Forth.voc

Forth-Vokabular

Stand: 01.11.2012

A: Assembler-Wort F: Forth-Wort C: Compiler-Wort

Wort	Typ	Kommentar	Stack
`.`	A	gibt TOS auf Port B aus; (Datenrichtungsbits von Port B werden alle auf 1gesetzt.)	(n –)
`-`	A	erg = a - b	(a b – erg)
`/`	A	dividiert a (ganzzahlig) durch b erg = a/b (ohne Rest)	(a b – erg rest)
`:`	C	leitet Doppelpunktdefinition ein.	
`;`	C	schließt Doppelpunktdefinition ab.	
`<`	A	`a b <` legt 1 (TRUE) auf den Stack, wenn a < b ist, sonst 0.	(a b – flag)
`>`	F	`a b >` legt 1 (TRUE) auf den Stack, wenn a > b ist, sonst 0.	(a b – flag)
`>com`	A	sendet TOS an die COM-Schnittstelle. Vorher muss die COM-Schnittstelle mit INITCOM initialisiert worden sein.	(n –)
`>eprom`	A	schreibt den Wert w in die Adresse a des EEPROMs. Vgl. `eprom>`	(w a –)
`>R`	A	schiebt den TOS auf den Returnstack Vgl. `R>`	(a –)
`>sram`	A	speichert den Wert w in der SRAM-Zelle mit der Adresse a	(w a –)
`1 bis 255`	AC	legt die Zahl auf den Stack.	(– n)
`and`	A	erg = a and b	(a b – erg)

Wort	**Typ**	**Kommentar**	**Stack**

`begin ... until`	A	`begin Bef1 Bef2 ... Befn until`
		wiederholt die Befehle `Bef1`, `Bef2`, ..., `Befn`, bis `until` auf TOS = 1 stößt.
		`begin`
		`until`
			(–)
			(n –)

`blink`	F	*bitmuster hp* `blink` gibt *bitmuster* auf Port B aus, wartet *hp* Millisekunden, gibt 0 auf Port B aus und wartet wieder *hp* Millisekunden.	(b hp –)
`com>`	A	legt über COM-Schnittstelle empfangenes Byte auf Stack. Vgl. `>com`.	(– n)
`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* `DDBitB` setzt den Anschluss *bit* des Ports B 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 –)

`do ... loop`	A	*e a* `do Bef1 Bef2 ... Befn loop`
		wiederholt die Befehle `Bef1`, `Bef2`, ..., `Befn`; die Schleife beginnt mit dem Index *a* und läuft bis *e* (einschließlich).
		Die Schleife wird mindestens einmal durchlaufen. Innerhalb der Schleife kann durch das Wort `I` auf den Index zurückgegriffen werden.
		`do`
		`loop`
			(e a –)
			(–)

`drop`	A	entfernt den TOS	(n –)
`dup`	A	dupliziert den TOS	(n – n n)

Wort	**Typ**	**Kommentar**	**Stack**
`end`	A	führt eine Endlosleerschleife aus; wird für das Ende eines Programms empfohlen.	(–)
`eprom>`	A	liest den Wert w aus der EEPROM-Adresse a und legt ihn auf den Stack Vgl. `>eprom`	(a – w)
`getOSCCAL`	A	legt den OSCCAL-Wert auf den Stack. Vgl. `SetOSCCAL`	(– n)
`I`	A	legt den Schleifenindex einer `do-loop`-Schleife auf den Stack. Darf nur zwischen `do` und `loop` auftauchen.	(– n)
`i2cread`	A	Ein Wert wird vom Slave gelesen; wenn ACK = 0 ist, wird ein Acknowledge-Signal gegeben.	(ACK – Wert)
`i2cstart`	A	Startsignal für I2C-Bus wird gesendet (SDA von 1 auf 0; dann SCL von 1 auf 0)	(–)
`i2cstop`	A	initialisiert den I2C-Bus (SCL und SDA auf 1); Datenrichtungsbits für SDA (PortB.5) und SCL (PortB.7) werden gesetzt.	(–)
`i2cwrite`	A	Ein einzelnes Byte wird an den Slave gesendet; das Acknowledge-Signal wird auf den Stack gelegt.	(Wert/Adr – ACK)
`init`	F	Systemwort, darf nicht geändert oder entfernt werden.	
`initCom`	A	initialisiert die COM-Schnittstelle: D0 = RxD D1 = TxD Baudate = 9600 8 Bit kein Paritätsbit	(–)

Wort	**Typ**	**Kommentar**	**Stack**
`initInt0`	A	*signaltyp* `initInt0` konfiguriert INT0 (Port D2) als Interrupteingang und legt diesen auf High. Je nach *signaltyp*-Wert lösen unterschiedliche Eingangssignale den Interrupt aus: 0: fallende Flanke 1: steigende Flanke Interrupts werden generell zugelassen.	(signaltyp –)
`initInt1`	A	wie `initInt0`, jedoch für den Eingang INT1 (Port D3).	(signaltyp –)
`initT0ovf`	A	*typ preset* `initT0ovf` initialisiert den Timer0-Interrupt: *typ* 0: Timer stoppen/deaktivieren 1: Systemtakt/1 2: Systemtakt/8 3: Systemtakt/64 4: Systemtakt/256 5: Systemtakt/1024 6: ext. Takt, fallend an T0 7: ext. Takt, steigend an T0 Timer-Interrupt werden freigegeben alle Interrupts werden freigegeben *preset*-Wert muss in Interruptroutine immer wieder neu gesetzt werden.	(typ preset –)
`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**

`int0`	F	Dieses Wort wird aufgerufen wenn das INT0-Interrupt ausgelöst wird.
		Aufbau eines Interruptwortes:
		`: int0 pushreg ...`
		`<beliebige Wörter> ...`
		`popreg reti;`
		Während das Wort `int0` ausgeführt wird, sind sämtliche Interrupts gesperrt.
			(–)

`int1`	F	Dieses Wort wird aufgerufen, wenn das INT1-Interrupt ausgelöst wird. Kann beliebig definiert werden.	(–)
`not`	F	ersetzt *flag* durch sein logisches Komplement.	(*flag* – $\overline{flag}$)
`or`	A	erg = a or b	(a b – erg)
`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 –)
`over`	A	kopiert das zweite Element des Stacks auf den TOS.	(a b – a b a)
`popreg`	A	Sämtliche internen Register r16-r29 werden wiederhergestellt.	(–)
`pushreg`	A	Sämtliche internen Register r16-r29 werden gesichert (in r2-r15).	(–)
`R>`	A	holt das oberste Element des Returnstacks und legt es auf den (Arbeits-) Stack. Vgl. `>R`	(– a)
`reti`	A	Interrupts werden freigegeben.	(–)
`rot`	A	rotiert die obersten drei Zahlen des Stacks.	(a b c – b c a)
`sei`	A	wie `reti`	
`setOSCCAL`	A	schreibt den Wert n in das OSCCAL-Register.	(n –)

Wort	**Typ**	**Kommentar**	**Stack**
`setTimer0`	A	setzt den Preset-Wert (TCNT0) von Timer0.	(preset –)
`skipIf`	A	überspringt den nächsten Befehl, wenn TOS gleich 1 (TRUE) ist.	(n –)
`sram>`	A	legt den Wert der SRAM-Zelle a auf den Stack	(a – w)
`stackInit`	A	Systemwort, darf nicht geändert oder entfernt werden	
`swap`	A	vertauscht die beiden obersten Zahlen des Stacks.	(n m – m n)
`T0ovf`	F	Dieses Wort wird aufgerufen, wenn das Timer0-Overflow-Interrupt ausgelöst wird. Zum Aufbau eines Interrupt-Wortes vgl. `int0`. Innerhalb von `T0ovf` muss ggf. der Preset-Wert des Timers mit `setTimer0` gesetzt werden.	(–)
`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.	
`toggleB`	A	toggelt das Register von Port B.	(–)
`VARIABLE`	C	leitet Variablendeklaration ein. Durch `VARIABLE` *abc* wird die Variable *abc* deklariert. Dadurch wird durch den Compiler im EEPROM ein Speicherplatz reserviert. Anschließend wird durch *abc* die Adresse von der zugehörigen Speicherstelle auf den Stapel gelegt.	
`wait`	F	wartet s Sekunden.	(s –)
`wait1ms`	A	wartet 1 Millisekunde.	(–)
`waitms`	F	wartet n Millisekunden.	(n –)

Wort	**Typ**	**Kommentar**	**Stack**
`wdogOff`	A	schaltet den Watchdog aus.	(–)
`wdogOn`	A	schaltet den Watchdog an.	(–)
`xor`	A	erg = a xor b	(a b – erg)
=	A	*a b* = legt 1 (TRUE) auf den Stack, wenn *a* = *b* ist, sonst 0 (FALSE).	(a b – flag)
+	A	erg = a + b	(a b – erg)
⋆	F	erg = a * b	(a b – erg)