Funktionsweise der do-loop-Schleife

Wir gehen aus von folgendem Beispiel:

```
10 3 do Bef1 Bef2 loop Bef3
```

Dem entsprechen im Speicher die Unterprogrammaufrufe

```
rcall <10>
rcall <3>
rcall <do>
rcall <Bef1>
rcall <Bef2>
rcall <loop>
rcall <Bef3>.
```

Dabei bedeuten die spitzen Klammern “Adresse von...”.

Durch die ersten beiden Unterprogramme werden der Startwert (3) und der Endwert (10) des Schleifenindex auf den Arbeitsstack gelegt. Jedesmal, wenn ein Unterprogramm aufgerufen wird, merkt sich der Mikrocontroller die Adresse des nächsten Befehls, indem er sie auf den Returnstack legt. Wenn also das `do`-Unterprogramm aufgerufen wird, wird die Adresse von `rcall <Bef1>` auf den Returnstack gelegt. Genauer betrachtet besteht diese Adresse aus zwei Bytes; wir bezeichnen diese Adressen als *AdrBef1(high)* und *AdrBef1(low)*. Unmittelbar nach dem Aufruf des do-Unterprogramms sehen unsere beiden Stacks also so aus:

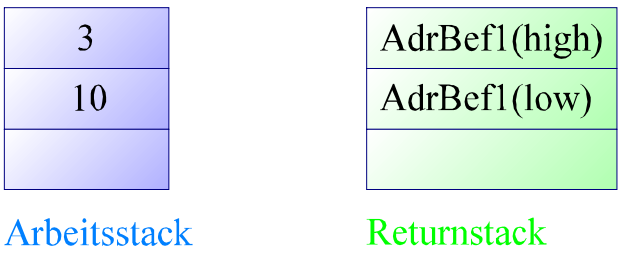

Abbildung 1

Alles, was das `do`-Unterprogramm leisten muss, ist dafür zu sorgen, dass der Mikrocontroller sich dieses Adresspaar langfristig merkt; nur so kann gewährleistet werden, dass er am Ende eines Schleifendurchlaufs wieder zum Anfang der Schleife kehren kann. Dieses Merken kann nicht über Arbeitsregister wie z. B. r16 geschehen; denn diese könnten z. B. durch das Unterprogramm von `Bef1` oder `Bef2` überschrieben werden. Ein sicherer Ort zum langfristigen Merken ist der Returnstack. Hier liegt unser Adresspaar zwar schon, aber am Ende des `do`-Unterprogramms wird dieses Adresspaar durch den ret-Befehl vom Returnstack in den Programmzähler geschoben und verschwindet dabei vom Returnstack.

Damit ist aber auch schon die Lösung in Sicht: Das Adresspaar auf dem Returnstack muss von `do` verdoppelt werden; so steht die Kopie auch nach der Ausführung von `do` noch zur Verfügung. Ebenso müssen auch die beiden Schleifenindizes gesichert werden. Dies leistet der folgende Assemblercode:

```
.def AdrH = r16
.def AdrL = r17

.def A = r18
.def E = r19

ld  A,-x        ; Startindex vom Parameterstack
ld  E,-x        ; Schleifenende vom Parameterstack

pop AdrH        ; Returnadresse vom Returnstack
pop AdrL

push AdrL       ; und wieder drauf (für LOOP)
push AdrH

push E          ; Schleifenparameter auf Returnstack
push A

push AdrL       ; Returnadresse noch einmal auf den Stack
push AdrH

ret             ; zum nächsten Befehl
```

Unmittelbar vor dem `ret`-Befehl des `do`-Unterprogramms und unmittelbar danach sieht der Returnstack dann so aus:

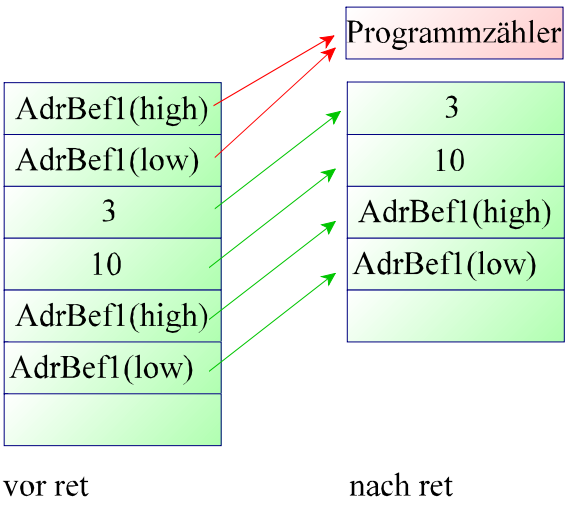

Abbildung 2

Durch den nun folgenden Unterprogrammaufruf `rcall <Bef1>` wird nun das Adresspaar von `rcall <Bef2>` auf den Returnstack gelegt; beim Rücksprung verschwindet es aber wieder. Und so geht es weiter, bis das folgende `loop`-Unterprogramm aufgerufen wird.

```
.def AdrH = r16     ; Returnstack
.def AdrL = r17

.def A = r18        ; Schleife: aktueller Index
.def E = r19        ; Endwert des Schleifenindex

.def nextAdrH = r20 ; Adresse von Bef3
.def nextAdrL = r21

pop nextAdrH        ; Adresse von Bef3 retten
pop nextAdrL

pop  A          ; aktueller Index vom Returnstack holen
pop  E          ; Endwert vom Returnstack

pop ZH          ; Adresse von Bef1 vom Returnstack
pop ZL

cp A, E
breq loopende  ; wenn A=E dann nach loopende springen

; sonst (A < E)
push ZL         ; und wieder drauf (für nächstes LOOP)
push ZH
inc A
push E          ; Schleifenparameter auf Returnstack
push A
ijmp            ; Sprung nach Adresse, die in Z steht (s. o.)
                ; keine zusätzliche Adr auf Stapel!
loopende:
push nextAdrL
push nextAdrH
ret             ; zum nächsten Befehl
```

In dieser Situation liegt das Adresspaar von `rcall <Bef3>` auf dem Returnstack. Das wird zunächst einmal vom Returnstack geholt und in Arbeitsregistern (nextAdrH und nextAdrL) zwischengespeichert; die Schleifenindizes 3 und 10 werden ebenso in Arbeitsregistern zwischengelagert (A bzw. E) . Die Werte *AdrBef1(high)* und *AdrBef1(low)* werden in das Registerpaar Z geschoben; dies wird einen indirekten Sprung zum Befehl `rcall <Bef1>` ermöglichen. Nun wird kontrolliert, ob der aktuelle Schleifenindex (3) kleiner als der Endwert (10) ist. Da dies der Fall ist, wird der Schleifenindex (3) um 1 erhöht, das Adresspaar und die Indizes für einen möglichen weiteren Schleifendurchlauf wieder auf den Returnstack gelegt und der indirekte Sprung `ijmp` ausgeführt. Der bedeutet einen Sprung an die Adresse, welche im Registerpaar Z steht, also zu `rcall <Bef1>`.

Dies wiederholt sich solange, bis der aktuelle Indexwert gleich dem Endwert (10) ist. In diesem Fall wird das zwischengespeicherte Adresspaar von `rcall <Bef3>` wieder auf den Returnstack gelegt; so springt das Programm durch den letzten Befehl von `loop`, nämlich dem `ret`-Befehl, wie gewünscht nicht mehr an den Anfang der Schleife, sondern zum Unterprogrammaufruf von `Bef3`.

Aufgabe

Das Wort `do` kann auch als F-Wort geschrieben werden:

```
: do swap R> R> over over >R >R rot >R rot >R >R >R ;
```

Schlagen Sie die Bedeutung von `R>` und `>R` im Vokabular nach und machen Sie sich die Funktionsweise dieser Befehlsfolge anhand von Stackdiagrammen klar.