So macht man Basic-Programme schneller

Teil 2

Diesmal wollen wir den Basic-Programmen, was die Geschwindigkeit anbelangt, mit einem Ausflug in die Assemblerprogrammierung, auf die Sprünge helfen.

Hier soll Ihnen kein Maschinensprache-Kurs zugemutet werden. Doch ein Programm in Maschinensprache besteht genauso aus Befehlen, Adressen und Variablen, wie ein Basic-Programm, nur sind sie in einem speziellen Zahlencode geschrieben. Dieser Zahlencode muß in den Arbeitsspeicher geladen werden. Die für uns einfachste Möglichkeit besteht darin, die Zahlen in den Speicher hineinzuPOKEn. Damit wir aber nicht unmäßig viele POKE-Befehle schreiben müssen, legen wir alle Code-Zahlen hinter DATA-Befehle und holen sie dann mit READ in eine einzige POKE-Schleife. Ich sage das deswegen, weil dieses Einlesen natürlich nicht zu dem Testprogramm gehören darf, dessen Laufzeit wir messen wollen. Das Testprogramm selbst sitzt zwischen den drei Zeilen der »Stoppuhr«. Das heißt, genauer gesagt sitzt das Programm in den Speicherzellen, in die wir es hineinPOKEn. Aber zwischen der Stoppuhr rufen wir es auf, der dem RUN entsprechende Befehl bei Maschinensprache heißt SYS.

Wie Sie gleich noch sehen werden fängt unser Testprogramm ab Speicherzelle 7168 an. Das Ganze sieht dann so aus:

```
10 TI$="000000"
20 PRINT CHR$ (147)
30 SYS 7168
1000 POKE 214,18:PRINT:PRINT
TI/60 "SEKUNDEN":END
```

Ab Zeile 2000 setzen wir jetzt das Programm, welches uns das Maschinenprogramm einliest. Um mit dem Einlesen zu beginnen, setzen wir noch eine Umleitung vor das Meßprogramm:

```
5 GOTO 2000
```

In Zeile 2000 löschen wir den Bildschirm. Zeile 2010 und 2020 und 2030 liest die Codezahlen, die von Zeile 2050 bis 2090 stehen, und POKEt sie in die Speicherplätze 7168 bis 7200.

Die Codezahlen sind für beide Computer fast identisch, nur die Adressen sind verschieden. Deshalb sind Zeile 2060 und 2080 beim C 64 anders als beim VC 20.

Sobald die Zahlen eingelesen sind, können Sie das Meßprogramm mit dem Befehl GOTO 10 (direkt eingetippt) starten.

Im Abdruck unten wird das etwas eleganter gemacht. Zuerst meldet das Programm das Ende des Einlesens (Zeile 2100 und 2101). Dann kommt die Anweisung, wie das Meßprogramm zu starten ist, nämlich durch Drücken irgendeiner Taste, die durch eine GET-Schleife abgefragt wird. Wenn eine Taste gedrückt wird, springt das Programm auf Zeile 10 (das geschieht in Zeile 2160).

```
5 GOTO 2000
10 TI$="000000"
20 PRINT CHR$(147)
30 SYS 7168
1000 POKE 214,18:PRINT:PRINT
TI/60 "SEKUNDEN":END
2000 PRINT CHR$(147)
2010 FOR A=7168 TO 7200
2020 READ B
2030 POKE A,B
2040 NEXT
2050 DATA 162,0,169,1,157
2060 DATA 0,30,169,6,157,0,150
(2060 DATA 0,4,169,14,157,0,216)
2070 DATA 232,224,0,208,241,169,1,
157
2080 DATA 254,30,169,6,157,254,150
(2080 DATA 254,4,169,14,157,254,216)
2090 DATA 232,224,120,208,241,96
2100 PRINT"DAS MASCHINEN-
PROGRAMM"
2110 PRINT"IST JETZT EINGELE-
SEN.":PRINT
2120 PRINT"ZUM STARTEN DES
PRO—"
2130 PRINT"GRAMMS "CHR$(18)
"TASTE"CHR$(146)"DRUECKEN"
2140 GET A$
2150 IF A$="" "THEN 2140
2160 GOTO 10
```

So, inzwischen haben Sie sicher Ihre Überraschung gehabt! 0,066 Sekunden Laufzeit beim C 64 und 0,033 beim VC 20.

Ich hoffe, daß ich Sie mit dem Virus der Maschinensprache infiziert habe.

Wir wollen im Folgenden ein paar arithmetische Funktionen untersuchen und beschleunigen. Als erste nehmen wir uns in **Version 17** die Multiplikation vor. Die Messung der Laufzeit erfolgt auf dieselbe Weise wie bei allen Programmen vorher auch. Deshalb bleiben die Zeilen 10, 20 und 1000 gleich. Die Multiplikation selbst soll 300 mal ausgeführt werden (Zeile 30). Dann wird das Ergebnis gedruckt (Zeile 60).

```
30 FOR Z=1 TO 300
50 NEXT
60 PRINT A
```

Als Multiplikation nehmen wir den Extremfall einer kurzen Zahl multipliziert mit einer langen.

```
40 A=3*0,123456789
```

Nach RUN bleibt der Bildschirm zuerst leer, bis dann nach 11,85 (14, 15) Sekunden das Ergebnis der Multiplikation und die Laufzeit ausgedruckt wird.

In **Version 18** vertauschen wir die beiden Zahlen, die in Zeile 40 multipliziert werden.

```
40 A=0.123456789*3
```

Diese einfache Manipulation bringt natürlich nach Adam Riese dasselbe Resultat wie vorher, aber die Laufzeit ist kürzer. Wir gewinnen beim VC 20 0,37 Sekunden, beim C 64 0,44 Sekunden. Dieser Gewinn ist nicht überwältigend, aber überraschend. Aber denken Sie nach!

Wie ist das, wenn Sie so eine Multiplikation auf dem Papier durchführen? Da ist die Rechnung im zweiten Fall auch einfacher. Der Computer hat genau dasselbe Problem.

In **Version 19** nützen wir noch eine kleine Eigenheit der Commodore-Computer aus, die auf ihre amerikanische Herkunft zurückzuführen ist. Bei den Angelsachsen ist es nämlich erlaubt, eine Null vor dem Dezimalpunkt wegzulassen. Beim Computer dürfen wir das auch. Obwohl das mit der Multiplikatikon direkt nichts zu tun hat, bietet sie uns doch eine gute Gelegenheit, die Einsparung durch das Weglassen der Null auch zeitlich zu messen. Also, Zeile 40 sieht jetzt so aus:

```
40 A=.123456789*3
```

Das bringt nicht sehr viel, 0,17 Sekunden beim VC 20, 0,20 Sekunden beim C 64. Aber Kleinvieh macht auch Mist.

Eine ähnliche Verbesserung, die wir hier nicht ausprobieren, wird erzielt durch den Ersatz einer alleinstehenden Null durch einen Punkt, zum Beispiel:

```
statt IF X=0 THEN —
jetzt IF x=. THEN —
```

Eine gewaltige Beschleunigung erfährt das Multiplikationsbeispiel, wenn wir die Regel 1 anwenden und die Variablen vordefinieren.

Regel 7
★Bei einer Multiplikation soll die längere Zahl vor der kürzeren stehen (langer Multiplikant, kurzer Multiplikator).
★Eine einzelne Null wird durch einen Punkt ersetzt, eine Null vor dem Dezimalpunkt wird weggelassen.

In **Version 20** ersetzen wir in Zeile 40 beide Zahlen durch Buchstaben, die wir in einer neuen Zeile 25 diese Werte zuweisen.

```
25 B=.123456789:C=3
40 A=B*C
```

Dieser Lauf bleibt nach 1,23 (1,48) Sekunden stehen, das heißt wir gewinnen 10,25 (12,23) Sekunden (von 11,48!). Also bitte Regel 1 unbedingt beachten!

Eine andere betrachtenswerte arithmetische Funktion ist das »Potenzieren« (Quadrat-/Kubikzahlen), ausgelöst durch das Zeichen ↑. **Version 21** erzielen wir durch Löschen der Zeile 25 und Abänderung der Zeile 40:

```
40 A= 4 ↑ 3
```

»Vier hoch drei« ergibt 64 und braucht 8,81 (10,53) Sekunden.

In **Version 22** wollen wir sehen, ob vordefinierte Variable auch so einschlagen, wie bei der Multiplikation.

```
25 B=4:C=3
40 A=B↑C
```

Man kann sich doch auf nichts verlassen! Diesmal sind wir nur um 0,18 (0,22) Sekunden schneller. Wir dürfen aber nicht aufgeben. **Version 23** macht alles wieder wett und zwar durch den simplen Trick, daß wir das Potenzieren in seine Grundelemente zerlegen.

Sie wissen doch: 4 hoch 3 (4↑3) ist dasselbe wie »4 zweimal mit sich selbst multipliziert« (4*4*4).

```
25 B=4 (C entfällt)
40 A=B*B*B
```

Ja, da schauen Sie, gell? Beim VC 20 braucht das Programm nur 1,68, also fast 7 Sekunden weniger. Beim C 64 sind es 8,31 Sekunden, die wir sparen.

Regel 8
Die Funktion Potenzieren (↑) soll durch Mehrfach-Multiplikation ersetzt werden.

Als letztes Objekt möchte ich oft aufgerufene Unterprogramme messen. Wir erreichen das ganz einfach dadurch, daß wir das letzte Programm (Version 23) abändern. So erhalten wir **Version 24:** Die Definition der Variablen (Zeile 25) und die Multiplikation (Zeile 40) verbannen wir als Unterprogramm an das Ende des Programms und springen innerhalb der 300fachen Schleife mit GOTO darauf.

Version Nr.	Programmier-Methode	Laufzeit (Sek.) VC 20	C 64
17	Multiplikation, lang x kurz	11,85	14,15
18	Multiplikation, kurz x lang	11,48	13,71
19	Null weglassen	11,31	13,51
20	Variable vordefinieren	1,23	1,48
21	Potenzieren (4 hoch 3) mit ↑	8,81	10,53
22	Variable vordefinieren	8,63	10,31
23	4 x 4 x 4 statt 4↑3	1,68	2,01

Laufzeiten der Programmversionen für arithmetische Funktionen

24	Unterprogramm am Ende, Sprung mit GOTO-GOTO	2,75	3,28
25	Unterprogramm am Ende, Sprung mit GOSUB-RETURN	2,61	3,13
26	Unterprogramm am Anfang, Sprung mit GOTO-GOTO	2,60	3,10
27	Unterprogramm am Anfang, Sprung mit GOSUB-RETURN	2,48	2,96

Laufzeiten der Programmversionen für Unterprogramme

```
25 löschen; 30 FOR Z=1 TO 300; 40
GOTO 40000
```

Alles andere bleibt, aber neu kommt dazu: 40000 B=4; 50000 A=B*B*B; 60000 GOTO 50

Es ist nicht weiter erstaunlich, daß dieser Umbau diese Version 24 gegenüber Version 23 verlangsamt. Aber merken Sie sich die Laufzeit, VC 20: 2,75 Sekunden, C 64: 3,28 Sekunden. Als nächstes ersetzen wir die beiden GOTO-Zeilen durch GOSUB-RETURN.

```
40 GOSUB 40000
60000 RETURN
```

Diese **Version 25** spart uns 0,14 (0,15) Sekunden. GOSUB ist schneller als GOTO!

Sie haben vielleicht schon gelesen, daß oft gebrauchte Unterprogramme am Anfang eines Programms stehen sollen. Den Grund dafür will ich Ihnen mit den nächsten zwei Versionen vorführen.

Version 26 macht das zunächst für die GOTO-Version. Wir bauen sie auf der Version 24 auf, mit folgenden Änderungen:

Die Zeitmessung lassen wir wie gehabt in den Zeilen 10, 20 und 1000, die Schleife und den Ausdruck des Resultats in den Zeilen 30, 50 und 60.

Nur beim Unterprogramm streichen wir alle Nullen der Zeilennummern, so daß es jetzt in den Zeilen 4, 5 und 6 steht. Um zu vermeiden, daß das Programm gleich mit dem Unterprogramm beginnt, fügen wir davor (Zeile 3) noch eine Umleitung ein, die sofort auf der Zeile 10 weitermacht. Schließlich brauchen wir noch den Sprung in das Unterprogramm, den wir in die Zeile 33 setzten. Das Ganze sieht jetzt so aus:

```
  3 GOTO10
  4 B=4
  5 A=B*B*B*
  6 GOTO 50
10 TI$="000000"
20 PRINT CHR$(147)
30 FOR Z=1 TO 300
33 GOTO 4
50 NEXT
60 PRINT A
1000 POKE 214,18:PRINT:PRINT
TI/60 "SEKUNDEN":END
```

Nach RUN erhalten wir beim VC 20 2,6 Sekunden, beim C 64 3,1 Sekunden. Gegenüber Version 24, unserem Vergleichsobjekt, sparen wir 0,15 (0,18) Sekunden.

Dasselbe passiert, wenn wir in der **Version 27** die GOTOs mit GOSUB-RETURN ersetzen.

```
  6 RETURN
33 GOSUB 4
```

Gegenüber der anderen GOSUB-Version (Version 25) sparen wir beim VC 20 0,13 Sekunden, beim C 64 0,17 Sekunden.

Regel 9
★ Der Aufruf von Unterprogrammen mit GOSUB ist schneller als mit GOTO.
★Häufig gebrauchte Unterprogramme gehören ganz an den Anfang eines Programms. Sie müssen dann allerdings zuerst mit einem GOTO umgangen werden.

Ich bin überzeugt, daß in Basic noch mehr spektakuläre Zeitgewinne stecken.

Falls Sie eine REGEL 10 oder noch mehr entdecken, ermuntere ich Sie um Mitteilung.

Wenn Sie Fragen haben, können Sie mich mit einer Leserzuschrift ansprechen.

(Dr. Helmuth Hauck/aa)