Port-Befehle

MikroForth stellt folgende Port-Befehle zur Verfügung:

Wort	Typ	Kommentar	Stack
.	A	gibt TOS auf Port B aus; (Datenrichtungsbits von Port B werden zuvor alle auf 1 gesetzt.)	(n –)
`blink`	F	`b hp blink` gibt das Bitmuster von b auf Port B aus, wartet hp Millisekunden, gibt 0 auf Port B aus und wartet wieder hp Millisekunden.	(b hp –)
`DDBitB`	A	`bit flag DDBitB` setzt den Anschluss bit des Ports B als Ausgang, wenn flag = 1, sonst als Eingang.	(bit flag –)
`DDBitD`	A	`bit flag DDBitD` setzt den Anschluss bit des Ports D als Ausgang, wenn flag = 1, sonst als Eingang.	(bit flag –)
`DDRB`	A	schreibt b in das Datenrichtungsregister des Ports B.	(b –)
`DDRD`	A	schreibt d in das Datenrichtungsregister des Ports D.	(d –)
`InPortB`	A	`bit InPortB` liest den Eingang bit des Ports B und legt 1/0 auf den Stack, wenn er high/low ist. Vgl. `DDRB` und `DDBitB`	(bit – flag)
`InPortD`	A	`bit InPortD` liest den Eingang bit des Ports D und legt 1/0 auf den Stack, wenn er high/low ist. Vgl. `DDRD` und `DDBitD`	(bit – flag)

Wort	**Typ**	**Kommentar**	**Stack**
`outPortB`	A	`bit flag outPortB` setzt den Ausgang bit des Ports B auf high/low, wenn flag = 1/0 ist. Vgl. `DDRB` und `DDBitB`	(bit flag –)
`outPortD`	A	`bit flag outPortD` setzt den Ausgang bit des Ports D auf high/low, wenn flag = 1/0 ist. Vgl. `DDRD` und `DDBitD`	(bit flag –)
`Ta0?`	F	Legt 1/0 auf Stack, wenn Taster Ta0 offen/geschlossen (D2=1/0) PortD.2 wird automatisch konfiguriert.	(– bit)
`Ta1?`	F	Legt 1/0 auf Stack, wenn Taster Ta1 offen/geschlossen (D3=1/0) PortD.3 wird automatisch konfiguriert.	

Exemplarisch werden wir hier die Wörter ".", `blink`, `Ta0?`, `DDBitD`, `OutPortD` und `InPortD` behandeln. Die restlichen Wörter sind in ihrer Bedeutung ganz ähnlich.

Um etwas interessantere Beispiele verwenden zu können, wollen wir allerdings zuvor eine einfache FORTH-Schleifenkonstruktion vorstellen: die BEGIN-UNTIL-Schleife. Diese sieht folgendermaßen aus:

```
begin Bef1 Bef2 Bef3 ... until
```

Die Befehle Bef1, Bef2, Bef3, ... werden der Reihe nach immer wieder durchlaufen. Allerdings geschieht dies nur solange, wie das Wort `until` auf dem TOS eine 0 vorfindet. Genauer gesagt: Das Wort `until` holt den Wert aus dem TOS und kontrolliert nach, ob er 0 oder 1 ist. Ist er 0 (FALSE), wird die Schleife ein weiteres Mal ausgeführt; ist er 1 (TRUE), so wird die Schleife beendet. Schreibt man also unmittelbar vor das Wort `until` eine 0, so wird eine Endlosschleife gebildet:

```
: endlos Bef1 Bef2 Bef3 ... 0 until ;
```

Kommen wir zu unserem ersten Beispiel: Alle LEDs an Port B sollen im Abstand von 100 ms immer wieder an- und ausgehen. Das Programm dafür ist recht einfach:

```
: main begin 255 . 100 waitms 0 . 100 waitms 0 until ;
```

Schauen wir uns die Definition von `main` Wort für Wort an. `begin` leitet die Endlosschleife ein, welche durch `0 until` begrenzt wird. Innerhalb der Schleife wird zuerst 255 auf den Stack gelegt. Diese Zahl wird durch das Wort "." sogleich vom Stapel genommen und am Port B ausgegeben. Dabei wird durch "." Port B automatisch als Ausgang konfiguriert; d. h. DDRB wird auf &B11111111 gesetzt.

Nun sind also alle LEDs an Port B eingeschaltet. Danach wird die Zahl 100 auf den Stack gelegt, um sogleich von dem Wort "waitms" geholt zu werden: Der Mikrocontroller wartet jetzt 100 ms. Anschließend wird die Zahl 0 auf dem Port B ausgegeben; die LEDs werden somit alle ausgeschaltet. Dann wartet der Mikrocontroller abermals 100 ms. Wir sind am Ende eines Schleifendurchlaufs angekommen. Nun beginnt das Ganze wieder von vorne und so weiter und so weiter... Unsere LEDs an Port B blinken also fortwährend.

In unserem zweiten Beispiel soll eine LED an PortD.6 über den Taster Ta0 aus- und eingeschaltet werden. Genauer gesagt soll die LED aus sein, solange der Taster Ta0 gedrückt ist, und leuchten, solange der Taster nicht betätigt ist. Das FORTH-Programm kann durch folgende Zeilen gebildet werden:

```
: schalten begin Ta0? 6 swap outPortD 0 until ;
: vorbereiten 6 1 DDBitD ;
: main vorbereiten schalten ;
```

Zunächst wird durch das Wort `vorbereiten` das Bit 6 des Datenrichtungsregisters von D auf 1 gesetzt; Port D.6 wird also als Ausgang konfiguriert. Das Wort `schalten` besteht aus einer Endlosschleife; zu Beginn der Schleife kontrolliert das Wort `Ta0?`, ob der Taster Ta0 gedrückt ist oder nicht. Ist Ta0 gedrückt, legt es den Wert 0 auf den Stack, sonst den Wert 1. Ähnlich wie schon bei dem Wort "." wird der zugehörige Eingang Port D.2 von dem Wort `Ta0?` automatisch konfiguriert (Eingang und Pull-up).

Anschließend wird die Zahl 6 auf den Stack gelegt; `swap` tauscht diesen Wert 6 mit dem von `Ta0?` gelieferten Ein-Aus-Wert aus. Nun liegen der Bit-Wert 6 und der Ein-Aus-Wert genau in der Reihenfolge auf dem Stapel, wie sie von `OutPortD` benötigt werden: unten der Bit-Wert und oben der An-Aus-Wert (im Vokabular als Flag bezeichnet). `6 1 OutPortD` schaltet z. B. die LED an PortD.6 an; `6 0 OutPortD` schaltet sie aus.

> ***Beachten Sie***: Nur bei den Wörtern ".", `Ta0?`, `Ta1?` und `blink` erfolgt eine automatische Konfigurierung der Ports; bei allen anderen Port-Befehlen müssen die Datenrichtungsbytes bzw. -Bits vom Anwender selbst mithilfe der Wörter `DDRB`, `DDRD`, `DDBitB` und `DDBitD` eingestellt werden.

Im dritten Beispiel soll eine Blink-Schleife über den Taster Ta0 abgebrochen werden. Genauer gesagt soll das Bitmuster 01010101 solange ein- und ausgeschaltet werden, bis der Taster Ta0 gedrückt wird. Das zugehörige Programm ist auch wieder sehr kurz und sieht so aus:

```
: main begin 85 100 blink Ta0? not until ;
```

Innerhalb der BEGIN-UNTIL-Schleife wird zunächst das `blink`-Wort mit dem Bitmuster &B01010101 = 85 und der halben Periodendauer 100 ms ausgeführt. Anschließend wird mit `Ta0?` der Zustand des Tasters Ta0 abgefragt; ist dieser Taster gedrückt, wird eine 0 auf den Stapel gelegt, sonst eine 1. Ohne das folgende Wort `not` würde dieser Ein-Aus-Wert des Tasters direkt von `until` ausgewertet: der Zustandswert 0 (Taster gedrückt) würde zu einem weiteren Schleifendurchlauf führen und der Zustandswert 1 (Taster offen) würde die Schleife abbrechen. Die Schleife würde also durch ein Öffnen und nicht - wie gefordert - durch ein Schließen des Tasters beendet.

Um zu einem korrekt funktionierenden Programm zu gelangen, muss also der Zustandswert umgekehrt werden: Aus dem Wert 1 muss eine 0 und aus dem Zustandswert 0 muss eine 1 gemacht werden. Dies kann man mit dem Wort `not` erreichen: Dieses Wort holt den Zustandswert vom Stapel und ersetzt ihn durch sein logisches Komplement. Die Wortfolge `Ta0? not` liefert jetzt wie gewünscht auf dem TOS den Wert 0, wenn der Taster offen ist, und den Wert 1, wenn der Taster geschlossen ist.

Aufgabe 1

Ein Blick in den FORTH-Editor zeigt, wie das Wort `Ta0?` definiert ist.

```
: Ta0? 2 0 DDBitD 2 1 outPortD 2 InPortD ;
```

Erläutern Sie diese Definition.

Aufgabe 2

Wie ließe sich das Programm zum ersten Beispiel mithilfe des `blink`-Wortes vereinfachen?

Aufgabe 3

Ändern Sie das Programm des zweiten Beispiels so ab, dass die LED leuchtet, wenn der Taster gedrückt ist, und sonst nicht.

Aufgabe 4

Der Attiny soll die Anzahl der Tastendrücke beim Taster Ta0 an PortB anzeigen; PortB muss entsprechend mit 8 LED bestückt werden. Beachten Sie: Der Tastendruck dauert immer eine gewisse Zeit; selbst bei flotten Menschen bleibt der Taster für mehrerer Millisekunden geschlossen, für den Attiny ist das aber eine halbe Ewigkeit!