Interrupts

MikroForth unterstützt auch das Interrupt-Konzept des Attiny-Mikrocontrollers. Allerdings gibt es im Standardvokabular nur Wörter für die Interrupts INT0, INT1 und T0OVF (Timer/Couter0 Overflow). Einer Erweiterung des Vokabulars für andere Interrupts steht jedoch nichts im Weg.

Am Beispiel des INT0-Interrupts soll nun dargelegt werden, wie die Interrupt-Programmierung in MikroForth erfolgt. Für grundlegende Fragen zum Interrupt-Konzept wird auf das entsprechende Kapitel verwiesen.

Und das soll unser Programm leisten: Ein an Port D.6 angeschlossener Beeper soll die ganze Zeit über tönen. Eine LED an Port B.0 soll währenddessen über eine fallende Flanke an Ta0 geschaltet werden; ob durch diesen Tastendruck diese LED ein- oder ausgeschaltet wird, soll durch den Zustand von Ta1 festgelegt werden.

Wie üblich muss der INT0-Interrupt zunächst initialisiert werden. Dies geschieht durch das Wort `initInt0`:

```
initInt0        ( f – )
```

Der Wert von f entscheidet darüber, ob der Interrupt durch eine fallende (f = 0) oder steigende Flanke (f = 1) am INT0-Eingang (Port D.2) erfolgt. Durch die Initialisierung mit `initInt0` werden auch folgende Aktionen erledigt:

- Port D.2 als Eingang konfigurieren,
- Port D.2 auf high legen,
- Interrupts freigeben.

In unserem Fall muss f = 0 sein, denn durch das Drücken von Ta0 geht D.2 von high nach low über.

Wird der INT0-Interrupt ausgelöst, wird nun automatisch das Wort `int0` ausgeführt. Die Bezeichnung dieses Wortes ist fest vorgegeben. Der Wortkörper kann jedoch beliebig vom Benutzer programmiert werden; jedoch muss dabei folgende Form eingehalten werden:

```
: int0 pushreg <benutzerdefinierter Teil> popreg reti ;
```

Das hat folgenden Grund: Der Interrupt wird in der Regel mitten in der Ausführung eines anderen Wortes erfolgen. Damit die Registerinhalte r16-r29, welche dieses andere Wort womöglich benutzt, nicht verloren gehen, werden sie als erstes durch das Wort `pushreg` gerettet. Dazu kopiert `pushreg` deren Inhalte in die Register r2 bis r15, die bei den A-Wörtern nicht zum Einsatz kommen (dürfen). In dem benutzerdefinierten Bereich von `int0` können jetzt alle Wörter beliebig benutzt werden. Am Ende werden durch `popreg` alle Register r16 bis r29

wieder hergestellt; so kann das in seiner Ausführung unterbrochene Wort korrekt weiterarbeiten.

Beim Auslösen des Interrupts wird der Attiny für weitere Interrupts global gesperrt. Um ihn wieder frei zu geben für erneute Interrupts, muss die Definition von `int0` mit dem Wort `reti` enden.

In unserem Fall sieht die Definition des Wortes `int0` so aus:

```
: int0 pushreg Ta1? . popreg reti ;
```

Der benutzerdefinierte Teil ist sehr kurz: Durch `Ta1?` wird der Zustand des Tasters Ta1 abgefragt. Ist Ta1 gedrückt, wird eine 0 auf den Stapel gelegt und diese dann an Port B ausgegeben; andernfalls wird eine 1 auf den Stapel gelegt und dann ausgegeben. Die LED wird also, je nachdem ob Ta1 gedrückt ist oder nicht, ein- bzw. ausgeschaltet.

Das gesamte Programm sieht dann so aus:

```
: int0 pushreg Ta1? . popreg reti ;
: BeeperAnAus 6 1 outportD 10 waitms 6 0 outPortD 10 waitms ;
: main 6 1 DDBitD 0 initInt0 begin BeeperAnAus 0 until ;
```

Im Hauptprogramm wird durch eine Endlosschleife für eine Schwingung mit der Periodendauer $2 \cdot 10$ ms am Port D.6 gesorgt.

Aufgabe

Testen Sie das Programm aus; betätigen Sie dazu den Taster Ta0 mehrfach bei unterschiedlichen Tasterzuständen von Ta1. Lassen Sie nun den `reti`-Befehl weg und führen Sie den Test erneut durch.