SERIE

Interne Variablen

Teil 3 des Action!-Centers führt in die Verwendung der compilerspezifischen Variablen ein.

Wer schon einmal mit Action! gearbeitet hat, wird sich nur schwer der Faszination dieser Sprache entziehen können. Action! ist angenehm, in der Bedienung, schnell im Kompilieren und erzeugt nicht zuletzt Programme, die mit reinen Assembler-Programmen in Laufzeit und Länge konkurrieren können.

8 Bit

In der heutigen Folge werden Sie sehen, wie die Möglichkeiten des Compilers noch besser zu nutzen sind. Ebenso wie Basic verfügt Action! über eine Reihe von internen Variablen, mit denen man z.B. die Code-Erzeugung hervorragend steuern kann. Es ist sogar möglich, Programme so zu schreiben, daß sich Speicherbereiche nutzen lassen, die während der Kompilierung nicht zur Verfügung stehen.

Was in Basic der POKE-Befehl, ist in Action! die SET-Direktive. Mit ihr kann man gezielt einzelne Speicherzellen oder 2-Byte-Werte in den Speicher eintragen. Da dies gelegentlich zu Verwirrungen führen kann, wollen wir zunächst zwei Beispiele betrachten:

SET $491=$80

trägt nur den Wert $80 in die Speicherzelle $491 ein. Dagegen beschreibt

SET $491=$5140

Adresse $491 mit $40 und Adresse $492 mit $51. Beachten Sie bitte, daß im ersten Beispiel die Speicherzelle $492 nicht verändert, also auch nicht mit Null beschrieben wurde! Man sollte sich demnach immer bewußt sein, ob man Bytes oder Cards eintragen will. Nur wenige wissen wahrscheinlich, daß die SET-Direktive auch mit dem Pointer-Symbol angewendet werden kann:

SET $491=$5000^ schreibt den Inhalt der Speicherzelle 5000 nach $491. Falls der Inhalt von $5001 ungleich Null ist, so wird dieser nach $492 transferiert. Auch hier sollte man ein bißchen vorsichtig sein.

Interne Variablen

Die SET-Direktive wird während des Kompilierens und nicht während der Laufzeit des erzeugten Programms ausgeführt. Dies ist von einiger Bedeutung. Sehen wir uns nun die wichtigsten internen Variablen des Action!-Compilers an, die zur Steuerung der Code-Erzeugung eingesetzt werden:

APPMHI (Adresse $0E.2 Bytes)

Dieses seltsame Kürzel, das für Application Memory High steht, ist eigentlich mehr eine Variable des Atari-Betriebssystems, die bei Action! aber sehr häufig verwendet wird. Hier findet sich immer ein Zeiger auf die nächste freie Speicherzelle, die Action! nutzen kann. Das hat nun mehrere Konsequenzen.

Wenn Sie APPMHI mit ?$0E im Monitor nach einem Kompiliervorgang abfragen, so wird die Endadresse des kompilierten Programms angezeigt. Fragen Sie diese Speicherzelle vor dem Kompilieren ab, erhalten Sie die Anfangsadresse, die im normalen Betrieb mit der Endadresse des Textspeichers übereinstimmt. Auf diese Weise läßt sich z.B die Länge der erzeugten Programme herausfinden. Wenn Sie APPMHI mit SET verändern, wird der Objectcode ab der SET-Direktive an die gewünschte Speicherstelle eingetragen. Im Prinzip ist SET $0E=<Adresse> nichts anderes als der .ORG- oder *=-Befehl eines Assemblers.

Poke-Befehl für Action!

CODEBASE (491,2)

Wenn Sie das Handbuch des Compilers studiert haben, dann wissen Sie, daß $0E und $491 immer gemeinsam auf die gleiche Adresse geändert werden sollten. Das muß allerdings nicht in jedem Fall sinnvoll sein, denn CODEBASE hat nur die Funktion, sich die Anfangsadresse des kompilierten Programms für eine eventuelle Aufzeichnung mit dem Monitorbefehl W auf Disk zu merken. Wenn Sie CODEBASE mit SET verändern, können Sie somit auch größere oder kleinere Segmente auf Disk speichern.

CODESIZE ($493,2)

Hier merkt sich Action!, wie viele Bytes erzeugt wurden, damit der entsprechende Speicherblock auf Disk geschrieben werden kann. Wenn Sie also diese Variable modifizieren, so empfiehlt es sich auch, sie auf den neuesten Stand zu bringen.

STSP ($495,1)

Diese 1-Byte-Variable hat zwar nur mittelbaren Einfluß auf die Code-Erzeugung, ist aber in vielen Fällen sehr wichtig. Sie gibt an, wie viele Pages für die Symboltabelle verwendet werden sollen. Die Voreinstellung dafür ist 8 (d.h. 2 KByte); bei langen Programmen mit vielen INCLUDES ist diese Grenze schnell erreicht.

Wenn der Compiler einen Error 61 (Out of Symbol Space) meldet, dann wird es Zeit, die Symboltabelle auf z.B. 12 Pages zu vergrößern. Der Befehl dazu lautet SET $495=12. Diese Anweisung ist direkt im Monitor zu geben, da die Symboltabelle vor dem Kompiliervorgang eingerichtet wird, so daß ein SET-Befehl im Programm keine Wirkung mehr hat. Hartgesottene Naturen können das natürlich trotzdem tun, müssen aber dann einen Error 61 bei der ersten Kompilierung in Kauf nehmen.

CODEOFF ($B5,2)

Mit Hilfe dieser Variablen können Sie Programme für Speicherbereiche schreiben, die während der Kompilierung belegt sind. Dies ist eine wirklich trickreiche und leistungsfähige Einrichtung des Action!-Compilers. Man trägt hierzu einen Offset in die Speicherzellen $B5 und $B6 ein, also einen Wert für den Unterschied zwischen der Adresse, bei der das Programm während des Kompilierens im Speicher abgelegt wird, und der Adresse, ab der es später laufen soll.

Dazu gleich ein Beispiel. Ein Programm für den Speicherbereich ab $A000 soll geschrieben werden. Während des Kompiliervorgangs ist dieser Bereich durch die Action!-Cartridge belegt. Mit den bereits besprochenen SET-Anweisungen legt man das Programm zunächst an die Adresse $8000 und setzt CODEOFF auf $2000. Es wird nun so kompiliert, daß es an der Adresse $A000 (= $8000 + $2000) lauffähig ist.

Wie das in der Praxis aussieht, zeigt Ihnen Listing 1.Wenn Sie dieses Programm mit W auf Diskette schreiben, wird es unter DOS übrigens genau an die Stelle geladen, an der es lauffähig ist. Bei solchen Programmen darf man natürlich nicht auf die ROM-Bibliothek des Action!-Moduls zurückgreifen (kein PRINT, POKE usw., keine Funktionen mit mehr als 3 Byte Parameter), oder es ist das Runtime-Modul zu verwenden.

Damit wollen wir das Action!-Center für heute beschließen. Ich hoffe, Sie haben etwas Interessantes gefunden, und würde mich freuen, wenn Sie das nächste Mal wieder dabei sind.

Peter Finzel

Listing in Action!

```
;************************************
;Beispiel zur Steuerung der Code-
;erzeugung bei ACTION!
;
;Programm compilieren und im Monitor
;abspeichern, dann Steckmodul ent-
;fernen und mit DOS laden. Das Pgm
;wird dann an die Adresse $A000 ge-
;laden (wo sich vorher das ACTION!-
;Steckmodul befand).
;************************************

BYTE WSYNC =$D40A,
     VCOUNT=$D40B,
     COLBK =$D01A,
     RTCLK =$14,
     SDMCTL=$22F

;
; Programm ab $8000 ablegen
;
SET $0E =$8000
SET $491=$8000
;
; Offset zur Adresse $A000
;
SET $B5 =$2000
;
PROC TEST()

SDMCTL=0
  DO
  WSYNC=0
  COLBK=RTCLK-VCOUNT
  OD
RETURN
```

Dieses Listing demonstriert, wie Speicherbereiche genutzt werden können, die bei der Compilierung gar nicht zur Verfügung stehen.