Basic-Compiler in Basic

Der Mikro-Compiler erzeugt reine Maschinensprache. Ergebnis: Programme laufen hundertmal schneller.

Assembler oder Basic? Das ist die Frage bei jedem Softwareprojekt. Beide Sprachen haben Vor- und Nachteile. Assembler ist extrem schnell und flexibel, aber sehr aufwendig zu programmieren und zu testen. Da hat man's mit Basic leichter. Als Hochsprache erspart es einem die Beschäftigung mit maschinenspezifischem Kleinkram. So kann man sich voll auf die Anforderungen der Anwendung konzentrieren. Wenn nur nicht das Schleichtempo wäre, mit dem Basic durch die Zeilen kriecht! Im Schnitt sind Programme in Basic mehrere hundertmal langsamer als solche in Assembler.

Schnell wie Assembler

Einen Ausweg aus diesem Dilemma bietet in vielen Fällen ein Compiler. Dabei handelt es sich um ein Programm, das Hochsprachenprogramme in Maschinensprache übersetzt. Der erzeugte Code ist zwar nicht so optimal wie handgeschriebener Assemblercode, aber immerhin oft zehnmal schneller als interpretiertes Basic.

Unser Mikro-Compiler kommt noch näher an das Tempo von Maschinensprache heran. Im Schnitt laufen Programme nach der Übersetzung mit hundertfachem Tempo! Allerdings verarbeitet der Compiler nur eine Untermenge des normalen Basic, genannt Mikro-Basic. Er erzeugt reine Maschinensprache. Da Mikro-Basic im Commodore-Basic enthalten ist, kann man Programme mit dem Basic-Interpreter entwickeln und austesten.

Der Compiler selbst ist in Basic geschrieben. Wer wissen will, wie man ein Übersetzungsprogramm programmiert, hat hier ein gutes Studienobjekt.

Nach dem Start verlangt der Compiler den Namen des Programms, das er bearbeiten soll. Wird hier nur Return gedrückt, so wird „TEST.COMP" als Filename genommen. (Falls ein anderer Voreinstellungswert gewünscht ist, muß Zeile 1820 entsprechend geändert werden.) Bei Eingabe von „*" als Name wird ins Basic zurückgekehrt.

Als nächstes ist die Adresse zu nennen, ab der der Maschinencode im Speicher abgelegt werden soll. Gibt man hier nur Return ein, so wird 49152 als Adresse genommen.

Der Compiler liest nun das Programm von Diskette und übersetzt es Zeile für Zeile. Parallel dazu listet er den Quelltext auf den Bildschirm. Auch Basic-Tokens werden dabei richtig ausgegeben. Dazu wird die entsprechende Betriebssystemroutine benutzt. Drückt man beim Kompilieren die Shifttaste, wird der Übersetzungsvorgang angehalten.

Wenn im Source Fehler entdeckt werden,wird eine Meldung in reverser Schrift ausgegeben. Da angenommen wird, daß das Programm mit dem Interpreter ausgetestet worden ist, wird nur ein Minimum an Überprüfungen vorgenommen. Wenn ein Fehler gefunden wird, ist es in der Regel ein in Mikro-Basic nicht zulässiger Befehl. Bei den meisten Errormeldungen werden zusätzlich zwei Werte angezeigt: Eine Nummer, die angibt, bei welchem Byte der Fehler entdeckt wurde, und der Inhalt dieses Bytes. Die Bytenummer bezieht sich dabei auf einen Puffer ab Adresse 828, in dem jeweils ein Befehl bearbeitet wird. Bei einigen Fehlern steigt der Compiler mit einer Fehlermeldung aus.

Nach der Kompilierung werden Start- und Endadresse des Maschinencodes, die Anzahl der Fehler und die für die Übersetzung benötigte Zeit ausgegeben. Dann erscheint ein Menü mit folgenden Wahlmöglichkeiten:

1. Save: Der erzeugte Maschinencode kann abgespeichert werden. Dazu ist der gewünschte Filename einzugeben. Drückt man hier nur Return, so wird das Kompilat unter dem Namen des Quelltextes und mit dem Zusatz „.ML" (machine language) abgespeichert. Existiert bereits ein File mit dem eingegebenen Namen auf der Diskette, wird es vor dem Abspeichern des neuen Programms automatisch gelöscht. Der Maschinencode kann später mit ..,8,1 geladen und mit SYS an die gewählte Startadresse gestartet werden. Das kann sowohl im Direkt- als auch im Programmodus erfolgen.

Starke Teilmenge

2. Execute: Das kompilierte Programm wird gestartet.
3. Compile: Ein anderes Programm wird kompiliert. Die Prozedur erfolgt wie oben beschrieben.
4. Quit: Rückkehr ins Basic.

Da ein Compiler für das vollständige Basic ein sehr großes Programm ergeben würde, beschränkt sich der Mikro-Compiler auf eine Teilmenge dieser Sprache. Bevor die Syntax von Mikro-Basic erklärt wird, müssen einige Begriffe eingeführt werden:

V steht für einen Variablennamen. Variablen müssen sich im ersten Buchstaben unterscheiden. Es sind also insgesamt 26 Variablen möglich. Sie werden im Speicherbereich von 680 bis 731 abgelegt. ▶

Ihr Wert muß zwischen 0 und 65 535 liegen.
N bezeichnet eine Zahl im Bereich von 0 bis 65 535.
X repräsentiert eine Variable (V) oder eine Zahl (N).
Ausdr ist ein numerischer Ausdruck, der mit X oder PEEK(X) beginnt, wobei X wieder für eine Zahl oder Variable steht. Folgen können beliebig viele der folgenden Terme:
+ X
— X
* X
/ X
AND X
OR X
Ein Ausdruck wird von links nach rechts ausgewertet. Klammern sind nicht erlaubt. Die vom Basic bekannte Hierarchie von Operatoren ist aufgehoben. So ergibt 2*3+4 wie üblich 10, 4+2*3 aber 18. Damit Programme mit dem Commodore-Basic kompatibel sind, müssen Ausdrücke so angeordnet werden, daß Punkt- vor Strichrechnung und diese vor AND/OR steht.
Kmpr steht für einen der folgenden Komparatoren: = (gleich), > (größer), < (kleiner), <> (ungleich).

Klein, aber oho

Hier nun die Liste der Befehle von Mikro-Basic. Eckige Klammern kennzeichnen Terme, die auch entfallen können:
1. [LET] V=Ausdr Wie in Basic, kann das LET bei Wertzuweisungen auch weggelassen werden.
2. PRINT [Ausdr][CHR$(Ausdr)] ["string"] [;]
Das PRINT-Kommando wird benutzt, um numerische Ausdrücke, ASCII-Zeichen (CHR$) oder Strings zu drucken. Will man eine beliebige Kombination dieser Terme ausgeben, so setzt man zwischen sie ein Semikolon. Am Ende des PRINT-Befehls verhindert das Semikolon das Vorrücken in die nächste Zeile.
3. IF Ausdr Kmpr Ausdr THEN [Befehle oder Zeilennummer]
IF..THEN-Befehle dürfen nicht verschachtelt werden.

```
LET a   = b          = PEEK(b)     * c          / c
        LDA b        LDA b         STA 97       STA 97
        LDX b+1      LDX b+1       STX 98       STX 98
                     STA $22       LDA c        LDA c
        LET a        STX $23       LDX c+1      LDX c+1
                     LDX #0        CLC          SEC
        STA a        LDY #0        JSR muldiv   JSR muldiv
        STX a+1      LDA(22),Y
        + c          - c           AND c        OR c
        CLC          SEC
        ADC c        SBC c         AND c        ORA c
        TAY          TAY           TAY          TAY
        TXA          TXA           TXA          TXA
        ADC c+1      SBC c+1       AND c+1      ORA c+1
        TAX          TAX           TAX          TAX
        TYA          TYA           TYA          TYA
IF      a = b        a > b         a < b        a < > b
        LDA a        LDA a         LDA a        LDA a
        LDX a+1      LDX a+1       LDX a+1      LDX a+1
        STA 34       STA 34        STA 34       STA 34
        STX 35       STX 35        STX 35       STX 35
        LDA b        LDA b         LDA b        LDA b
        LDX b+1      LDX b+1       LDX b+1      LDX b+1
        CPX 35       CPX 35        CPX 35       CPX 35
        BEQ +4       BEQ +4        BEQ +4       BEQ +4
        BNE endif    BCS endif     BCC endif    BEQ endif
        BNE +6       BCC +6        BCS +6       BNE +6
        CPA 34       CPA 34        CPA 34       CPA 34
        BNE endif    BCS endif     BCC endif    BEQ endif
        BNE endif    BEQ endif     BEQ endif    BEQ endif
PRINT   a            CHR$(a)                    "string"
        LDA a        LDA a                      LDA #<string
        LDX a+1      LDX a+1                    LDY #>string
        STA 34       JSR basout                 JSR prtstr
        TAX                                     CLC
        LDA 34                                  BCC endstr+1
        JSR prtint                      string  ASC"string"
        LDA #13                         endstr  BRK
        JSR basout
SYS     a                            FOR a = b TO c STEP d
        LDA a                                   LDA b
        LDX a+1                                 LDX b+1
        STA 20                                  JMP start
        STX 21                          loop    LDA a
        JSR sysin                               LDX a+1
                                                STA 34
                                                STX 35
POKE    a,b                                     LDA c
                                                LDX c+1
        LDA b                                   CPX 35
        LDX b+1                                 BEQ +4
        STA 34                                  BCS contu
        STX 35                                  BCC +6
        LDA a                                   CPA 34
        LDX a+1                                 BEQ +2
        LDY #0                                  BCS# contu
        STA(34),Y                               JMP next+3
                                        contu   LDA d
andere                                          LDX d+1
                                                CLC
GOTO    JMP n                                   ADC a
GOSUB   JSR n                                   TAY TXA
RETURN  RTS                                     ADC a+1
END     RTS                                     TAX TYA
STOP    RTS                             start   STA a
REM     kein Code                               STX a+1
                                        NEXT
                                        next    JMP loop
```

Benutzte Adressen:

start	$C000 (default)	prtint	$BDCD
muldiv	$C003 (default)	prtstr	$AB1E
basout	$FFD2	sysin	$E136

Tabelle 1: Maschinencode, den der Compiler für jeden Befehl erzeugt.

4. FOR V=Ausdr TO Ausdr [STEP Ausdr]
Auch FOR..NEXT-Schleifen können nicht geschachtelt werden.
5. NEXT
6. POKE Ausdr,Ausdr
7. SYS Ausdr
8. GOTO N
9. GOSUB N
10. RETURN
11. END oder STOP
12. REM [Bemerkungen]

Eine Zeile kann mehrere Befehle enthalten, die durch Doppelpunkte voneinander getrennt werden. Hier einige Beispiele für korrekte Befehle:

```
R=PEEK(A)*100/M
IF Y*40+X>2023 THEN PRINT CHR$(147);
FOR I=1 TO X+A:PRINT I+64:NEXT
SYS B+1024:RETURN
GOSUB 500:PRINT "TOTAL";T
GOTO 20
POKE A-I,J AND 15:END
```

Ungültig sind hingegen:

```
R=COS(B)
PRINT TI$
GET X$(I)
OPEN 15,8,15
```

Aufpassen muß man bei negativen Zahlen: Sie werden im Zweier-Komplement verarbeitet. So wird —1 intern als 2↑16—1=65535 abgelegt. Wertzuweisungen wie A=—7 sind erlaubt, solche wie A=—B jedoch nicht. Berechnungen mit negativen Zahlen werden korrekt durchgeführt. Der Print-Befehl gibt allerdings keine negativen Zahlen aus. So wird beispielsweise bei

```
A=—1:PRINT A
```

als Ergebnis 65535 gemeldet. Es empfiehlt sich daher, negative Zahlen nur bei Wertzuweisungen und als Ergebnis von Zwischenberechnungen zu verwenden.

Wird bei arithmetischen Operationen die höchste Zahl überschritten, die sich in zwei Bytes darstellen läßt, so erfolgt keine Fehlermeldung. Es liegt also in der Verantwortung des Programmierers, auf Bereichsüberschreitungen zu achten.

Mikro-Basic unterstützt keine Arrays. Integer-Arrays kann man aber simulieren, indem man zusammenhängende Speicherbereiche verwendet. So läßt sich beispielsweise

```
FOR I=1 TO 5:A(I)=I:NEXT
```

ersetzen durch

```
FOR I=1 TO 5:POKE 828+I,I:NEXT
```

Ebenso können die fehlenden Strings mit Speicherbereichen simuliert werden, indem man in einem Byte je einen Buchstaben unterbringt.

Anstatt des GET-Befehls kann man PEEK(197) verwenden, wodurch man den Tastaturcode der gedrückten Taste erhält. Mit einigen Zeilen in Mikro-Basic kann er leicht in ASCII umgerechnet werden. Dazu werden die Konvertierungstabellen im ROM benutzt (siehe die Zeile 70 ff. im Beispiellisting 2).

In der Regel wird man es vorziehen, Programmpassagen, die den Sprachumfang von Mikro-Basic sprengen, im Normalbasic zu schreiben und nur zeitkritische Subroutinen des Basicprogramms zu kompilieren. Sie werden im Hauptprogramm dann durch SYS anstatt durch GOSUB aufgerufen. Variablen werden vom Hauptprogramm durch Poken an eine Adresse übergeben, die im kompilierten Programmteil dann mit PEEK ausgelesen wird.

Tabelle 1 zeigt den Maschinencode, den der Compiler für jeden Befehl erzeugt. Kompliziertere Ausdrücke werden in eine Kombination dieser Bausteine übersetzt. In den meisten Fällen enthält der Akkumulator das Low Byte einer Zahl und das X-Register das High Byte. Tauchen im Programm Multiplikationen oder Divisionen auf, so wird dem Code eine Spezialroutine für diese Operationen hinzugefügt.

An den Listings 2 und 3 kann man die Leistungsfähigkeit des Compilers überprüfen. Listing 2 dient hauptsächlich zum Test, ob der Compiler funktioniert. Das Programm löscht zuerst den Bildschirm und meldet sich mit „TEST-.COMP". Dann setzt es den Cursor in die zehnte Zeile und schreibt „TEST". Als nächstes werden die Zahlen von 1 bis 5 angezeigt. Dann wird die Tastatur ausgelesen und das eingegebene Zeichen auf den Bildschirm gebracht. Anschließend wird angezeigt, ob das Zeichen gleich, größer oder kleiner als der Buchstabe A ist.

Das Programm in Listing 3 demonstriert den Geschwindigkeitsunterschied zwischen kompilierten und unkompilierten Programmen. Es füllt den Bildschirm mit verschiedenen Farbmustern. In interpretiertem Basic dauert das eine Minute. Bei der kompilierten Version ist es schwierig, genaue Zeitangaben zu machen, dafür geht alles zu schnell: Der ganze Bildschirm ist im Bruchteil einer Sekunde gefüllt!

Beim Abtippen des Programms ist zu beachten: Die Zeilen 1590 und 2200 sind länger als 80 Zeichen. Basicbefehle müssen bei ihnen also abgekürzt werden, damit die Zeilen in eine Doppelzeile passen.

(Victor H. Cortes)

```
 10 PRINT "[DOWN,CBM G]MICRO COMPILER"   1984
 15 REM VON VIC CORTES
 20 GOSUB 1780:GOTO 590                  1131
 30 REM .. ERSTE VARIABLE
 40 GOSUB 400:POKE A,169:POKE A+1,L:K=2  2781
 50 IF V THEN POKE A,173:POKE A+2,H:K=3  2162
 60 A=A+K:POKE A,174:POKE A+1,C:POKE A+  2892
    2,H
 70 IF V=0 THEN POKE A,162:POKE A+1,H    2345
 80 A=A+K:RETURN                          670
 90 REM .. AUSDRUCK
100 P=0:IF PEEK (U)=194 THEN U=U+2:P=1   2804
110 GOSUB 40                              437
120 IF U>912 THEN PRINT "[RVS]OVERFLOW"  3778
    :ER=ER+1:RETURN
130 O=0:B=PEEK (U):IF B=173 THEN 280     2269
140 IF B=172 THEN 280                    1205
150 IF B=170 THEN O=109:POKE A,24:A=A+1  3108
160 IF B=171 THEN O=237:POKE A,56:A=A+1  3145
170 IF B=175 THEN O=45                   1785
180 IF B=176 THEN O=13                   1803
190 IF O=0 THEN 320                      1016
200 U=U+1:GOSUB 400:POKE A,O-4:POKE A+   2796
    1,L:K=2
210 IF V THEN POKE A,O:POKE A+2,H:K=3    2436
220 A=A+K:POKE A,168:POKE A+1,138:A=A+2  3224
```