SERIE

QUICK

Der neue Wirbelwind für Ihren XL/XE

Heute kommen wir zu den wichtigsten Bestandteilen von Quick, nämlich zum Compiler und zum Befehlssatz. Dabei wollen wir nicht nur auf die Bedienung des Compilers und die Syntax der Sprache eingehen, sondern uns auch mit dem Aufbau und der Arbeitsweise des Compilers befassen. Die beiden Programme, die zum Compiler gehören, finden Sie dann im nächsten **ATARI***magazin*.

Dem Compiler auf die Finger geschaut

Der Compiler hat die Aufgabe, Quick-Quelltexte, die Sie mit dem Editor erstellt haben, in lauffähige Maschinenprogramme zu übersetzen. Diese dürfen nicht zu lang werden (wie es bei kompilierten Programmen oft der Fall ist) und sollen vor allem schnell sein. Es ist klar, daß ein Compiler diese Forderungen nicht hundertprozentig erfüllen kann. Man muß also einen brauchbaren Kompromiß finden.

Zunächst wollen wir uns mit der Arbeitsweise des Compilers beschäftigen. Es handelt sich um einen 2-Pass-Compiler. Das bedeutet, daß ein Text in drei Durchgängen in ein Programm übersetzt wird.

Pass 1

Hier werden alle verwendeten Variablen in eine Tabelle eingetragen. Das gleiche gilt für Unterprogramme. Dabei wird auch die korrekte Gliederung des Programms überprüft.

Pass 2

Bei diesem Durchgang kommt es zur eigentlichen Übersetzung in Maschinensprache. Dabei unterscheidet der Compiler drei verschiedene Arten.

1. Wertzuweisung

Darunter versteht man jeden Transfer von einer Variable in eine andere, also z.B.:
A=B oder A=10

Dieser Transfer von Werten wird optimal in Maschinensprache übersetzt:

"Quick" heißt die neue Programmiersprache für XL/XE-Computer. In dieser Folge setzen wir uns mit Programmierung und Arbeitsweise des Compilers auseinander. Damit können Sie nicht nur mit "Quick" arbeiten sondern erfahren auch, was sich "hinter den Kulissen" tut.

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

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

2. Makros

Einfache Befehle werden durch fertige Programmteile in Maschinensprache übersetzt. Dies entspricht der Vorgehensweise eines echten (nicht Atmas II) Makroassemblers. Dazu ebenfalls ein Beispiel:

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 Runtime-Teil auf. Dieser ca. 3 KByte umfassende Block enthält die quickinternen Unterprogramme. Er wird während des Kompilierens nachgeladen und ins Programm eingebaut. Somit hat jedes Quick-Programm eine Mindestlänge von 3 KByte. (Die Runtime-Bibliothek wird im nächsten Heft abgedruckt.)

Diese Befehle sind die "langsamsten". Zunächst müssen Variablen übergeben werden (auf verschiedene Arten), dann erfolgt der Aufruf des Unterprogramms, und schließlich sind eventuelle Rückgabevariablen zu übertragen. Andererseits bestehen die Routinen dann aber auch wieder zu 100 % aus Maschinensprache.

Viele in der Assemblerecke veröffentlichte Routinen finden sich in modifizierter Form im Runtime-Teil wieder; sie sind fester Bestandteil von Quick. DIGI(G,A,E) beispielsweise spielt digitalisierte Sounds, und MOUSE fragt eine ST-Maus ab.

Die Bedienung des Compilers

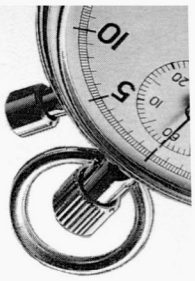
Zunächst müssen Sie den File-Namen des zu kompilierenden Programms eingeben. Falls Sie vorher den Editor verlassen haben, wird der dort verwendete File-Name gleich angezeigt. Dann wird kompiliert. Dabei erscheint die gerade bearbeitete Zeile. Aufgrund der Geschwindigkeit sieht man sie aber meist nur aufblitzen.

Nach erfolgreichem Kompilieren haben Sie dann folgende Auswahl:

Another File: noch ein File kompilieren

Exit: Sprung in die Shell

Save: Abspeichern des kompilierten Programms unter dem gleichen Namen mit Extender .OBJ



Run: Startet ein kompiliertes Programm. Eine Rückkehr in den Compiler ist möglich, wenn man nur Speicherbereiche vor seinem Programmbeginn benutzt. Die Endadresse des Programms steht nach erfolgreichem Kompilieren unter dem kleinen Auswahlmenü.

Falls beim Kompilieren ein Fehler auftritt, erscheint eine entsprechende Meldung mit Fehlernummer und der betreffenden Zeile. Außerdem steht hinter der Fehlernummer der Name des Unterprogramms, in dem der Fehler vorkam. Dazu gleich ein Beispiel:

Error #05 in FARBEN (C)ont (E)xit

Diese Meldung besagt, daß im Unterprogramm FARBEN ein unbekannter Befehl verwendet wurde. Befindet sich der Fehler nicht in einem Unterprogramm, so wird MAIN angezeigt.

Nun können Sie mit Exit den Compiler verlassen oder mit Cont den Kompiliervorgang fortsetzen. Im zweiten Fall ist das kompilierte Programm dann natürlich nicht fehlerfrei; außerdem können Folgefehler auftreten.

Aufbau eines Quick-Quelltextes

Damit kommen wir jetzt zum wichtigsten Teil. Wie müssen Quick-Quelltexte aussehen? Jeder von ihnen hat folgenden Aufbau:

A) Include-File-Namen-(eventuell)
B) Variablendeklarationsteil
C) Hauptprogramm (MAIN)
D) Unterprogramme (PROC)

Include-File-Namen

Mit dem Include-Befehl lassen sich Libraries (Unterprogramme) beim Kompilieren nachladen:

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

Dieser Aufbau (Keyword, [,File-Namen,]) ist typisch und immer exakt einzuhalten. Man muß auch stets beachten, daß nur ein Befehl pro Zeile verwendet werden darf.

Variablendeklarationsteil

In diesem Teil des Hauptprogramms müssen alle globalen Variablen deklariert werden. Bei ihnen handelt es sich um Variablen, die im gesamten Programm bekannt sind und sich somit überall verwenden lassen. Dem Compiler sind sie ganz am Anfang mitzuteilen.

Im Gegensatz zu Basic gibt es zunächst keine Fließkommavariablen. Stattdessen existieren drei andere Arten.

1-Byte-Variablen: BYTE-Variablen. Sie belegen nur 1 Byte im Speicher.

2-Byte-Variablen: WORD-Variablen. Sie belegen 2 Byte im Speicher.

Felder: ARRAY-Variablen. Sie belegen 1 bis 255 Byte (je nach Deklaration). Diese Arrays können sowohl als Strings wie auch als eindimensionale Felder benutzt werden.

Der Aufbau des Deklarationsteils muß folgendermaßen aussehen:

```
Typ
[
Variablenname,Variablenname,...
...
]
```

```
PMBASE=112
GRAC=3
PCOL1=120
PCOL2=120
PL=29696
DLI(DISP)
DMA=34
.PAUSE(150)
POS(0,1)
PRINT("Auch ein VBI z.B. fuer eine
PRINT("Playerbewe-")
PRINT("gung ist ganz einfach")
X=48
Y=100
PDAT=25600
PDAT1=25613
FLAG=0
DMA=62
VBI(GNOM)
.PAUSE(150)
POS(0,1)
PRINT("Und das Hauptprogramm kann ");
PRINT("sich dabei  ")
PRINT("noch mit Grafiken beschaefti");
PRINT("gen  ")
POKE (87,8)
BS=32988
FARBE=1
REPEAT
   Z=0
   Z1=319
   COLOR(FARBE)
   REPEAT
      PLOT (Z,0)
      DRAW (Z1,59)
      ADD (Z,6,Z)
      SUB (Z1,6,Z1)
   UNTIL Z>=319
   FARBE+
UNTIL FARBE=255
ENDMAIN


INTER DISP
BEGIN
   PUSH
```

Die Typen sind BYTE, WORD, ARRAY:

```
BYTE
[
A1,F3,DA_S1
INT
]
WORD
[
W1,WO
]
ARRAY
[
FELD(10),TEXT(40)
]
```

Bei Feldvariablen ist hinter dem Namen die Länge des Feldes (1 bis 255) anzugeben. Der Compiler weist jeder Variablen einen Speicherplatz in einem dafür vorgesehenen Bereich zu. Es existiert aber auch die Möglichkeit, direkt mitzuteilen, an welche Stelle die Variable gelegt werden soll. Dies bietet mitunter große Vorteile. Auch dazu ein Beispiel:

```
BYTE
[
COL1=708, COL2=709
]
```

Nun kann man später im Programm statt SETCOL(0,1,10) einfach COL1=26 schreiben. Das geht natürlich viel schneller und benötigt weniger Platz. Auf diese Weise werden POKE und PEEK (die ebenfalls vorhanden sind) in vielen Fällen überflüssig.

Hauptprogramm

Es beginnt mit MAIN und endet mit ENDMAIN. Dazwischen steht das Programm, das aus allen möglichen Quick-Befehlen bestehen kann.

Unterprogramme

Sie beginnen mit PROC Unterprogrammname und enden normalerweise mit ENDPROC. Der erste Teil des Unterprogramms besteht aus der Deklaration der lokalen Variablen, die in drei Gruppen gegliedert sind: IN, OUT, LOCAL.

Bei IN sind die Variablen zu deklarieren, die beim Aufruf des Unterprogramms vom rufenden Programm an dieses übergeben werden. Bei OUT sind dann die Variablen zu deklarieren, die ans rufende Programm zurückgegeben werden sollen. (Wichtig! Reihenfolge und Anzahl der Variablen sind exakt einzuhalten.)

Bei LOCAL werden zusätzliche Variablen deklariert, die nur interne Verwendung im Unterprogramm finden sollen. Es ist also auch möglich, im Unterprogramm Variablen mit den

gleichen Namen wie im Hauptprogramm zu benutzen, die dann aber nur in diesem Unterprogramm bekannt sind. Sie können somit in jedem Unterprogramm Schleifen mit I als Zählvariable verwenden, die sich gegenseitig nicht stören. Besonders wichtig sind lokale Variablen für die Libraries (dazu später mehr).

Rekursive Aufrufe sind allerdings nicht möglich. Globale Variablen dürfen natürlich auch im Unterprogramm Verwendung finden.

Der Befehl BEGIN beendet den Deklarationsteil und leitet den Befehlsteil ein. Nun wollen wir uns wieder ein Beispiel anschauen:

```
PROC BEISPIEL
IN
BYTE
[
VAR1
]
OUT
BYTE
[
VAR2
]
WORD
[
WERT1
]
LOCAL
ARRAY
[
TEXT
]
BEGIN
...
ENDPROC
```

Beim Aufruf des Unterprogramms ist also ein Wert zu übergeben, und zwei Variablen sind entgegenzunehmen. Der Aufruf erfolgt so:

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

oder

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

Beachten Sie bitte den Punkt! Er macht den eigentlichen Aufruf aus. Falls Sie eine Variable zweimal zurückbekommen möchten, so erhält sie den Wert der zweiten OUT-Variable. Beim Aufruf .BEISPIEL(10,W2,W2) hat W2 also den Wert von WERT1.

Wertzuweisungen

In Quick gibt es keine Möglichkeit, Terme, also numerische Ausdrücke, einer Variablen zuzuweisen. Folgendes ist beispielsweise falsch:

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

Bei Wertzuweisungen in Quick darf rechts vom Gleichheitszeichen nur eine Zahl, eine Variable oder ein Text stehen. Richtig ist also folgendes:

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

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

	mit Vorzeichen (signed)
BYTE	-128 bis +127
WORD	-32768 bis +32767

	ohne Vorzeichen (unsigned)
BYTE	0 bis 255
WORD	0 bis 65535

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

Bei der Verarbeitung von Arrays sind einige Dinge zu beachten.

Array als String

Man kann einem Array direkt einen Text zuweisen. (Falls dieser länger ist als die Dimensionierung, wird er abgeschnitten.) Dies sieht dann so aus:

```
TEXT="Hallo"
```

Der Inhalt eines Arrays läßt sich außerdem direkt in ein anderes kopieren:

```
FELD=TEXT
```

Man kann auch indiziert zuweisen:

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

Dies ergibt dann folgendes:

```
"ABCHallo"
```

Eine Überschreitung der Dimensionierung findet dabei keine Beachtung! Bei indizierter Zuweisung dürfen aber keine Texte zugewiesen werden. Dies ist also falsch:

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

Das Ende eines Strings wird durch eine Null gekennzeichnet. Deshalb müssen Sie ihn immer um eins länger dimensionieren, als es eigentlich nötig ist.

Array als CHR$-Ersatz

In Basic löscht beispielsweise folgende Eingabe den Bildschirm:

```
PRINT CHR$(125)
```

In Quick schreiben wir:

```
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 selbst entscheiden müssen, ob Sie 1- oder 2-Byte-Zahlen benutzen.

FELD(0)=1000: Damit sind FELD(0) und FELD(1) belegt.

FELD(0)=100: 8 Bit; nur FELD(0) ist belegt.

FELD(0)=!100: 16 Bit; FELD(0) und FELD(1) werden belegt. Mit dem Ausrufezeichen erzwingt man also 16 Bit. Bei negativen Zahlen gilt folgende Notation:

```
FELD(0)=-!100
```

Beachten Sie dies bitte: Zuerst kommt das Vorzeichen!

Die Befehle

Die Befehle

Zum festen Sprachschatz von Quick gehören rund 60 Kommandos. Einige dürften Ihnen von Basic bekannt sein; andere erinnern eher an Assembler oder C. Der Compiler bindet die Befehle entweder direkt in den fertigen Programmcode ein oder ruft ein entsprechendes Unterprogramm aus dem Runtime-Teil auf. Letzteres geschieht bei besonders aufwendigen Kommandos. Sie sind im folgenden mit einem Klammeraffen (@) versehen.

* Kommentar

Das Sternchen kennzeichnet einen Kommentar. Er kann allein in der Zeile oder auch rechts von einem Befehl stehen. Der Quelltext darf übrigens auch Leerzeilen enthalten.

OPEN (NR,AUX1,AUX2, NAME)

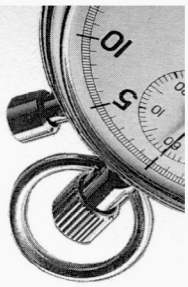
Öffnet Kanal NR mit den Parametern AUX1, AUX2 und dem Dateinamen NAME. Auch dazu gleich ein Beispiel:

```
OPEN (1,4,0,"D:TEXT.TXT")
```

Wie bei den meisten anderen Befehlen dürfen die Parameter nur BYTE- oder WORD-Variablen bzw. Zahlen sein. Arrays können nur als Strings beim Aufruf verwendet werden, nicht jedoch als Integer-Variable in der Form FELD(1). Das bedeutet, daß eine Indizierung beim Befehlsaufruf nicht möglich ist. Folgendes ist also falsch:

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

Korrekt ist diese Schreibweise:



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

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 ist jetzt möglich, INPUT dagegen nicht. Sie können aber auch schreiben:

```
OPEN (6,12,0,"E:")
```

Nun sind PRINT und INPUT möglich.

Diesen Befehl sollte man bei jedem Programm verwenden, es sei denn, die GRAPHICS-Routine aus der Grafik-Library wird benutzt.

CLOSE (NR)

Dieser Befehl schließt den Kanal NR.

BGET (NR,ANZ,ADR)

Liest ANZ Bytes von Kanal NR ab der Speicherstelle ADR. Diese Anweisung dient oft auch zur Eingabe von Zeichen von der Tastatur bei nicht geöffnetem Editor und zur Umgehung von INPUT.

BPUT (NR,ANZ,ADR)

Dieser Befehl schreibt ANZ Werte ab ADR auf Kanal NR.

INPUT (A) @

Dient zur Eingabe einer Zahl (A ist BYTE oder WORD) bzw. eines Textes (A ist ARRAY).

Noch ein wichtiger Hinweis. Der INPUT-Befehl funktioniert nur, wenn man zuvor einen Editorkanal geöffnet hat!

PRINT (A1,A2,A3,...)
bzw.
?(A1,A2,A3,...) @

Schreibt auf den Bildschirm. In der Klammer können beliebig viele Parameter angegeben werden, die durch Kommas zu trennen sind. Hier ist die Verwendung von BYTE, WORD, gesamten Arrays und Texten möglich:

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

Nach Zahlen wird jeweils ein Leerzeichen eingefügt. Am Ende des Befehls kann man einen Strichpunkt anhängen. Dann wird nach dem PRINT kein RETURN ausgeführt, und der nächste PRINT beginnt direkt hinter dem vorherigen:

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

Damit erhalten Sie folgendes:

Hallo, Welt

SIGN

Bei PRINT werden Variablen mit Vorzeichen ausgegeben. Bei Vergleichen finden die Vorzeichen Beachtung.

UNSIGN

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

POS (X,Y)

Setzt den Cursor an die Stelle X,Y.

Rund 60 Kommandos gehören zum Sprachumfang von "Quick"

LOCATE (WERT) @

Liest den Inhalt des Bildschirms an der Stelle des Cursors in WERT.

COLOR (A)

Wählt Farbregister 0 bis 4 für PLOT und DRAW.

PLOT (X,Y) @

Setzt einen Punkt in der gewählten Farbe an die Stelle X,Y.

DRAW (X,Y) @

Zieht eine Linie an die Stelle X,Y.

PLAYER (Z,I,L,Q) @

Überträgt L Daten ab Adresse Q in die Page Z, wobei I als

Index zur Zieladresse addiert wird.

Dieser Befehl eignet sich sehr gut, um Player-Daten in den Player-Bereich zu kopieren. Dabei kann I sozusagen als Y-Position verwendet werden. (I darf nur von 0 bis 255 betragen!)

CLR (P,A) @

Löscht ab Page P A*256 Bytes.

CUT (X1,Y1,X2,Y2,ADR) @

Schneidet ein Rechteck mit den Eckpositionen X1,Y1 und X2,Y2 aus und legt es in den Speicher ab Adresse ADR.

Dieser Befehl funktioniert in GRAPHICS 8, bei geradem X1 und X2 auch in GRAPHICS 15,7 (dabei X1 und X2 einfach verdoppeln!).

PASTE (M,X1,Y1,ADR) @

Kopiert die Paste-Daten in den Bildschirm. Die linke obere Ecke gibt man durch X1,Y1 an. Bei M = 0 wird einfach überschrieben, bei M = 1 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, 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) @

Läßt digitalisierte Sounds erklingen. Die Daten werden von Adresse A bis E vorgespielt. Bei A und E findet nur das High-Byte Beachtung. Als Geschwindigkeit kann man 1 bis 255 (schnell – > langsam) angeben. Dann ist der Bildschirm abgeschaltet. Bei G = 0 geschieht dies zwar nicht, dafür ist aber auch nur ein konstantes Tempo möglich.

Die Sounddaten müssen im "Soundsampler"-Format des **ATARI***magazins* vorliegen. Dabei ist ein Byte in 4 Bits aufgeteilt, die jeweils einen Soundwert (0 bis 15) darstellen.

MOUSE @

Liefert die Position einer in Port 2 angeschlossenen ST-Maus in den Speicherzellen 178 (MOUSEX) und 179 (MOUSEY). Die Maus wird so lange abgefragt, bis ihr linker Knopf gedrückt wird.

Durch das Eintragen von Werten in 178 und 179 läßt sich die Maus auch setzen. Der Mauszeiger muß getrennt mit Hilfe des PLAYER-Befehls, z.B. im VBI (dazu später mehr), dargestellt werden.

```
DATA (ADR)
[
1,4,876,4563,34,...
]
```

Schreibt die Daten ab Position ADR in den Speicher.

POKE (A,B)

Schreibt den Wert von A in Speicherzelle B.

PEEK (A,B)

Schreibt den Inhalt der Speicherzelle A (8 Bit Inhalt) in Variable B.

DPEEK, DPOKE

Hier geschieht das gleiche wie beim POKE- bzw. PEEK-Befehl, nur mit 16 Bit.

BMOVE (Q,Z,L) @

Kopiert einen Speicherbereich ab Adresse Q und mit der Länge L an die Adresse Z (bis Z+L-1). Überlappende Blöcke werden problemlos übertragen.

FMOVE (Q,Z,L) @

Wie BMOVE, nur muß L kleiner 256 sein. Überlappende Blöcke werden eventuell nicht richtig kopiert. Der Befehl ist schneller als MOVE.

CALL (A, X, Y,ADR) @

Ruft ein Maschinenprogramm ab Adresse ADR auf. Zuvor 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 Pro-

gramm. Hier lassen sich auch Variablen verwenden, wobei deren Adresse eingesetzt wird.

Dieser Befehl dient zum einfachen Einbinden von kurzen Maschinenspracheteilen, besonders auch für DLIs (dazu später mehr).

REGX (Z)
REGY (Z)
REGA (Z)
REGP (Z)

Diese Befehle übertragen X, Y, den Akku oder das Statusregister in Variable Z. Vorsicht, REGP verändert den Akku! Folgende Schreibweise ist falsch:

```
REGP(VAR1)
REGA(VAR2)
```

Richtig heißt es:

```
REGA(VAR2)
REGP(VAR1)
```

PROCADR(Unterprogrammname)

Nach dem Aufruf enthalten die Speicherzellen $D0 und $D1 die Adresse des entsprechenden Unterprogramms.

ADD(A,B,C)

Entspricht C=A+B. Überläufe werden nirgends überprüft.

A+

Erhöht die 8-Bit-Variable A um 1.

SUB(A,B,C)

Entspricht C=A−B.

A−

Erniedrigt die 8-Bit-Variable A um 1.

MULT(A,B,C) @

Entspricht C=A*B.

DIV(A,B,C) @

Entspricht C=A/B.

AND(A,B,C)

C=A AND B (bitweise).

OR(A,B,C)
EOR(A,B,C)

Auch diese Vergleiche funktionieren wie gewohnt.

ASLW(A)

Schiebt den Inhalt der Variablen A (16 Bit) um ein Bit nach links.

ASLB(A)

Schiebt den Inhalt der Variablen A (8 Bit) um ein Bit nach links.

ASRW(A)

Schiebt den Inhalt von A (16 Bit) vorzeichenrichtig nach rechts.

ASRB(A)

Schiebt den Inhalt von A (8 Bit) vorzeichenrichtig nach rechts.

LSRW(A)

Schiebt A (16 Bit) um ein Bit nach rechts (ohne Vorzeichenbeachtung).

LSRB(A)

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

Verzweigungen und Schleifen

Im Text lassen sich Labels setzen, die als Sprungziele dienen. Ein Label besteht aus einem Minuszeichen und einer Zahl von 0 bis 384, also z.B.:

```
−10
```

Die Labels können dann mit JUMP angesprungen werden:

```
JUMP (10)
```

Eleganter und strukturierter geht es aber mit folgenden Befehlen:

```
IF Vergleich ...
( ELSE
  ... )
  ENDIF

  REPEAT
  ...
  UNTIL Vergleich
```

und

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

Ein Vergleich ist dabei ein Ausdruck der Form WERT OPERATOR WERT. Dazu ein 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. Falls sie falsch ist, erfolgt gleich ein Sprung zum ersten Befehl hinter WEND. Ansonsten wird der Block so lange wiederholt, bis die Bedingung nicht mehr wahr ist. Bei REPEAT...UNTIL erfolgt dagegen auf jeden Fall ein Durchlauf, da die Bedingung erst am Ende überprüft wird.

Schleifen lassen sich beliebig schachteln.

Interruptprogrammierung

Nun kommen wir zu etwas ganz Besonderem. In Quick kann man ein normales Unterprogramm als DLI oder VBI verwenden. Dieses 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 selbst alle (CPU-) Register retten und am Ende wieder herstellen. Dafür gibt es die Befehle PUSH und PULL. Auch dazu gleich wieder ein Beispiel:

```
MAIN
...
VBI(INTERRUPT)
DLI(FARBEN)
...
ENDMAIN

INTER INTERRUPT
...
BEGIN
...
ENDVBI

INTER FARBEN
...
BEGIN
PUSH
...
PULL
ENDDLI
```

Auf diese Weise hat man aber nur prozessorinterne Register gerettet. Falls Sie mit @ gekennzeichnete Routinen im Interrupt verwenden, müssen Sie auch compilerinterne Variablen retten. Dies geschieht folgendermaßen:

```
IPUSH
ZPUSH
...
IPULL
ZPULL
```

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

Bei Interrupt-Unterprogrammen lassen sich nur LOCAL- (und globale) Variablen verwenden, jedoch keine IN- und OUT-Variablen.

Fehlermeldungen des Compilers

Der Compiler erzeugt während des Übersetzens eine Reihe von Fehlermeldungen. Diese sind jedoch nicht immer ganz eindeutig und beziehen sich eventuell auch nicht auf die richtige Zeile.

Das laufende Maschinenprogramm selbst erzeugt keine Fehlermeldungen, höchstens Abstürze.

Die Fehlermeldungen wollen wir Ihnen nun im einzelnen vorstellen. Beachten Sie bitte, daß die Nummern hexadezimal angegeben sind.

Nr. Bedeutung

1 [fehlt.

2 Name bei PROC fehlt.

3 MAIN zweimal verwendet. In einem Quick-Programm darf es nur ein Hauptprogramm geben.

4 Kein MAIN oder PROC vor MAIN. Reihenfolge Hauptprogramm – Unterprogramme muß eingehalten werden.

5 unbekannter Befehl

6 ungültiger Wert

7 Unbekannte Variable. Variable wurde nicht deklariert.

8 Ungültige Wertzuweisung. Falscher Variablentyp; einer Variablen wurde z.B. ein Text zugewiesen.

9 Zahl zu groß

A Längenangabe bei ARRAY-Deklaration fehlt. Jedes Array kann 1 bis 255 Einträge lang sein.

B Längenangabe bei ARRAY- Deklaration zu groß (>255)

C Ungültiger Wert als Index. Falscher Variablentyp als Index oder falscher Wert.

D Unbekanntes Unterprogramm. Es wird ein Unterprogramm aufgerufen, das nicht definiert ist. Vielleicht fehlt aber auch der INCLUDE-Befehl.

E Zu wenige Parameter bei Unterprogrammaufruf. Es müssen immer so viele IN- und OUT-Variablen, wie im Unterprogramm deklariert, übergeben werden.

F interner Fehler

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. ENDIF, UNTIL oder WEND gefunden, ohne zugehörigen Anfang.

1D Text kann nicht geladen werden. Falscher File-Name beim Compiler eingegeben.

1E Falsche Adresse bei DATA. Keine Variable als Adresse verwenden.

1F Label-Nummer ungültig. Wert zu groß oder negativ.

20 kompiliertes Programm zu lang

21 zu viele Variablen

22 zu viele Unterprogramme

23 zu viele Unterprogrammaufrufe oder Label-Sprünge

24 zu viele Include-Files

25 BEGIN fehlt. BEGIN muß nach der Variablendeklaration im Unterprogramm stehen.

26 Falsche Variablenart. Bei Befehlsaufruf falsche Variablenart als Parameter benutzt.

27 ENDIF, UNTIL oder WEND fehlt. Dieser Fehler tritt erst am Ende eines Programmteils (END-MAIN, ENDPROC, ...) auf. Überprüfen Sie den Programmteil auf die richtige Verschachtelung.

Im nächsten Teil folgen dann das Compiler-Programm sowie Tips und Tricks zu Quick.

Andreas Binner
Harald Schönfeld