auch compilerinterne Variablen retten. Das geschieht mit

```
IPUSH
ZPUSH
…
IPULL
ZPULL
```

IPUSH ist bei den meisten Vergleichen und bei x-Routinen notwendig. ZPUSH wird bei PEEK oder POKE oder X-Routinen benötigt.
Eine Ausnahme bilden PLAYER und CLR. Hier ist kein IPUSH und ZPUSH nötig, weil im Interrupt andere compilerinterne Variablen verwendet werden.

Bei Interrupt-Unterprogrammen können nur **LOCAL** (und globale) Variablen verwendet werden, jedoch keine IN und OUT Variablen.
Es muß natürlich auch beachtet werden, daß ein DLI nur wenige Millisekunden (also nur einige Bildschirmzeilen) lang dauern darf. Ein VBI darf im äußersten Fall 1/50 Sekunden lang sein.

Fehlermeldungen des Compilers

Der Compiler erzeugt während des Übersetzens eine Reihe von Fehlermeldungen. Die Fehlermeldungen sind nicht immer ganz eindeutig und eventuell auch nicht immer in der richtigen Zeile.
Das laufende Maschinenprogramm selbst erzeugt keine Fehlermeldungen, höchstens Abstürze.

Nummer (hex)	Bedeutung
01	"[" fehlt
02	Name bei PROC fehlt
03	MAIN zweimal verwendet In einem Quickpragramm darf es nur ein Hauptprogramm geben
04	kein MAIN oder PROC vor MAIN Reihenfolge Hauptprogramm - Unterprogramme muß eingehalten werden
05	unbekannter Befehl
06	ungültiger Wert
07	unbekannte Variable Variable wurde nicht deklariert
08	ungültige Wertzuweisung Falscher Variablentyp, z.B. wurde einer BYTE- Variable ein Text zugewiesen
09	Zahl zu groß
0A	Längenangabe bei ARRAY-Deklaration fehlt Jeder Array kann 1 bis 255 Einträge lang sein
0B	Längenangabe bei ARRAY-Deklaration zu groß (>255)
0C	ungültiger Wert als Index Falscher Variablentyp als Index oder falscher Wert
0D	unbekanntes Unterprogramm Es wird ein Unterprogramm aufgerufen, das nicht deklariert wurde. Oder vielleicht fehlt der INCLUDE-Befehl
0E	zu wenige Parameter bei Unterprogrammaufruf Es müssen immer so viele IN- und OUT- Variablen wie im Unterprogramm deklariert übergeben werden.
0F	interner Fehler wie haben Sie das geschafft???
11	ARRAY mit Index hier nicht erlaubt Nur BYTE oder WORD- Variable verwenden
12	Kein Index erlaubt Array ohne Index verwenden
13	Fehler bei INCLUDE File kann nicht geladen werden
14	Parameter fehlt falscher Befehlsaufruf
15	")" fehlt
16	nur BYTE erlaubt
17	"(" fehlt
18	nur BYTE oder WORD erlaubt
19	Operator fehlt Vergleich hat nicht die richtige Syntax

1A	interner Stack übergelaufen Weniger tief verschachteln (Schleifen, if)
1B	falsche Verschachtelung
1C	IF, REPEAT oder WHILE fehlt END/F, UNTIL oder WEND gefunden, ohne zugehörigen Anfang
1D	Text kann nicht geladen werden Falscher Filename beim Compiler eingegeben
1E	falsche Adresse bei DATA Keine Variable als Adresse verwenden
1F	Labelnummer ungültig Wert zu groß oder negativ
20	compiliertes Programm zu lang
21	zu viele Variablen
22	zu viele Unterprogramme
23	zu viele Unterprogrammaufrufe oder Labelsprünge
24	zu viele Includefiles
25	BEGIN fehlt BEGIN muß nach der Variablendeklaration im Unterpragramm stehen
26	falsche Variablenart Bel Befehlsaufruf falsche Variablenart als Parameter benutzt
27	ENDIF, UNTIL oder WEND fehlt Dieser Fehler tritt erst am Ende eines Programmteils (ENDMAIN, ENDPROC,...) auf. Überprüfen Sie den Pragrammteil auf die richtige Verschachtelung

Tips & Tricks

Nun können Sie also mit dem Programmieren richtig loslegen. Damit das ein bißchen einfacher geht, hier einige Zusatzinformationen.

Geschwindigkeitsoptimierung
Wie auch in jeder anderen Programmiersprache kann man auch in Quick für jedes Problem verschiedene Programmlösungen finden. Oft geht es schneller, nicht den komplexen Quickbefehl zu benutzen:
Wollen Sie z.B. in einem Interrupt die Helligkeit eines Farbregisters ändern, so ist es sicher ungeschickt den SETCOL- Befehl zu verwenden. Dabei muß nämlich die Farbe mit 16 multipliziert und zur Helligkeit addiert werden. Es geht dagegen viel schneller eine Quickvariable auf die Adresse des Farbregisters zu legen und dann direkt das Register anzusprechen.

```
Langsam:          Schnell:
BYTE              BYTE
[                 [
H                 H= 708
]                 ]
…                 …
REPEAT            REPEAT
  H+                H+
  SETCOL(0,0,H)   UNTIL H=15
UNTIL H=15
```

Wie man sieht wird immer der "INC"-Befehl (+) verwendet. Das geht wesentlich schneller als der ADD-Befehl. Das ist aber nur bei BYTE-Variablen erlaubt.

Oft ist es nötig BYTE-Variablen zusammen auf einen Ausgangswert zu setzen. Das geht mit dem REGA-Befehl ganz besonders schnell:

langsam	schneller
A=0	A=0
B=0	REGA(B)
C=0	REGA(C)

Befehle wie AND, OR, ADD, SUB, COL, ASL, LSR sind recht schnell. Oft ist es günstig solche Befehle "unnötig oft" auszuführen, als deren Ausführung durch langsame Abfragen mit "IF-ENDIF" zu umgehen.

Zu den langsamen Befehlen, die in zeitkritischen Programmteilen nicht unbedingt verwendet werden sollten, gehören IF-ENDIF, PRINT, REPEAT,... Natürlich sind sie immer noch vergleichsweise schnell (bezogen auf alles andere als Assembler).

Speicheraufbau

Oft ist es wichtig zu wissen, wo im Speicher noch Platz für Player, Zeichensätze, Bildschirme usw. ist. Der folgende Speicherplan gilt für den Moment nach dem Compilieren eines Quelltextes.

1C70-1EFF Shell
1F00-41FF Compiler 4200-4FFF Runtime
5000-XXXX Programm
Frei
B000-BFFF Variablen C000-C7FF Editor Teil1
OS
E000-FFFF Editor Teil 2

Quick
Speicheraufbau

Grundsätzlich ist also Platz vom Programmende bis AFFF um Daten in den Speicher zu schreiben. Zu beachten ist, daß der Bildschirm unterhalb von AFFF aufgebaut wird, wenn Sie OPEN benutzen. Wenn Sie sich daran halten, gelangen Sie im Programm durch Drücken von RESET wieder in die Shell zurück.

Natürlich können Sie auch Bereiche des Compilers benutzen. Das bedeutet aber, daß Sie die Systemdisk nach jedem Programmstart neu booten müssen. Der Runtimebereich darf aber nicht überschrieben werden.

Woher weiß man aber, wo das compilierte Programm zu Ende ist? Diese Adresse des ersten freien Bytes wird nach dem Compilieren angezeigt. Es ist aber auch möglich diese Adresse im laufenden Quickprogramm zu erfahren: In den Speicherzellen 4FFE und 4FFF steht der entsprechende Wert.

```
WORD
[
FREI=20478
]
…
?("Erstes freies Byte ",FREI)
```

Das Eigenleben des Compilers

Der Compiler setzt bereits einige Systemvariablen auf bestimmte Werte: Linker Rand auf Null, Farben,... Wenn Sie Ihr Programm mit Run vom Compiler aus starten werden diese Einstellungen natürlich übernommen.
Speichern Sie das Programm dagegen ab, und starten es dann vom DOS aus, so gelten diese Voreinstellungen natürlich nicht! Ihr Programm sieht dann manchmal ganz anders aus.

Assemblernahe Programmierung: Nur für Geübte!

Mit dem lNLINE-Befehl gibt es in Quick die Möglichkeit direkt in den Compilierprozeß einzugreifen. Hier werden die Daten im lNLINE-Block direkt in das Programm eingefügt. Das setzt natürlich voraus, daß die Werte ein ausführbares Maschinenprogramm ergeben, oder daß dieser Befehl mit einem JUMP übersprungen wird. INLINE ist z.B. besonders nützlich bei zeitkritischen Programmteilen, denn man kann mit seiner Hilfe kleine Maschinenprogramme einfügen:

Bsp. Verzweigung

```
BYTE
[
WERT
]
…
IF WERT=5
  ?("Es war 5")
ENDIF
```

ohne IF

```
BYTE
[
WERT
LDA=173
BEQ=240
CMP=224
]
…
INLINE
[
LDA,WERT,CMP,5,BEQ,3
]
JUMP(1)      ;JUMP() belegt 3 Bytes
?("Es war 5")
-1
```

Wie schon erwähnt, wird im INLINE-Befehl die Adresse einer Variable eingesetzt. Hat die Variable eine Adresse <256, so wird nur ein Byte eingesetzt, sonst 2 Byte. Damit ist es möglich, sich Assemblerbefehle zu definieren. Wenn die Variable CMP auf die Adresse 224 gelegt wird, "so wird im INLINE-Befehl 224 an die Stelle der Variable gesetzt. Man sollte diesen Variablen aber keinen Wert zuweisen, da sie nur Platzhalter für Zahlen sind. In dem Speicherbereich liegen nämlich viele Systemvariablen.
Bedenken Sie daß sich Maschinenbefehle unterscheiden können, je nachdem auf welche Art von Adressen (Zeropage oder nicht) sie sich beziehen.

Interrupts

Für alle (Basic-)Programmierer, die sich bisher nicht mit Interrupts beschäftigt haben, hier einige grundsätzliche Informationen. Ein Interrupt ist ein kurzes Maschinen/Quick- Programm, das in regelmäßigen Abständen aufgerufen wird. in Quick gibt es dabei 2 verschiedene Interruptarten:

VBI: Dieser wird jede fünfzigstel Sekunde (nicht jedoch während I/O-Operationen) aufgerufen. Er darf etwa 24000 Maschinenzyklen lang sein, d.h. nicht ganz 1/50 Sekunde, denn dann wird er ja schon wieder aufgerufen. Der VBI eignet sich z.B. zum Bewegen von Playern, Spielen von Musik, Darstellen eines Mauszeigers usw.
DLI: Der Displaylist-Interrupt wird vom Grafikchip ausgelöst, wenn er in der Displaylist auf einen Befehl der größer als 127 ist trifft. Es genügt also nicht, den DLI in Quick einzuschalten, es muß auch ein entsprechender Befehl in der Displaylist angebracht werden. Der DLI sollte nur wenige Befehle lang sein (weniger als 4500 Maschinenzyklen). Es ist theoretisch auch möglich mehrere verschiedene DLIs zu verwenden.
Näheres siehe **ATARI***magazin* 7/88 und Beispielprogramm auf der Disk.

Die Libraries

Quick ist eine Sprache, die leicht zu erweitern ist. Dies geschieht mit Hilfe von Libraries. Eine Library ist eine Sammlung von Unterprogrammen, die vom Compiler nachgeladen werden kann. Eine Library besitzt einen einfachen Aufbau: Sie besteht aus einer Reihe von Routinen, die ganz herkömmlich mit PROC und ENDPROC gekennzeichnet sind.
Wollen Sie Routinen einer Bibliothek benutzen, so müssen Sie mittels des lNCLUDE-Belehls am Anfang Ihres Programmes die Library laden und die Routinen der Library wie normale Unterprogramme aufrufen. Beachten Sie, daß der Aufruf einer Libraryroutine eventuell den SIGN oder UNSIGN-Modus ändern könnte.

Zum Grundumfang von Quick gehören die folgenden 3 Libraries:

Die Grafik-Library: **GRAPH.LIB**
Diese Bibliothek bietet eine Reihe nützlicher Grafikfunktionen an:

.GRAPHICS(GR)
Schaltet eine Grafikbetriebsart wie im BASIC ein. Dabei kann ein Grafikmodus von 0 bis 15 gewählt werden. Wird 16 dazu addiert, wird kein Textfenster dargestellt. Wird 32 addiert, so wird der Bildschirmspeicher nicht gelöscht.
ln Textfenstern kann kein INPUT ausgeführt werden.

.FRAME(X1,Y1,X2,Y2)
Zeichnet ein Rechteck mit den Eckkoordinaten X1,Y1 und X2,Y2.

.BOX(X1,Y1,X2,Y2)
Zelchnet ein ausgefülltes Rechteck.

.CIRCLE(X,Y,R)
Zeichnet einen Kreis um den Mittelpunkt X,Y mit dem Radius R. Diese Routine funktioniert ab Grafikmodus 4 aufwärts. In Grafikstufen, in denen die X-Koordinaten in einem anderen Verhältnis dargestellt werden als die Y-Koordinaten (z.B. GRAPHICS 15) wird der Radius in X-Richtung entsprechend angepaßt.

.DISC(X,Y,R)
Zeichnet einen ausgefüllten Kreis.

.FILL(X,Y,X1,Y1,X2,Y2)
Füllt einen einfarbigen Bildschirmbereich ab der Position X,Y. Mit X1,Y1,X2,Y2 kann der maximal zu füllende Bildschirmausschnitt festgelegt werden.

Die Mathematik-Library: **MATH.LIB**
Damit werden die Fließkommaroutinen des Betriebssystems angesprochen, so daß mit Fließkommazahlen gerechnet werden kann. Eine Fließkommazahl wird als ARRAY der Länge 6 dargestellt. Das 0-te Byte gibt dabei den Exponenten und (im 7.Bit) das Vorzeichen an. Die folgenden 5 Bytes stellen 10 Ziffern in BCD-Format dar. Wenn Sie nur Routinen der Library benutzen, müssen Sie sich jedoch mit der Internen Darstellung kaum befassen.

.IFP(WORD,FLOAT)
Wandelt eine Integerzahl (unsigned Word) In eine Fließkommazahl (d.h. in ein 6 Byte langes Array).

.FPI(FLOAT,WORD)
Wandelt eine Fließkommazahl in eine Integerzahl um. Falls die Floatzahl negativ oder größer als 65536 war, ist der Wert der lntegerzahl unspezifiziert.

.AFP(ASC,FLOAT)
Wandelt einen ASCII-String (beliebiges Array) in eine Fließkommazahl um.

.DFP(D1,D2,D3,D4,D5,D6,FLOAT)
Setzt 6 Datenbytes zu einer Fließkommazahl zusammen.

.FASC(FLOAT,ASC)
Wandelt eine Fließkommazahl in einen String um.

.FADD(FIN1,FIN2,FOUT)
Addiert 2 Fließkammozahlen und liefert das Ergebnis zurück.

.FSUB(FIN1,FIN2,FOUT)
Subtrahiert 2 Zahlen.

.FMUL(FIN1,FIN2,FOUT)
Multipliziert 2 Zahlen.

.FDIV(FIN1,FIN2,FOUT)
Dividiert 2 Zahlen.

.FCMP(FLOAT,BYTE)
Überprüft, ob eine Fließkommazahl >0, =0 oder <0 ist und liefert ein Byte mit dem Wert 2 1 oder 0 zurück.

.FPRT(FLOAT)
Druckt eine Zahl auf dem Bildschirm aus.

Die Numerische Library: **NUMERIC.LIB**
Diese Bibliothek stellt eine Reihe mathematischer Funktionen zur Verfügung:

.EXP(FIN,FOUT)
Berechnet e^FloatIN und liefert das Ergebnis zurück.

.EXP10(FIN,FOUT)
Berechnet 10^FloatIN.

.LOG(FIN,FOUT)
Berechnet den natürlichen Logarithmus von FloatIN.

.LOG10(FIN,FOUT)
Berechnet den dekadischen Logarithmus.

.SQR(FIN,FOUT)
Berechnet die Wurzel von ABS(FloatIN).

.SIN(FIN,FOUT)
Berechnet den Sinus von FloatIN im Bogenmaß.

.COS(FIN,FOUT)
Berechnet den Cosinus von FloatIN.

.ATN(FIN,FOUT)
Berechnet den Arcustangens von FloatIN im Bogenmaß.

.ABS(FIN,FOUT)
Berechnet den Absolutwert von FloatIN.

Programmierung eigener Libraries:

Bei der Programmierung eigener Libraries müssen Sie nur wenig beachten.
Da eine Library "hinten" am eigentlichen Programm durch den Compiler angehängt wird, können Sie natürlich nur IN,OUT oder LOCAL- Variablen verwenden (keine globalen!). Das heißt, alle Variablen müssen beim Aufruf einer Libraryfunktion übergeben werden.
JUMPs sollten spärlichst verwendet werden, denn JUMP-Nummern sind immer global, so daß hier Konflikte mit gleichen Nummern in anderen Libraries oder im Hauptprogramm auftreten können.

Sie sollten die Standardlibraries nicht ändern, oder wenn doch, dann sollten Sie einen anderen Namen für die Library verwenden. Es ist nämlich sehr wahrscheinlich, daß auf diese Libraries in anderen Programmen immer wieder zurückgegriffen wird. Änderungen in diesen Libraries könnten sich dann sehr störend auswirken.

Bei der Benutzung der Libraries sollten Sie bedenken, daß immer die gesamte Library an das Programm angehängt wird. Dadurch wird das Programm natürlich länger. Eventuell ist es deshalb manchmal sinnvoll einzelne Teile einer Library direkt in das Programm zu übernehmen. Dabei müssen Sie aber beachten, daß einige Routinen andere Libraryroutinen benötigen.

Beispielprogramme:
Auf der Systemdiskette befinden sich einige kommentierte Quelltexte, die Ihnen die Programmierung von Quick im allgemeinen, und die Benutzung der Libraries im Besonderen beispielhaft erklären sollen.

QUICK hat inzwischen einen monatelangen "Praxistest" bei vielen Usern hinter sich. Die so gewonnenen Erfahrungen dienten nun zur Verbesserung von QUICK. Die Version 2.0 ist zwar voll aufwärtskompatibel, doch es gibt einige Änderungen, die wir Ihnen nun vorstellen möchten:

Das QUICK-Programmiersystem

Auf der Systemdisk befinden sich nun die folgenden Files:

COM	Compiler	Neue Version nur für v2.0
EDI	Editor	Neu, aber auch für v1.6
EXT	Compilerzusatz	Ganz neu
AUTORUNSYS	Shell	Neue Version nur für v2.0
RUNTIMEDBJ	Runtimeteil	Neu, aber auch mit v1.6 zu verwenden
GRAPH.LIB, MATH.LIB, NUMERIC.LIB		Neu, aber auch mit v1.6 zu verwenden

BYTE-Variablen

Bisher war es möglich BYTE-Variablen als unsigned oder signed zu verwenden. Da es in der Praxis aber meist unnötig ist signed Byte zu verwenden ist es bei v2.0 nicht mehr möglich BYTE als signed zu verwenden. SIGN hat nun also nur noch Wirkung auf WORD-Variablen.
Falls Sie alte v1.6 Programme compilieren möchten, in denen negative BYTEs vorkommen, müssen Sie den Befehl OLD als ersten Befehl in das Listing einfügen. Nun verhält sich v2.0 wie v1.6. Der Grund für die Änderung ist, daß QUICK nun 20% schnellere und kürzere Programme erzeugen kann, jedoch nur wenn auf signed BYTE verzichtet wird.

Neue Befehle

SYNC(X)
Fügt X Synchronisationsbefehle ins Listing ein. X muß eine Zahl, keine Variable sein. Der Befehl wird im DLI benötigt. Möchte man z.B. den Wert eines Farbregisters am Ende einer Zeile ändern, so war es bisher nötig explizit die Speicherzelle WSYNC anzusprechen, um ein Flackern zu verhindern (wie in Assembler). Der Befehl SYNC führt nun eine solche Zeilenendesynchronisation aus. Es ist manchmal auch praktisch mehrere Synchronisationen hintereinander auszuführen, um so in bestimmten Abständen Farben zu ändern, ohne mehrere DLIs einbauen zu müssen.

VADR(X)
Schreibt die Adresse der Variable X in die Speicherzellen 208, 209.

PADR(Name)
Schreibt die Adresse des Unterprogramms Name in 208,209. (Nicht wie irrtümlich angegeben PROCADR(Name)).

OLD
Schaltet auf V1.6 BYTE-Modus.

/X
High-Operator. Liefert den Inhalt des Highbytes einer WORD-Variable. Der Befehl ist z.B. sehr praktisch um zu testen, ob eine WORD-Variable negativ ist. Dann ist nämlich das 7.Bit des Highbytes gesetzt:
Langsam:

```
SIGN
```

```
IF X-.0
ENDIF
UNSIGN
```

Schneller (da /X als Byte behandelt wird):

```
IF /X>127
ENDIF
```

$XXXX

Hex-Operator. Die rechts vom Zeichen stehende Zahl wird als hexadekadische Zahl interpretiert. Sie können nun also anstatt A=54016 das folgende schreiben: A=$D300. Die Reihenfolge aller möglicher Operatoren ist so: -!$

Bei INPUT können jedoch keine Hex-zahlen eingegeben werden.

DATA(X)

Bisher konnte bei DATA in der Klammer nur eine Zahl X stehen, die angab, an welche Stelle die Daten geschrieben werden sollen. Nun kann auch eine Variable X angegeben werden. Die Daten werden dann an die Adresse der Variable geschrieben (nicht dorthin, wo die Variable zeigt!).

Bsp:

```
ARRAY
[STR(10)]
…
DATA(STR)
[
$1B,120,0
]
```

Damit wurde in das ARRAY STR die Zeichenfolge 27,120,0 geschrieben. So etwas ist z.B. beim Ausgeben von Steuerzeichen (auch an den Drucker) äußerst praktisch.

Änderung in der Shell:

Durch Eingeben von "D" können Sie nun von der Shell ins DOS springen. Vom DOS kommen Sie zur Shell zurück, indem Sie AUTORUN.SYS laden.

Anderungen im Editor:

Nachdem der Compiler und die Shell bereits vor einigen Monaten mit einem Update auf die Version 2.0 gebracht wurden, gibt es nun auch einen neuen Editor.

Die wichtigste Änderung am Editor ist eine Korrektur an der MERGE- Routine. Bisher kam es oft vor, daß der Editor einige Zeit nach Ausführen dieser Funktion abstürzte. Auch das versehentliche Einfügen des Anfangspfeils kann nun nicht mehr vorkommen.

Eine echte Verbesserung stellt die Möglichkeit dar, mit der ESCAPE-Taste zwischen Text- und Grafikzeichen- Modus hin und her zu schalten. Damit ist es nun einfach möglich beliebige Grafikzeichen in Texte einzubinden. Aber auch für die Darstellung der internen Zeichencodes ist das sehr praktisch. Bedenken Sie, dass das "Herzchen" den Wert 0 hat, und somit nicht mit PRINT auf den Bildschirm gedruckt werden darf! In anderen Funktionen dient ESC aber weiterhin zum Verlassen ohne Ausführung der Funktion.

Mit CONTROL-A können Sie nun die Nummer des Laufwerks beim Directory mit CONTROL-I ändern. Es wird nun "New Drive?" angezeigt. Daraufhin geben Sie 1 bis 8 für das entsprechende Laufwerk ein.

Beim Abspeichern und Laden können Sie nun die Laufwerksnummer ändern, ohne den gesamten Textnamen weglöschen und dann wieder eintippen zu müssen. Durch Drücken von SHIFT 1 bis 8 wird die entsprechende Nummer direkt hinter dem "D" eingefügt. Damit das klappt, müssen Sie immer "D1:" anstatt "D:" eingeben.

Änderung im Runtime:

Nun kann INPUT auch in Textfenstern ausgeführt werden.

Änderungen im Compiler:

Der Compiler ist so ausgelegt, daß er die 130XE-Ramdisk voll unterstütz. D.h. er kopiert das RUNTIME.OBJ-File beim ersten Compilieren in die Ramdisk (falls vorhanden), so daß der nächste Compiliervorgang dann wesentlich schneller abläuft.
Libraries können Sie selbst im DOS in die RAMDISK kopieren. Woher Sie geladen werden müssen können Sie direkt im Quelltext bei INCLUDE angeben.
Außerdem produziert der Compiler nun selbststartende Programme, d.h. wenn Sie ein compiliertes QUICK-Programm abspeichern, können Sie es im DOS laden, wobei es automatisch gestartet wird.
Die wichtigste Änderung ist aber, daß der Compiler nun 20% schnellere und kürzere Programme erzeugt, weil einige Programmteile (Vergleiche aller Art) optimiert wurden. BYTE-Vergleiche sind nun so kurz, daß sie auch in Assembler nicht schneller wären. Zu beachten ist, daß der Vergleich zweier Zahlen immer als WORD geschieht. D.h., die beliebte Endlosschleife

```
REPEAT
UNTIL 1=0
```

wird langsamer ausgeführt, als der Vergleich von zwei oder einer BYTE-Variable:

```
REPEAT
UNTIL A<>A (oder auch z.B. UNTIL A=1)
```

Diese Optimierung bringt auch mit sich, daß Sie im Interrupt kein IPUSH mehr benötigen, wenn nur BYTE-Vergleiche im Interrupt benutzt werden (siehe S.19).
Außerdem ist es jetzt nicht mehr nötig den obligatorischen Bildschirmeditor-OPEN-Befehl am Anfang eines jeden QUICK-Programms auszuführen, da dies der Compiler bei Run automatisch tut.

Neue Fehlermeldungen:

Nummer (hex)	Bedeutung
28	Runtime.Obj kann nicht geladen werden. Das File befindet sich weder auf D1: noch auf D8:
29	Label fehlt Bei JUMP wurde eine Zahl angegeben, die als Label nirgends angegeben wurde
1E	Falsche Adresse bei DATA
entfällt	entfällt

Andere DOS-Versionen
Wir haben QUICK so umgeschrieben, daß es theoretisch mit jedem DOS funktionieren könnte, wenn das DOS nicht länger als DOS2.5 ist. (Praktisch läuft es aber wohl nur mit DOS 2/2.5/3). Hätten wir die Anfangsadresse von QUICK ($1C70) geändert, wäre die neue Version zur alten

inkompatibel geworden. Das erschien uns nicht annehmbar. Eventuell ist es vielleicht möglich andere DOS-Versionen zu verwenden, wenn nur ganz wenige Buffer und Laufwerke installiert werden.

Auch bei DOS 2.5 ist zu beachten, daß nur 3 Laufwerke (einschließlich Ramdisk) angemeldet sein dürfen. Notfalls ist die DOS-Konfiguration mit dem DOS-Utility SETUP.COM auf folgende Einstellung zu ändern:
Active Drives: 1 2 8
Number of Buffers: 3

Korrektur der Fehler im Handbuch zu V1.6

S.13: *

Hinter manchen Befehlen (z.B. ?( ); und LSRW) darf kein Kommentar stehen. Er sollte deshalb grundsätzlich in eine freie Zeile geschrieben werden.

OPEN

Der Screen-OPEN-Befehl am Programmanfang ist nicht mehr notwendig

S.15: *MOUSE*

Die Speicherzellen 178/179 sollten besser MX und MY genannt werden. Dann kann man mit dem Befehl MX=10 den Mauszeiger z.B. wirklich an die X-Position 10 setzen.

S.16: *POKE(A,B)*

schreibt den Inhalt von B in die durch den Inhalt von A angegebene Adresse

PROCADR

heißt tatsächlich PADR

S.23: Speicherplan:

Der Compiler reicht nur von $1F00 bis $40FF. Dafür beginnt der Runtimeteil schon bei $4100. Außerdem wird der Bereich von 1536 bis 1792 zum Zwischenspeichern des Filenames beim Wechsel vom Compiler in den Editor benutzt. Sie können ihn aber trotzdem überschreiben.

S.24: *INLINE*

Mit dem neuen SIGN-losen Byte-Vergleich erübrigen sich INLINE-Befehle für schnelle Byte-Vergleiche!

S.25: *INPUT in beliebigen Grafikstufen*

Nun kann INPUT auch in Textfenstern ausgeführt werden.

Noch eine Information für interessierte QUICK-User:

Beim Verlag Werner Rätz gibt es in unregelmäßigen Abständen das QUICK*magazin* auf Diskette. Dort werden nützliche Tipps und Tricks, interessante Utilities, neue Updates usw. geliefert.

QUICK Referenzkarte

Shell:

E	zum Editor
C	zum Compiler
Control-Q	Kaltstart
D	zum DOS

Editor:

TAB	2 Zeichen rechts
CTRL-Delete	Zeichen löschen
Clear	Text löschen
;	zum Zeilenende
H	zum Textanfang
N	zum Textende
U	Seite hoch
D	Seite runter
X	Zeile löschen
V	Zeilen vereinigen
B	Blockanfang
E	Blockende
C	Block einfügen
F	Suchbegriff eingeben
R	Nochmal suchen
L	Text laden
M	Text dazu laden
S	Text abspeichern
I	Directory
A & 1 bis 8	Laufwerksnummer
Q	Quit

Compiler:

R	Programm starten
S	Prg. abspeichern
A	Prg. compilieren
E	zur Shell

***Quick*-Befehlsübersicht:**

Blockmarkierungen:

OLD	Auf V1.6 BYTE-Modus umschalten
INCLUDE	Libraries laden
BYTE	Variablendeklaration
WORD,ARRAY	Variablendeklaration
MAIN, ENDMAIN	Hauptprogrammblock
PROC, ENDPROC	Unterprogrammblock
INTER,ENDVBI	VBl-Block
INTER,ENDDLI	DLI-Block
INLINE	Maschinensprache-Block
DATA(adr)	Daten-Block

I/O-Befehle:

OPEN(k,a1,a2,f)	Öffnet Kanal
CLOSE(k)	Schließt Kanal
BGET(k,anz,adr)	Lädt Daten
BPUT(k,anz,adr)	Schreibt Daten
INPUT(a)	Liest Daten von E:
PRINT(a,b,...)[;]	Schreibt auf E: & S:
LPT(a,b,...)[;]	Schreibt auf Kanal 5

Grafikbefehle:

POS(x,y)	Cursor auf x,y
LOCATE(w)	Wert unter Cursor
COLOR(c)	Wählt Zeichenfarbe
SETCOL(n,f,h)	Setzt Farbwerte
PLOT(x,y)	Setzt Punkt
DRAW(x,y)	Zieht Linie nach x,y
CUT(x1,x2,y1,y2,adr)	Bildteil ausschneiden
PASTE(m,x,y,adr)	Bildteil einsetzen
PLAYER(z,i,l,q)	Kopiert Playerdaten
MOUSE	Fragt ST-Maus ab

Soundbefehle:

SOUND(k,h,v,l)	Schaltet Sound ein
DIGl(g,a,e)	Spielt Digitalsounds

Speichermanipulationen:

POKE(a,b)	Schreibt Byte in RAM
PEEK(a,b)	Liest Byte aus RAM
DPOKE,DPEEK	Schreibt/Liest Word
BMOVE(q,z,l)	Kopiert Speicherbereich
FMOVE(q,z,l)	Kopiert schneller
CLR(p,anz)	Löscht Pages

Maschinensprache-Befehle:

CALL(a,x,y,adr)	Ruft Ass-Prg auf
REGA(a),REGX(a)	Liest Prozessor-
REGY(a),REGP(a)	register

Arithmetische-Befehle:

ADD(a,b,c)	C=A+B
SUB(a,b,c)	C=A-B
a+	A=A+1 (a ist Byte)
a-	A=A-1 (a ist Byte)
MULT(a,b,c)	C=A*B
DIV(a,b,c)	C=A/B
AND(a,b,c)	C=A AND B
OR(),EOR()	wie bei AND
ASLW(a),ASLB(a)	1 Bit nach links
LSRW(a),LSRB(a)	1 Bit nach rechts

ASRW(a),ASRB(a)	1 Bit nach rechts mit Vorz.

Kontrollstrukturen:

IF <Vergleich> [ELSE] ENDIF
WHILE <Vergleich> WEND
REPEAT UNTIL <Vergleich>
JUMP(x)

Sonstiges:

SYNC(n)	Zeilendesynchronisation
PADR(n),VADR(n)	Unterprogr./Variabl.adr.
DLI(n)	DLI anschalten
VBI(n)	VBI anschalten
.n	Unterpragramm rufen
PUSH,PULL	Register retten
IPUSH,IPULL	Register retten
ZPUSH,ZPULL	Register retten
SIGN	Vorzeichen beachten
UNSIGN	Vorzeichen nicht beachten