Einführung in CORE WAR Teil 2

Einige Gladiatoren (Fortsetzung)

Gnom - der Bombenleger

Schon etwas größer als *Knirps* – aber immer noch klein – ist der Kämpfer *Gnom* (*Dwarf* im englischen Original). Er wirft in kurzen, konstanten Abständen Bomben in die Arena in der Hoffnung, den Widersacher zu treffen. Der Abstand zwischen zwei Bombenzielen in der Arena wird dabei so gewählt, dass sich Gnom nicht selbst abschießt, wenn die Arena einmal umrundet ist.

```
; Redcode-Programm  "Gnom"

      MOV    3   @3      ; Die DAT-Bombe wird in die Speicher
                         ; zelle mit der Adresse geworfen, die der
                         ; Operand B der DAT-Anweisung angibt

      ADD   #8    2      ; Es wird die Schussweite 8 zu dem
                         ; Operanden B der DAT-Aweisung addiert.

      JMP   -2           ; Es wird zur MOV-Anweisung gesprungen.

      DAT   #0  #-4      ; Der Operand B enthaelt das aktuelle
                         ; Bombenziel; weiterhin ist die DAT-
                         ; Anweisung die Bombe selbst.

      END
```

Hier tauchen gleich drei neue Anweisungen auf. Die **DAT**-Anweisung ist besonders interessant: Man kann sie nicht ausführen, und den Versuch der Ausführung muss ein Kämpfer mit dem Leben bezahlen; sie ist also eine Bombe (genauer gesagt eine Tretmine), die bei der leichtesten Berührung explodiert. Andererseits erfüllt die **DAT**-Anweisung noch einen weiteren Zweck: Man kann in ihren Operanden Daten ablegen – insbesondere kann so eine Bombe die relative Adresse der Speicherzelle enthalten, in der sie einschlagen soll.

Speicher-adresse	Anzahl ausgeführter Anweisungen		
	0	1	2
n - 1		DAT #0 #-4	DAT #0 #-4
n	-> MOV 3 @3	MOV 3 @3	MOV 3 @3
n + 1	ADD #8 2	-> ADD #8 2	ADD #8 2
n + 2	JMP -2	JMP -2	-> JMP -2
n + 3	DAT #0 #-4	DAT #0 #-4	DAT #0 #4
n + 4			

Die Programmausführung beginnt wieder mit der ersten Anweisung. Wie Sie die Adresse erhalten, die der Operand A der **MOV**-Anweisung angibt, ist Ihnen bereits von *Knirps* her bekannt. Sie zählen drei Speicherzellen nach unten ab und erhalten so direkt die Speicherzelle n + 3; deshalb wird diese Art der Adressierung als *direkt* bezeichnet. Der zweite Operand ist am Anfang mit dem Klammeraffen (**@**) versehen; dies bedeutet, dass er *indirekt* adressiert ist. Was heißt das? Stellen Sie sich vor, Sie haben auf einer Party jemanden kennen gelernt und vergessen, sich die Telefonnummer geben zu lassen. Sie kennen aber eine Person, die die Adresse der Person Ihres Interesses hat; so kommen Sie zwar nicht direkt, aber *indirekt* zum Ziel. Sehr ähnlich verhält sich nun die Sache mit der *indirekten* Adressierung: Es wird der Operand B der Speicherzelle betrachtet, die man erhält, wenn man den Operanden als *direkt* adressiert annimmt – dieser Operand B ist in dem Beispiel -4. Der Wert dieses Operanden wird dann als Adresse relativ zu der Speicherzelle genommen, aus der der Wert entnommen wurde. Es ergibt sich also in dem Beispiel, dass der Operand B der **MOV**-Anweisung die Speicherzelle n-l bezeichnet. Die **MOV**-Anweisung schleudert also die **DAT**-Bornbe in die Speicherzelle n-1.

Wie bei der **MOV**-Anweisung gibt der Operand A der **ADD**-Anweisung die Quelle und der zweite das Ziel an. Der Operand A ist *unmittelbar,* was durch das Nummernzeichen (**#**) kenntlich gemacht ist: Es wird durch einen *unmittelbar* adressierten Operanden keine Adresse einer Speicherzelle

angegeben; er hat somit lediglich einen Wert. Dieser ist in dem Beispiel 8. Der Operand B weist auf die Speicherzelle n + 3, in der die **DAT**-Anweisung steht. Es wird 8 zu der -4 im Operanden B der **DAT**-Anweisung addiert, der ja, wie Sie gesehen haben, die Zieladresse für die Bombe angibt.

Nach der **ADD**-Aweisung wird die Sprunganweisung **JMP** ausgeführt. Der Operand gibt das Sprungziel an; es wird also anschließend mit der **MOV**-Anweisung am Programmanfang fortgefahren.

Speicher-adresse	Anzahl ausgeführter Anweisungen		
	3	4	5
n - 1	`DAT #0 #-4`	`DAT #0 #-4`	`DAT #0 #-4`
n	`-> MOV 3 @3`	`MOV 3 @3`	`MOV 3 ©3`
n + 1	`ADD #8 2`	`-> ADD #8 2`	`ADD #8 2`
n + 2	`JMP -2`	`JMP -2`	`-> JMP -2`
n + 3	`DAT #0 #4`	`DAT #0 #4`	`DAT #0 #12`
...			
n + 7		`DAT #0 #4`	`DAT #0 #4`

Die **MOV**-Anweisung wirft nun die Bombe, und zwar acht Speicherzellen weiter als zuvor in die Speicherzelle *n + 7*. Danach wird das Bombenziel von der **ADD**-Anweisung wieder neu eingestellt. Das Spielchen setzt *Gnom* dann solange fort, bis entweder sein Gegner oder er selbst darniederliegt oder MARS den Kampf für unentschieden erklärt. Wieviel Zeit verstreichen muß, bis MARS ungeduldig wird, lässt sich bei den Optionen im Programm einstellen.

Eine interessante Beobachtung können Sie machen, wenn Sie *Gnom* einmal unter der Speichergröße 8192 und ein weiteres Mal unter 7999 laufen lassen.

Quelle: Aus dem Begleitheft zu *Core War* – Krieg der Kerne, Spektrum der Wissenschaft 1993, S. 26 ff.