PIC-Assembler - Befehle

zurück zu PIC-Prozessoren , Elektronik , Homepage

Befehlsübersicht

ADDLW , ADDWF , ANDLW , ANDWF , BCF , BSF , BTFSC , BTFSS , CALL , CLRF , CLRW , CLRWDT , COMF

DECF , DECFSZ , GOTO , INCF , INCFSZ , IORLW , IORWF , MOVF , MOVLW , MOVWF , NOP

RETFIE , RETLW , RETURN , RLF , RRF , SLEEP , SUBLW , SUBWF , SWAPF , XORLW , XORWF

Befehlsübersicht nach Gruppen

Kopierbefehle (MOV...)
Löschbefehle (CLR...)
Setzen und Löschen einzelner Bits, Bitverschiebungen, Bitvertauschungen (BSF, BCF, RLF, RRF, SWAPF)
Aritmetische Operationen (ADD.., SUB.., COM..., AND..., IOR..., XOR...)
Increment und Decrement (INC... und DEC...)
Steuerbefehle und Anderes (NOP, BTFSC, BTFSS, GOTO, CALL, RETURN, RETLW, RETFIE, SLEEP, CLRWDT)

Quellen

Immer wieder werde ich nach einem deutschsprachigen Buch gefragt, in dem die Programmierung der PICs erläutert ist. Ich kenne keines, obwohl es bestimmt gute Bücher dieser Art gibt. Ich habe alles was ich weiß aus den Veröffentlichungen von Microchip. Für jeden PIC-Typ gibt es auf der Microchip-Homepage ein Datenblatt, das auch die Programmierung beschreibt. Ich empfehle jedem, der des Englischen mächtig ist, diese Datenblätter zu nutzen. Das sind die Originalquellen, und sie bieten deshalb den höchsten Standard an Fehlerfreiheit.

Jede Sekundärliteratur, also jedes Buch jeder Zeitschriftenartikel und z.B. meine Homepage sind naturgemäß fehlerträchtiger als das Orginal.

Trotzdem möchte ich nachfolgend in deutsch die Befehle des PIC erläutern. Wer mir guten Gewissens ein tolles deutsches Buch zu diesem Thema empfehlen kann, soll das gern per eMail tun. Ich bin gern bereit an dieser Stelle gute Sekundärliteratur zu erwähnen.

nach oben

Allgemeines

Den PIC betrachtet man am Besten als einen 8-Bit Prozessor, mit nur einem Arbeitsregister - dem Register 'W'.

Es gibt Ein-Adressbefehle und Zwei-Adressbefehle. Ein Ein-Adressbefehl bezieht sich nur auf das Arbeitsregister oder nur auf eine Speicherzelle. Ein Beispiel hierfür ist der Löschbefehl, der eine Speicherzelle oder W auf den Wert 0 setzt. Zwei-Adressbefehle arbeiten immer mit dem Arbeitsregister und einer Speicherzelle. Ein Beispiel ist der MOV-Befehl, der den Wert einer Speicherzelle in das Arbeitsregister kopiert (oder umgekehrt).

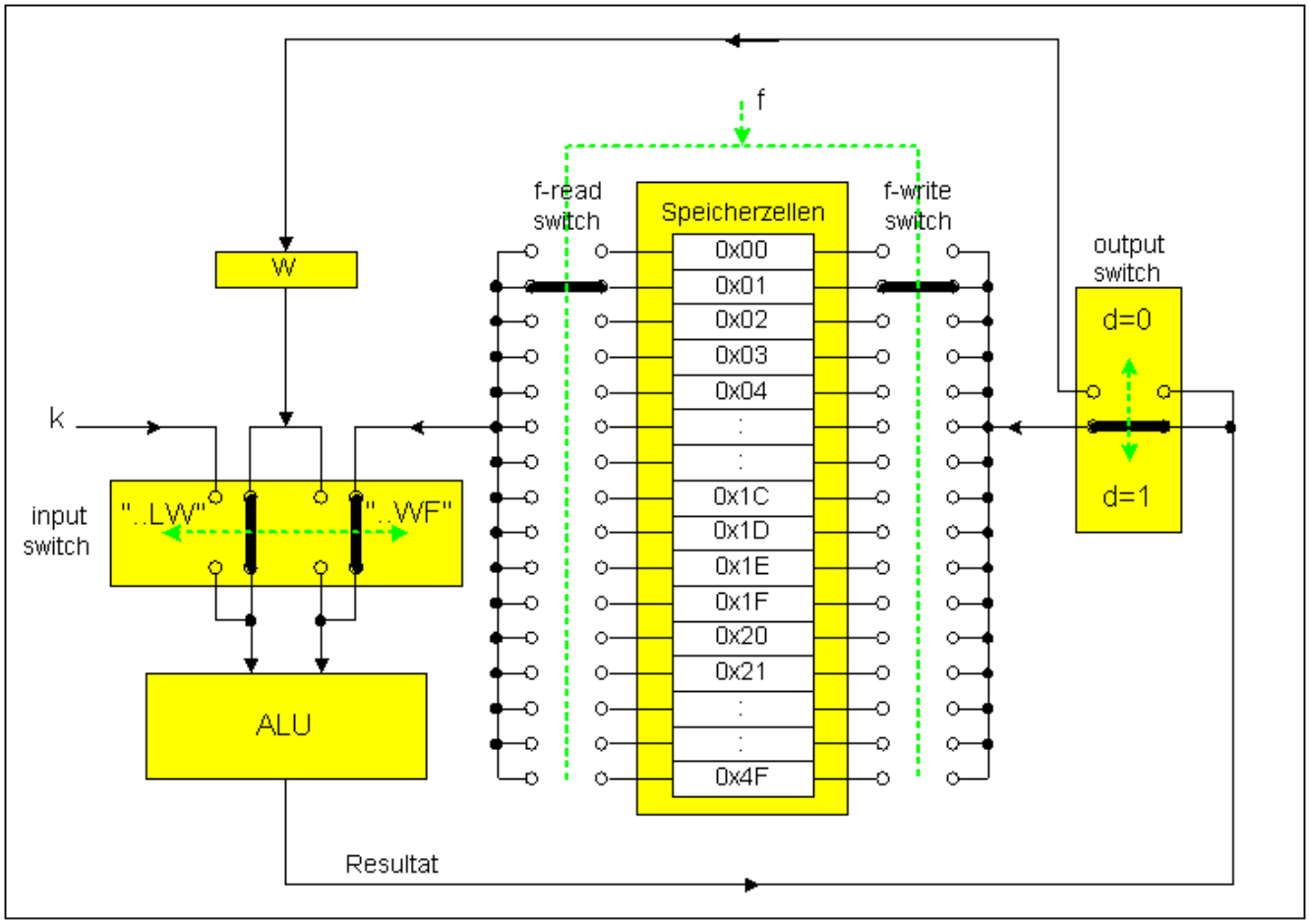
Somit ist es nicht möglich, den Wert einer Speicherzelle unter Umgehung von W direkt in eine andere Speicherzelle zu kopieren. Hier muß der Wert zunächst aus der Speicherzelle nach W kopiert werden, um anschließend von W in die andere Speicherzelle kopiert zu werden. Dafür werden also zwei Befehle benötigt.

Zu den Zwei-Adressbefehlen gehören auch die "literal"-Befehle, das sind Befehle mit Zahlen. Hier ersetzt ein fester Zahlenwert die Speicherzelle. So kann man z.B. eine Zahl in das Arbeitsregister laden, oder eine Zahl zum Arbeitsregister dazuaddieren. Der Zahlenwert ist dann Teil des Befehlscodes, und muß nicht vorher in einer Speicherzelle stehen. Literal-Befehle arbeiten immer nur mit dem Arbeitsregister zusammen. Man kann also keine Zahl direkt in eine Speicherzelle laden, sondern nur in das Arbeitsregister, von dem es mit einem weiteren Zwei-Adressbefehl (MOV) in die Speicherzelle kopiert werden muß.

Für eine normale Addition nach dem Muster a+b=c würde man eigentlich drei Adressen benötigen (je eine für a, b und c) solche Befehle gibt es aber nicht. Auch Additionen und Subtraktionen sind Zwei-Adressbefehle. Der Wert aus dem Arbeitsregister W wird zunächst mit dem Wert aus einer Speicherzelle addiert. Die resultierende Summe wird dann (je nach Befehl) ins Arbeitsregister oder in die selbe Speicherzelle geschrieben. Einer der beiden Ausgangswerte wird dabei also vernichtet. (a := a+b)

Multiplikation, Division oder komplizierteres gibt es als Assemblerbefehl gar nicht. Benötigt man so eine Operation, muß man auf ein Unterprogramm zurückgreifen, das diese Funktionen als Algoritmus realisiert. Solche Algoritmen gibt es aber schon fertig z.B. bei Microchip.

Das kling alles nach einer Beschränkung auf ein Minimum, und das war auch die Idee der Designer. Der PIC ist ein RISC-Prozessor, der nur über einen minimalen Befehlsvorrat verfügt. Das macht den Chip schnell und billig. Gerade einmal 35 unterschiedliche Befehle muß man als Programmierer kennen, mehr gibt es nämlich nicht.



Die Grafik zeigt den möglichen Datenfluß im Prozessor bei einem Zweiadressbefehl. Die zwei Eingänge der ALU ("Rechenzentrale" des Chip) werden entweder von einer Zahl (k) und W oder von W oder einer Speicherzelle gespeist. Das steuert ein "input swich", der bei allen Befehlen der Form "...LW" (literal-Befehl) in der linken und bei "...WF"-Befehlen in der rechten Position steht.
Das Resultat der Operation geht aus der ALU über den "output switch" in eine Speicherzelle oder zurück in W. Dieser

"output switch" wird vom Wert des Bits "d" gesteuert. Welche Speicherzelle genau in die Operation einbezogen ist, steuern der "f-read" und "f-write switch". Beide stehen in der selben Position, die durch den Wert "f" vorgegeben ist. "k", "d", "f" sowie "..LW" und "..WF" stammen aus dem Befehl, den der Prozessor gerade abarbeitet. Der Befehl enthält darüberhinaus noch die Art der mathematischen Operation (Addition, Subtraktion ...) die die ALU ausführen soll.

In der momentanen Schalterstellung werden die Werte aus W und der Speicherzelle 01h miteinander verknüpft (z.B. addiert). Das Resultat wird wieder in der Speicherzelle 01h gespeichert.

nach oben

Speicherzellen (am Beispiel PIC16F84)

Wie schon erwähnt besitzt der Prozessor neben dem 8-Bit-Arbeitsregister W noch **Speicherzellen** , die auch 8-Bit breit sind. Jede Speicherzelle besitzt eine eigene Adresse, mit der sie angesprochen (addressiert) wird. Im 16F84 gibt es Speicherzellen von der Adresse 00h bis zur Adresse 4Fh und noch eine weite Gruppe von 80h bis CFh. Diese beiden Gruppen werden als Bank 0 und Bank 1 bezeichnet.

Viele dieser Speicherzelle dienen nur der Speicherung von 1-Byte Daten. Andere werden zur Steuerung des Prozessors benutzt. Als reine Speicherzellen werden im 16F84 die Zellen mit den Adressen 0Ch bis 4Fh und 8Ch bis CFh verwendet. In Wirklichkeit verbergen sich hinter diesen zwei Adressgruppen die selben Speicherzellen. Die Zelle mit der Adresse 0Ch hat nämlich auch die Adresse 8Ch, die Zelle 0Dh kann auch mit 8Dh addressiert werden u.s.w. Somit stehen uns 68 freie Speicherzellen zur Verfügung.
In größeren PICs hat jede Bank eigene Speicherzellen. Im 16F84 sind nur die zur Steuerung des PIC benütigten Speicherzellen 00h-0Bh und 80h bis 8Bh in den beiden Bänken verschieden.

Die 68 Byte nehmen sich neben den mehrere 100MByte großen Hauptspeichern moderner PCs recht bescheiden aus, aber für viele Zwecke reicht das aus.

Von 00h bis 0Bh und 80h bis 8Bh liegen Speicherzellen, die zur Steuerung des Prozessors verwendet weren. Diese **Steuerregister** werden wie normale Speicherzellen beschrieben und gelesen. Ihre Funktionen sind im Datenblatt des PIC erläutert. Da ihre Addressierung mit Hilfe der hexadezimalen Adresse umständlich ist (wer kann sich schon die ganzen Zahlen merken) sind für die Steuerregister leicht zu merkende Aliasnamen eingeführt worden. Die Zuordnung der Aliasnamen zu den physischen Adressen steht im der *.INC-Datei, die für den PIC16F84 z.B. P16f84.INC heißt. Um die Aliasnamen verwenden zu können, muß im Assemblerprogram das INC eingebunden werden. Das geschieht mit folgender Zeile am Anfang des Programms:

```
#include <P16f84.INC>
```

Auch für die einfachen Speicherzellen kann man beliebige Aliasnamen festlegen. Dazu dient im Assemblerprogramm der EQU-Befehl. Das ist eigentlich gar kein richtiger Befehl, sondern nur ein Hinweis für den Assembler, das in Zukunft eine bestimmte Speicherzelle mit einem bestimmten Namen addressiert wird:

```
Temp Equ 0x10
```

Diese Zeile legt für die Speicherzelle mit der Adresse 10h (hier Schreibweise 0x10) den neuen Namen Temp fest. Man muß bei diesen Namen übrigens die Groß- und Kleinschreibung beachten!

nach oben

Flags

Flags sind einzelne Bits im Prozessor, die sich Besonderheiten eines Rechenergebnisses merken. Sie werden für die Ablaufsteuerung des Programms dringend gebraucht. Alle Flags stehen im Register STATUS (Adresse 03h). An dieser Stelle mögen die beiden Flags genügen, die in den Bits 0 und 2 des STATUS-Registers stehen:

bit 0 : Das **Carry**-Bit
Mit seiner 8-Bit Datenbreite kann der PIC (ohne spezielle Algoritmen) nur mit Zahlen von 0 bis 255 rechnen. Wird der Wert 255 überschritten, so fängt der PIC wieder bei 0 an. So ergibt die Berechnung 255+1 das lustige Ergebnis 0 und 250+20 ergibt 14. Allerdings fällt dem PIC dieses Überlaufen auf, und er setzt in diesem Fall das Carry-Bit auf 1. Bei einer Addition ohne Überlauf bleibt das Carry-Bit dagegen auf 0.
ACHTUNG: Bei einer Subtraktion verhält sich das Carry-Bit genau verkehrt herum. 20-5=15 setzt Carry auf 1 aber 20-25=251 löscht das Carry-Flag.

bit 2 : Das **Zero**-Bit
Ergibt eine Operation zufällig genau Null, so wird das Zero-Bit auf 1 gesetzt. Ergibt die Operation ein anderes Ergebnis, so geht das Zero-Bit auf 0.

Nicht alle Operationen, von denen man es erwarten würde, beeinflussen die Flags. So beeinflußt der INCF-Befehl, der den Inhalt einer Speicherzelle um 1 erhöht, zwar das Zero-Bit, aber nicht das Carry-Bit. In der unten folgenden Beschreibung der einzelnen Befehle sind die beeinflußten Flags jeweils aufgelistet.

nach oben

Schreibregeln

Einige Zeichen werden im folgenden Text immer wieder auftreten. Deshalb folgt nun ihre Erläuterung:

f	eine Speicherzelle
d	Ergebins wird gespeichert in: d=0: Arbeitsregister W d=1: Speicherzelle
W	das Arbeitsregister
k	ein Zahlenwert von 0 ... 255 (bei CALL und GOTO: 0..2047)
b	ein Zahlenwert von 0 bis 7

Die Befehle

Kopierbefehle (MOV...)

Mit MOV-Befehlen werden Werte im Prozessor "transportiert" also von einer Speicherzelle oder dem Arbeitsregister in eine andere Speicherzelle kopiert. Der Begriff "kopieren" ist dabei genauer als das englische "move" was bewegen bedeuten würde. Die Speicherzelle, aus der der Wert kommt behält diesen Wert nämlich beim MOV-Befehl unverändert bei, und eine Kopie ihres Inhalts wird in der Ziel-Speicherzelle angelegt.
Mit dem MOVLW-Befehl lassen sich feste Zahlenwerte in die Speicherzellen bringen.

MOVF	Kopiere den Inhalt der Speicherzelle f nach...
Syntax:	`MOVF f,d`

Bedeutung:	`wenn d=0:`
	`Der Inhalt der Speicherzelle f wird in das Arbeitsregister kopiert`
	
	`wenn d=1`
	`Der Inhalt der Speicherzelle wird in die selbe Speicherzelle kopiert. Es passiert also gar nichts. Allerdings kann man dadurch prüfen, ob in der Speicherzelle der Wert 0 steht, da dann das Zero-Flag gesetzt werden würde.`

Beispiel:	`MOVF PORTA,0 ;Das Register PORTA wird in das Arbeitsregister kopiert`
Flags:	Z

MOVWF	Kopiere den Inhalt von W in die Speicherzelle f
Syntax:	`MOVWF f`
Bedeutung:	`Der Inhalt des Arbeitsregisters W wird in die Speicherzelle f kopiert`
Beispiel:	`MOVWF PORTA ;Das Arbeitsregister wird nach PORTA kopiert`

MOVLW	Kopiere einen Zahl (L) nach W
Syntax:	MOVLW k
Bedeutung:	Die Zahl k wird in das Arbeitsregisters W geschrieben
Beispiel:	MOVLW 5 ; Der Wert 5 wird in das Arbeitsregister geschrieben

Löschbefehle (CLR...)

Mit Löschbefehlen lassen sich Speicherzellen oder das Arbeitsregister auf 0 setzen. Dabei wird das Zero-Flag gesetzt.

CLRF	Lösche die Speicherzelle f
Syntax:	CLRF f
Bedeutung:	In die Speicherzelle f wird mit der Wert 0 geschrieben.
Beispiel:	CLRF PORTA ;In das Register PORTA wird 0 geschrieben
Flags:	Z=1

CLRW	Lösche W
Syntax:	CLRW
Bedeutung:	Das Arbeitsregisters W wird mit 0 beschrieben
Beispiel:	CLRW ;In das Arbeitsregister wird 0 geschrieben

Setzen und Löschen einzelner Bits, Bitverschiebungen, Bitvertauschungen (BSF, BCF, RLF, RRF, SWAPF)

Mit den BSF und BCF-Befehlen lassen sich einzelne Bits in einer beliebigen Speicherzelle auf 1 oder 0 stellen. Dabei werden Flags **nicht** beeinflußt.
Die Bits innerhalb einer 8-Bit-Speicherzelle tragen die Nummern 0 bis 7, wobei Bit Nr. 0 den kleinsten Wert hat (LSB = 1) wärend Bit Nr.7 den höchsten Wert besitzt (MSB = 128). Einzelne Bits im Arbeitsregister W lassen sich nicht direkt setzen.

BSF	Ein Bit (das Bit Nr. b) in einer Speicherzelle f setzen
Syntax:	BSF f,b
Bedeutung:	In der Speicherzelle f wird das Bit b auf 1 gesetzt
Beispiel:	BSF PORTA,3 ; Im Register PORTA wird das Bit Nr.3 auf 1 gesetzt ; da die Bits (vom kleinsten angefangen) 0,1,2,3...7 nummeriert werden ; ist Bit 3 das viert"kleinste" Bit mit dem Wert 8 ; War PORTA vorher Null, so ist jetzt PORTA=8.

BCF	Ein Bit (das Bit Nr. b) in einer Speicherzelle f löschen
Syntax:	BCF f,b
Bedeutung:	In der Speicherzelle f wird das Bit b auf 0 gesetzt
Beispiel:	BCF PORTA,2 ; Im Register PORTA wird das Bit Nr.2 auf 0 gesetzt ; (das dritt"kleinste" Bit mit dem Wert 4). ; Was PORTA vorher 6, so ist jetzt PORTA=2.

RLF	Alle Bits der Speicherzelle f um eine Position nach links verschieben
Syntax:	RLF f,d
Bedeutung:	In der Speicherzelle f werden alle Bits auf die nächst höhere Position verschoben. Bit 6 wandert zur Position 7, Bit 5 zur Position 6, ..., und Bit 0 zur Position 1. In die Position 0 wird der Wert des Carry-Flags eingetragen. Das Bit 7 wird in das Carry-Flag geschoben. Es ist ein ein linksherum-Rotieren der Speicherzelle f durch das Carry-Register um eine Position. wenn d=0: Das Ergebnis der Verschiebung wird Arbeitsregister W gespeichert. Die Speicherzelle f bleibt unverändert. wenn d=1 Das Ergebnis der Verschiebung wird wieder in der Speicherzelle f gespeichert.
Beispiel:	RLF 0x20,1 ; Im Register 20h rotiert alles um 1 Position nach links. ; War in 20h vorher der Wert 6 und das Carry-Flag=1, so steht ; jetzt in 20h der Wert 13 (dezimal) und C-Flag=0.
Flags:	C

RRF	Alle Bits der Speicherzelle f um eine Position nach rechts verschieben
Syntax:	RRF f,d
Bedeutung:	In der Speicherzelle f werden alle Bits auf die nächst niedrigere Position verschoben. Bit 1 wandert zur Position 0, Bit 2 zur Position 1, ..., und Bit 7 zur Position 6. In die Position 7 wird der Wert des Carry-Flags eingetragen. Das Bit 0 wird in das Carry-Flag geschoben. Es ist ein ein rechtsherum-Rotieren der Speicherzelle f durch das Carry-Register um eine Position. wenn d=0: Das Ergebnis der Verschiebung wird Arbeitsregister W gespeichert. Die Speicherzelle f bleibt unverändert. wenn d=1 Das Ergebnis der Verschiebung wird wieder in der Speicherzelle f gespeichert.
Beispiel:	RRF 0x20,1 ; Im Register 20h rotiert alles um 1 Position nach rechts. ; War in 20h vorher der Wert 13 (dezimal) und das Carry-Flag=0, so steht ; jetzt in 20h der Wert 6 und C-Flag=1.
Flags:	C

SWAPF	Obere und untere 4 Bit der Speicherzelle f austauschen
Syntax:	SWAPF f,d
Bedeutung:	Die obere und die untere Hälfte des Wertes in f werden ausgetauscht. wenn d=0: Das Ergebnis des Tauschs wird Arbeitsregister W gespeichert. Die Speicherzelle f bleibt unverändert. wenn d=1 Das Ergebnis des Tauschs wird wieder in der Speicherzelle f gespeichert
Beispiel:	SWAPF 0x20,1 ; Die obere und untere Hälfte des Werts in 20h ; werden vertauscht, War vorher in 20h der Wert 25h ; gespeichert, so steht nun eine 52h dort.

Aritmetische Operationen (ADD.., SUB.., COM..., AND..., IOR..., XOR...)

Das sind die Befehle zur "Datenverarbeitung". Hier wird also gerechnet.

ADDWF	Das Arbeitsregister mit einer Speicherzelle addieren
Syntax:	ADDWF f,d
Bedeutung:	Der Inhalt von W wird mit dem Inhalt von f addiert. wenn d=0: Das Ergebnis wird Arbeitsregister W gespeichert. Die Speicherzelle f bleibt unverändert. wenn d=1 Das Ergebnis wird wieder in der Speicherzelle f gespeichert
Beispiel:	ADDWF 0x20,1 ; Der Inhalt der Speicherzelle 20h wird mit dem ; Inhalt von W addiert. Die Summe wird in der ; Speicherzelle 20h abgelegt.
Flags:	C, DC, Z

ADDLW	Das Arbeitsregister mit einer Zahl addieren
Syntax:	ADDLW k
Bedeutung:	Der Wert von W wird um k erhöht.
Beispiel:	ADDLW 5 ; Der Wert von w wird um 5 erhöht.
Flags:	C, DC, Z

SUBWF	W von einer Speicherzelle abziehen
Syntax:	SUBWF f,d
Bedeutung:	Vom Wert, der in f steht, wird W abgezogen. wenn d=0: Das Ergebnis wird Arbeitsregister W gespeichert. Die Speicherzelle f bleibt unverändert. wenn d=1 Das Ergebnis wird wieder in der Speicherzelle f gespeichert.
Beispiel:	SUBWF 0x20,1 ; Der Inhalt der Speicherzelle 20h wird um den ; Inhalt von W vermindert. Das Ergebnis wird in der ; Speicherzelle 20h abgelegt.
Flags:	C, DC, Z

SUBLW	W von einer Zahl abziehen
Syntax:	SUBLW k
Bedeutung:	Von der Zahl k wird der momentane Wert von W abgezogen. Das Ergebnis wird wieder in W gespeichert.
Beispiel:	SUBLW 5 ; Die Differenz von k und W wird in W abgelegt. ; War W vorher 3, so ist W nun 2.
Flags:	C, DC, Z

COMF	eine Speicherzelle invertieren (Complement bilden)
Syntax:	COMF f,d
Bedeutung:	Alle Bits der Speicherzelle f werden invertiert. wenn d=0: Das Ergebnis wird Arbeitsregister W gespeichert. Die Speicherzelle f bleibt unverändert.

	wenn d=1 Das Ergebnis wird wieder in der Speicherzelle f gespeichert.
Beispiel:	COMF 0x20,1 ; Der Wert im Register 20h wird invertiert. ; Stand in 20h vorher 0, so steht dort jetzt 0FFh.
Flags:	Z

ANDWF	W und eine Speicherzelle mit der UND-Funktion verknüpfen
Syntax:	ANDWF f,d
Bedeutung:	Der Wert, der in f steht, wird mit W UND-verknüpft. wenn d=0: Das Ergebnis wird Arbeitsregister W gespeichert. Die Speicherzelle f bleibt unverändert. wenn d=1 Das Ergebnis wird wieder in der Speicherzelle f gespeichert.
Beispiel:	ANDWF 0x20,1 ; Der Inhalt der Speicherzelle 20h wird mit dem ; Inhalt von W UND-verknüpft. Das Ergebnis wird in der ; Speicherzelle 20h abgelegt.
Flags:	Z

ANDLW	W und eine Zahl mit der UND-Funktion verknüpfen
Syntax:	ANDLW k
Bedeutung:	Die Zahl k wird mit dem momentane Wert von W UND-verknüpft. Das Ergebnis wird wieder in W gespeichert.
Beispiel:	ANDLW 5 ; W wird mit 5 UND-verknüpft. Das Resultat ; wird wieder in W gespeichert. ; War W vorher 11h, so ist W jetzt 01h.
Flags:	Z

IORWF	W und eine Speicherzelle mit der ODER-Funktion verknüpfen (inclusive-ODER)
Syntax:	IORWF f,d
Bedeutung:	Der Wert, der in f steht, wird mit W ODER-verknüpft. wenn d=0: Das Ergebnis wird Arbeitsregister W gespeichert. Die Speicherzelle f bleibt unverändert. wenn d=1 Das Ergebnis wird wieder in der Speicherzelle f gespeichert.
Beispiel:	IORWF 0x20,1 ; Der Inhalt der Speicherzelle 20h wird um den ; Inhalt von W ODER-verknüpft. Das Ergebnis wird in der ; Speicherzelle 20h abgelegt.
Flags:	Z

IORLW	W und eine Zahl mit der ODER-Funktion verknüpfen (inclusive-ODER)
Syntax:	IORLW k
Bedeutung:	Die Zahl k wird mit dem momentane Wert von W ODER-verknüpft. Das Ergebnis wird wieder in W gespeichert.

Beispiel:	IORLW 5 ; W wird mit 5 ODER-verknüpft. Das Resultat ; wird wieder in W gespeichert. ; War W vorher 11h, so ist W jetzt 15h.
Flags:	Z

XORWF	W und eine Speicherzelle mit der Exclusiv-ODER-Funktion verknüpfen
Syntax:	XORWF f,d
Bedeutung:	Der Wert, der in f steht, wird mit W EX-ODER-verknüpft. wenn d=0: Das Ergebnis wird Arbeitsregister W gespeichert. Die Speicherzelle f bleibt unverändert. wenn d=1 Das Ergebnis wird wieder in der Speicherzelle f gespeichert.
Beispiel:	XORWF 0x20,1 ; Der Inhalt der Speicherzelle 20h wird um den ; Inhalt von W EX-ODER-verknüpft. Das Ergebnis wird in der ; Speicherzelle 20h abgelegt.
Flags:	Z

XORLW	W und eine Zahl mit der Exclusiv-ODER-Funktion verknüpfen
Syntax:	XORLW k
Bedeutung:	Die Zahl k wird mit dem momentane Wert von W EX-ODER-verknüpft. Das Ergebnis wird wieder in W gespeichert.
Beispiel:	XORLW 5 ; W wird mit 5 EX-ODER-verknüpft. Das Resultat ; wird wieder in W gespeichert. ; War W vorher 11h, so ist W jetzt 14h.
Flags:	Z

Increment und Decrement (INC... und DEC...)

Increment- und Decrement-Befehle erhöhen bzw. verringern den Wert einer Speicherzelle oder des Arbeitsregisters jeweils um 1. Sie eignen sich damit zum Aufbau von Zählschleifen. Läuft eine Speicherzelle beim Incrementieren über (255+1=0) oder beim Decrementieren unter (0-1=255) wird das Carry-Flag nicht gesetzt. Ist das Ergebnis von Increment oder Decrement aber 0 (255+1=0 oder 1-1=0), so wird das Zero-Flag gesetzt.

INCF	Erhöhe den Wert aus der Speicherzelle f um 1
Syntax:	INCF f,d
Bedeutung:	wenn d=0: Der Wert in f wird mit 1 addiert, und das Ergebnis in W gespeichert. wenn d=1: Der Wert in f wird mit 1 addiert, und das Ergebnis wieder in f gespeichert.
Beispiel:	INCF 0x20,1 ; Der Inhalt der Speicherzelle mit der Adresse 20h wird um 1 erhöht. ; War er vorher z.B. 45, so ist er jetzt 46.
Flags:	Z

DECF	Verringere den Wert aus der Speicherzelle f um 1
Syntax:	DECF f,d
Bedeutung:	wenn d=0: Vom Wert in f wird 1 abgezogen, und das Ergebnis in W gespeichert. wenn d=1: Vom Wert in f wird 1 abgezogen, und das Ergebnis wieder in f gespeichert.

Beispiel:	`DECF 0x20,1 ; Der Inhalt der Speicherzelle mit der Adresse 20h wird um 1 erniedrigt.` `; War er vorher z.B. 45, so ist er jetzt 44.`
Flags:	Z

Eine Besondere Form der DEC... und INC...-Befehle beinhaltet einen relativen Sprung: Falls das Ergebnis der Incrementierung oder Decrementierung 0 ist, wird der folgende Befehl übersprungen. Damit lassen sich einfach Schleifen aufbauen, die eine bestimmte Anzahl von Zyklen durchlaufen werden müssen. Da die Auswertung des Null-Zustandes schon intern erfolgt, wird das eigentliche Zero-Flag **nicht** beeinflußt.

INCFSZ	Erhöhe den Wert aus der Speicherzelle f um 1. Falls das 0 ergibt, dann ignoriere den nachfolgenden Befehl.
Syntax:	`INCFSZ f,d`
Bedeutung:	wenn d=0: Der Wert in f wird mit 1 addiert, und das Ergebnis in W gespeichert. wenn d=1: Der Wert in f wird mit 1 addiert, und das Ergebnis wieder in f gespeichert. Ist das Ergebnis der Addition Null, dann wird der nächste Befehl im Programm übersprungen, und mit dem übernächsten weitergebacht.
Beispiel:	`INCFSZ 0x20,1 ; Der Inhalt der Speicherzelle mit` `; der Adresse 20h wird um 1 erhöht`

DECFSZ	Verringere den Wert aus der Speicherzelle f um 1. Falls das 0 ergibt, dann ignoriere den nachfolgenden Befehl.
Syntax:	`DECFSZ f,d`
Bedeutung:	wenn d=0: Vom Wert in f wird 1 abgezogen, und das Ergebnis in W gespeichert. wenn d=1: Vom Wert in f wird 1 abgezogen, und das Ergebnis wieder in f gespeichert. Ist das Ergebnis der Subtraktion Null, dann wird der nächste Befehl im Programm übersprungen, und mit dem übernächsten weitergebacht.
Beispiel:	`DECFSZ 0x20,1 ; Der Inhalt der Speicherzelle mit` `; der Adresse 20h wird um 1 erniedrigt`

Beispiel für eine Programmschleife, die 5 Mal durchlaufen wird:

```
    MOVLW    5        ; 5 ins Arbeitsregister laden
    MOVWF    0x20     ; die 5 wird in die Speicherzelle 0x20 kopiert
LOOP                  ; eine Einsprungmarke
;    weitere Befehle in der Schleife
;    können hier eingefügt werden
    DECFSZ   0x20,1   ; der Wert in der Speicherzelle 20h wird um 1 verringert
    GOTO     LOOP     ; Sprung zur Marke LOOP

    nächster Befehl
```

Die ersten beiden Zeilen sin Vorbereitung. Von "LOOP" bis "GOTO LOOP" reicht die Schleife.
Der DECFSZ-Befehl wird immer am Schleifenende ausgeführt. Ist das Ergebnis nicht 0 (sondern 4, 3, 2 oder schließlich 1), so wird der darauf folgende "GOTO LOOP"-Befehl ausgeführt, und der Prozessor macht ab er oben stehenden LOOP-Marke weiter. Beim fünften Mal ist das Ergebnis des DECFSZ-Befehle Null, und der folgende Befehl wird ignoriert. Der GOTO-Befehl wird also nicht ausgeführt, und das Programm wird mit den Befehlen nach der GOTO-Zile fortgesetzt.

Steuerbefehle und Anderes (NOP, BTFSC, BTFSS, GOTO, CALL, RETURN, RETLW, RETFIE, SLEEP, CLRWDT)

Der NOP-Befehl tut gar nichts (no operation)

NOP	Einen Takt lang gar nichts tun
Syntax:	NOP
Bedeutung:	Der Prozessor legt für einen Arbeitszyklus eine Pause ein
Beispiel:	NOP ; nichts ändert sich
Flags:	keine

Die Bittestbefehle (BTF..) sind bedingte Sprünge. In Abhängigkeit vom Wert eines beliebigen Bits einer beliebigen Speicherzelle wird der auf diesen Befehl folgende Befehl ausgeführt oder übergangen. Ist dieser (bedingt ausgeführte) Befehl ein Sprungbefehl (GOTO), ergeben die beiden Befehle zusammen einen bedingten Sprung..
Mit den Bittestbefehlen werden in der Regel Sprünge in Abhängigkeit von den Flags realisiert. Die Flags sind einzelne Bits in der Speicherzelle mit dem Bezeichner STATUS (Speicherzelle 0x03 oder 03h). Das Zero-Bit ist dort das Bit Nr. 2 und das Carry-Flag ist das Bit Nr. 0 .
Die addressierung des Zero-Flag wäre demzufolge "STATUS,2" und des Carry-Flag "STATUS,0". Um das zu vereinfachen, wurden für 0 und 2 die Bezeichner C und Z eingeführt. Damit ist das Zero-Flag mit "STATUS,Z" und das Carry-Flag mit "STATUS,C" zu addressieren.

BTFSC	Übergehe nachfolgenden Befehl, wenn Bit=0 (bit test f, skip if clear)
Syntax:	BTFSC f,b
Bedeutung:	wenn in der Speicherzelle f das Bit Nr. b den Wert 0 hat, dann übergehen den nachfolgenden Befehl.
Beispiel:	BTFSC STATUS,Z ; prüfe das Zero-Flag GOTO Marke1 ; bei Z=1 wird diese Zeile ausgeführt: Sprung zu Marke1 GOTO Marke2 ; bei Z=0 wird diese Zeile ausgeführt: Sprung zu Marke2

BTFSS	Übergehe nachfolgenden Befehl, wenn Bit=1 (bit test f, skip if set)
Syntax:	BTFSS f,b
Bedeutung:	wenn in der Speicherzelle f das Bit Nr. b den Wert 1 hat, dann übergehen den nachfolgenden Befehl.
Beispiel:	BTFSS STATUS,C ; prüfe das Carry-Flag GOTO Marke1 ; bei C=0 wird diese Zeile ausgeführt: Sprung zu Marke1 GOTO Marke2 ; bei C=1 wird diese Zeile ausgeführt: Sprung zu Marke2

GOTO	Unbedingter Sprung
Syntax:	GOTO k
Bedeutung:	Springe im Programm zur Adresse k. (k ist 11 Bit lang) Normalerweise ist k eine Marke. Dadurch wird dann ein Bezeichner und keine nackte Zahl verwendet.
Beispiel:	GOTO Marke1 ; Springe zur Marke1 im Programm. . . Marke1 ; hier geht es weiter.

CALL	Sprung in ein Unterprogramm
Syntax:	CALL k
Bedeutung:	Speichere die Adresse des nachfolgenden Befehls im Stack. Dann springe zur Adresse k. (k ist 11 Bit lang) Normalerweise ist k eine Marke. Dadurch wird dann ein Bezeichner und keine nackte Zahl verwendet. An der Adresse k steht ein Unterprogramm, das mit dem Befehl RETURN oder RETLW enden muß.

Beispiel:	`CALL Marke1 ; Springe zur Marke1 im Programm.`
	`.`
	`.`
	`Marke1`
	`. ; hier beginnt das Unterprogramm`
	`.`
	`.`
	`RETURN ; hier endet das Unterprogramm,`
	`; springe zurück an die Stelle direkt hinter CALL`

Am Ende eines Unterprogramms kehr man mit RETURN und RETLW autiomatisch zu der Stelle im Hauptprogramm zurück, von der das Unterprogramm gerufen wurde.
RETLW setzt dabei W auf einen bestimmten Wert. Falls ein Unterprogramm mehrere mögliche "Enden" hat, kann man so leicht erkennen, welchen Ausgang das Unterprogramm genommen hat.

RETURN	Rückkehr aus einem Unterprogramm
Syntax:	RETURN
Bedeutung:	Hole den bei CALL gespeicherten Wert aus dem Stack, und springe zu dieser Adresse. Dort geht das Programm weiter, das dieses Unterprogramm aufgerufen hatte.
Beispiel:	RETURN ; hier endet das Unterprogramm

RETLW	Rückkehr aus einem Unterprogramm mit einem bestimmten Zahlenwert in W
Syntax:	RETLW k
Bedeutung:	Schreibe zuerst die 8-Bit-Zahl k in das Arbeitsregister W. Hole dann den bei CALL gespeicherten Wert aus dem Stack, und springe zu dieser Adresse. Dort geht das Programm weiter, das dieses Unterprogramm aufgerufen hatte.
Beispiel:	RETLW 1 ; hier endet das Unterprogramm ; das Unterprogramm endet mit einer 1 in W.

RETFIE wird nur benötigt, wenn man mit Interupts arbeitet.

RETFIE	Rückkehr aus der Interuptroutine
Syntax:	RETFIE
Bedeutung:	Setze das Bit "GIE" auf 1 (nun sind Interupts wieder erlaubt). Hole den bei der Auslösung des Interupts gespeicherten Wert aus dem Stack, und springe zu dieser Adresse. Dort geht das Programm weiter, das durch den Interupt unterbrochen wurde.
Beispiel:	RETFIE ; hier endet die Interutroutine.

Der Sleep-Befehl versetzt den Prozessor in den stromsparenden stand-by-Mode.
Durch die einige bestimmte Interrupts wird der PIC wieder geweckt. Falls das zum Interrupt zugehörige Interrupte-Enable-Bit gesetzt ist, läuft der PIC einfach im Programm weiter. Ist auch noch das GIE-Bit gesetzt, wird zusätzlich noch die Interruptbehandlungsroutine aufgerufen.

SLEEP	Prozessor in den Stand-By-Modus schalten
Syntax:	SLEEP
Bedeutung:	Der Prozessor wird in den Schlafmodus versetzt (stand-by). Dazu werden der WDT und sein Vorteiler auf 0 gesetzt. Das TO-Flag (inverses Time-out-Status-Bit)wird gesetzt. Das PD-Flag (inverses Power-Down-Status-Bit)wird gelöscht. Der Taktgenerator stoppt.
Beispiel:	SLEEP ; jetzt wird geschlafen
Flags:	TO, PD

CLRWDT ist nur nötig, wenn man den Watch-Dog-Timer nutzt.

CLRWDT	Löschen des Watch-Dog-Timers
Syntax:	CLRWDT
Bedeutung:	Der Watch-Dog-Timer und der Vorteile des Watch-Dog-Timers werden auf 0 gesetzt. Die Statusflags TO (inverses Time-out-Status-Bit) und PD (inverses Power-Down-Status-Bit) werden gesetzt.
Beispiel:	CLRWDT ; WDT und Prescaler=0.
Flags:	TO, PD

nach oben

zurück zu PIC-Prozessoren , Elektronik , Homepage

Autor: sprut
erstellt: 30.11.2001
letzte Änderung: 03.02.2006