

## ***Dokumentation PIC16F84 Simulator***

*von*

***Erik Brandner  
Andreas Malzew***

*Abgabedatum 23. Juni 2014*

***TINF12B3***

## Abbildungsverzeichnis

Abbildung 1 - GUI Oberfläche .....	5
Abbildung 2 - Funktionsbuttons .....	6
Abbildung 3 - Befehlsliste .....	7
Abbildung 4 - Speicheransicht .....	8
Abbildung 5 - Spezialregister .....	9
Abbildung 6 - Portübersicht .....	10
Abbildung 7 – Laufzeitanzeige .....	10
Abbildung 8 - Ablaufplan BTFSS .....	12
Abbildung 9 - Ablaufplan BTFSC .....	13
Abbildung 10 - Ablaufplan CALL .....	13
Abbildung 11 - Ablaufplan MOVF .....	14
Abbildung 12 - Ablaufplan RRF .....	15
Abbildung 13 - Ablaufplan SUBWF .....	16
Abbildung 14 - Ablaufplan DECFSZ .....	17
Abbildung 15 - Ablaufplan XORLW .....	17

## Inhaltsverzeichnis

Abbildungsverzeichnis .....	1
Inhaltsverzeichnis .....	2
1 Simulation .....	3
2 Der PIC16F84 Mikrocontroller .....	4
3 Simulationsprogramm .....	5
3.1 GUI.....	6
3.1.1 Buttons .....	6
3.1.2 Befehlsliste .....	7
3.1.3 Speicheransicht.....	8
3.1.4 Spezialregister .....	9
3.1.5 Port A und Port B .....	10
3.1.6 Stack und Laufzeit.....	10
4 Klassen .....	11
5 Befehle .....	12
5.1 BTFSS (Bit Test f, Skip if Set ) .....	12
5.2 BTFSC (Bit Test f, Skip if Clear ) .....	12
5.3 CALL (Call subroutine ) .....	13
5.4 MOVF (Move f) .....	14
5.6 RRF (Rotate Right f through Carry ) .....	14
5.7SUBWF (Subtract W from f ) .....	15
5.8 DECFSZ (Decrement f, Skip if 0 ).....	16
5.9 XORLW (Exclusive OR literal with W ) .....	17
6 Fazit.....	18

## 1 Simulation

Der Begriff Simulation bezeichnet die Nachahmung eines Systems, dessen Analyse sehr komplex und unter Umständen schwer durchzuführen ist. Die Simulation ermöglicht somit die Durchführung von Experimenten, deren Ergebnisse direkt auf das reale Modell bezogen werden können. Ein Simulator ist die Implementierung eines Simulationsmodells und stellt eine Abstraktion der Realität dar. In diesem Zusammenhang ist es sehr wichtig, dass die wichtigsten Aspekte beachtet und richtig implementiert werden.

Im Fall der Simulation des PIC1684F Mikroprozessor bedeutet das, dass kein echter Mikroprozessor zum Einsatz kommt. Die Testprogramme werden direkt ins Simulationsprogramm geladen und ausgeführt. Hieraus ergibt sich die Anforderung dass das Simulationsprogramm alle Funktionen des echten Mikrocontrollers bereitstellen muss.

Die Simulation birgt einige Vor- und Nachteile, die im Folgenden erläutert werden.

### **Vorteile:**

- Kostenersparnis, da keine Hardware benötigt wird.
- Vermeidung von Hardwaredefekten.
- Schnelles und einfaches laden des Programms.
- Anschauliches Design der GUI erleichtert die Fehlersuche.

### **Nachteile:**

- Verfälschte Ergebnisse durch fehlerhafte Implementierung.
- Implementierung der Simulation stellt einen großen Zeit- und Kostenfaktor dar.
- Mehraufwand durch notwendige Tests des Simulation-Programmcodes

## 2 Der PIC16F84 Mikrocontroller

Der PIC16F84 Mikrocontroller ist ein 8 Bit Mikrocontroller mit einem RISC-Befehlssatz. Der RISC Befehlssatz ist ein reduzierter Befehlssatz und umfasst 35 einfache Befehle. Die Befehle des PIC16F84 lassen sich in 3 Gruppen aufteilen: die Byte orientierten Befehle, die Bit orientierten Befehle und die Befehle die mit Konstanten arbeiten. Die meisten Befehle arbeiten mit einem Maschinenzyklus, vereinzelt Befehle benötigen zwei Maschinenzyklen. Die Befehle sind mit einem 14 Bit Opcode codiert.

Die PIC-Mikrocontroller-Familie besitzt eine sogenannten Harvard-Architektur, d. h. Daten- (RAM) und Programmspeicher (ROM) werden von der CPU über unterschiedliche Busse angesprochen. Dadurch kann sich die Ausführungsgeschwindigkeit verdoppeln, da in einem Takt auf beide Speicher gleichzeitig zugegriffen werden kann, im Gegensatz zur „Von-Neumann-Architektur“, in Zugriffe nur nacheinander möglich sind.

Der Datenspeicher ist in zwei Bereiche untergliedert. Im SFR (Special Function Registers) liegen, wie der Name erahnen lässt, spezielle Register die unter anderem Einfluss auf das Verhalten des PICs haben. In diesem Bereich sind die Adressen teilweise doppelt belegt, so dass zwischen Bank 0 und Bank 1 mittels des fünften Bits des Status-Registers umgeschaltet werden kann. Im GPR (General Purpose RAM) hingegen können beliebige Daten abgelegt werden.

## 3 Simulationsprogramm

In diesem Abschnitt wird der prinzipielle Aufbau des Mikrocontroller-Simulations-Programmes erläutert.

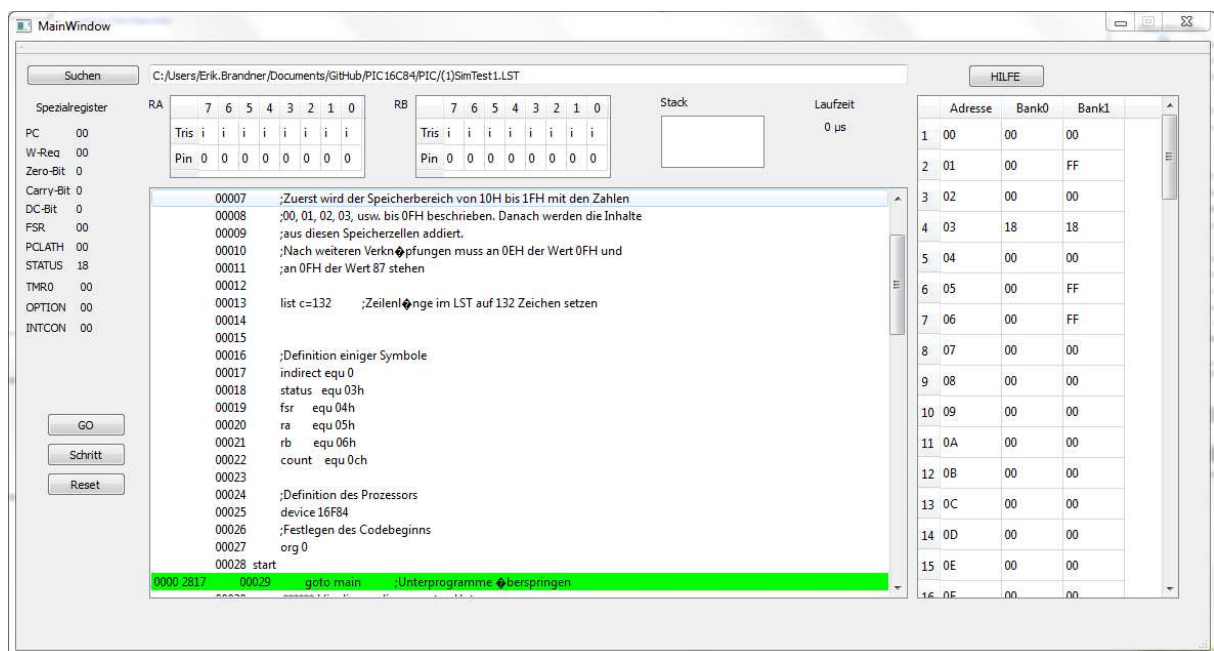


Abbildung 1 - GUI Oberfläche

## 3.1 GUI

Die grafische Benutzeroberfläche ist simpel und übersichtlich gestaltet. Auf die einzelnen Elemente wird im Folgenden näher eingegangen.

### 3.1.1 Buttons

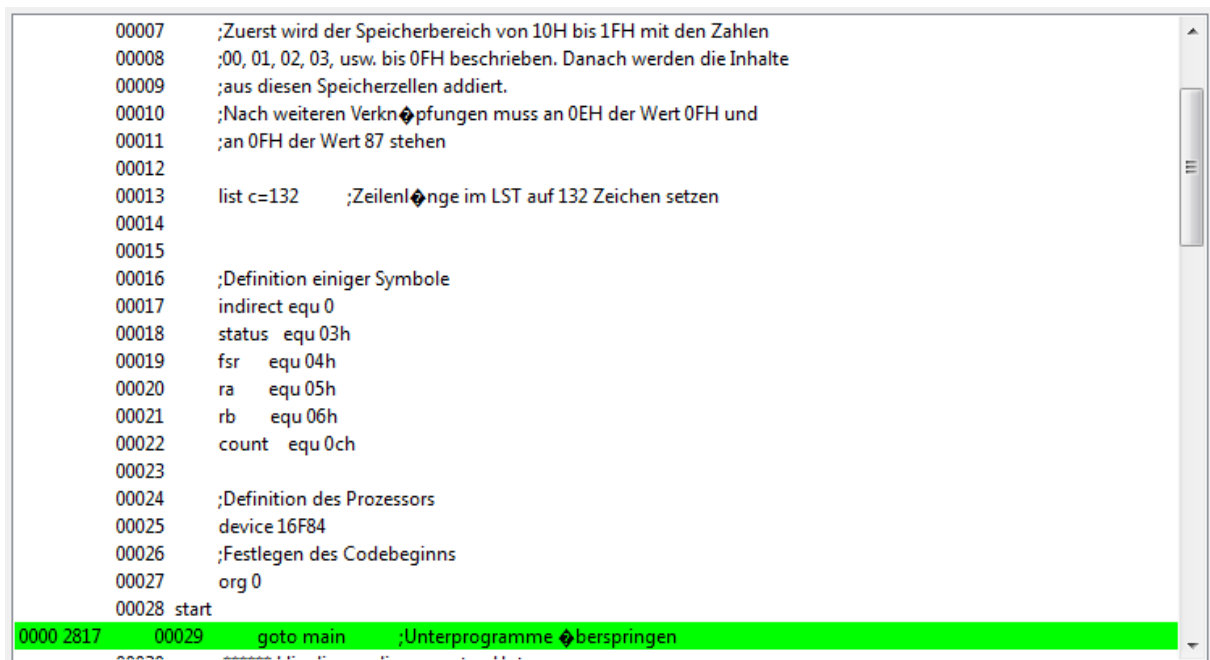
In der linken oberen Ecke befindet sich der Button zum Einlesen und Öffnen der LST-Datei. Weiter unten am linken Rand befinden sich die 3 Buttons zur Steuerung des Programmablaufes. Der GO-Button startet den Programmdurchlauf und ein weiterer Klick darauf hält das Programm wiederum an. Der Schritt-Button dient der schrittweisen Ausführung des Programmes und der Reset-Button setzt die benötigten Register und Speicherelemente auf ihren Ursprungszustand zurück. In der rechten oberen Ecke befindet sich schließlich noch der Hilfe-Button über den man die Dokumentation bezüglich dieses Programmes aufrufen kann.



Abbildung 2 - Funktionsbuttons

#### 3.1.2 Befehlsliste

Zentral gelegen befindet sich die Befehlsliste, die den größten Teil der Oberfläche für sich beansprucht. Zu Beginn des Programmes ist die Liste noch leer, sobald man über den Suchen-Button jedoch eine LST-Datei eingelesen hat füllt sich die Ansicht allerdings mit Inhalt. Die Zeilen der LST-Datei werden dann auf dem Bildschirm und Breakpoint können durch einen Doppelklick auf die gewünschte Zeile gesetzt werden. Die aktuelle Zeile des Programmes wird beim Durchlauf grün hinterlegt.



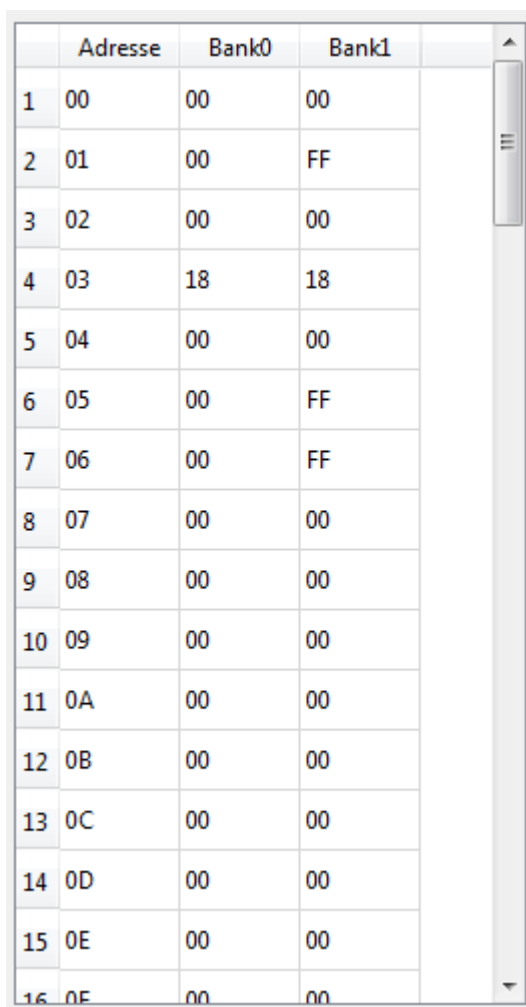
```
00007 ;Zuerst wird der Speicherbereich von 10H bis 1FH mit den Zahlen
00008 ;00, 01, 02, 03, usw. bis 0FH beschrieben. Danach werden die Inhalte
00009 ;aus diesen Speicherzellen addiert.
00010 ;Nach weiteren Verknüpfungen muss an 0EH der Wert 0FH und
00011 ;an 0FH der Wert 87 stehen
00012
00013 list c=132 ;Zeilenlänge im LST auf 132 Zeichen setzen
00014
00015
00016 ;Definition einiger Symbole
00017 indirect equ 0
00018 status equ 03h
00019 fsr equ 04h
00020 ra equ 05h
00021 rb equ 06h
00022 count equ 0ch
00023
00024 ;Definition des Prozessors
00025 device 16F84
00026 ;Festlegen des Codebeginns
00027 org 0
00028 start
0000 2817 00029 goto main ;Unterprogramme überspringen
```

Abbildung 3 - Befehlsliste



#### 3.1.3 Speicheransicht

Die Speicheranzeige dient zur Visualisierung der Speicherbänke Bank 0 und Bank 1 die jeweils die entsprechenden Register enthalten. Die linke Spalte Adresse gibt den Adressbereich in hexadezimaler Schreibweise an und die beiden Spalten Bank 0 und Bank 1 enthalten dann die dementsprechenden aktuellen Werte ebenfalls als hexadezimaler Wert. Diese Werte werden dynamisch zur Laufzeit des Programmes aktualisiert. Da nicht alle Zeilen auf einmal ausgegeben werden können ist diese Ansicht scrollbar gestaltet.



	Adresse	Bank0	Bank1
1	00	00	00
2	01	00	FF
3	02	00	00
4	03	18	18
5	04	00	00
6	05	00	FF
7	06	00	FF
8	07	00	00
9	08	00	00
10	09	00	00
11	0A	00	00
12	0B	00	00
13	0C	00	00
14	0D	00	00
15	0E	00	00
16	0F	00	00

Abbildung 4 - Speicheransicht

#### 3.1.4 Spezialregister

Am linken Rand sind die Werte einiger speziellen Register dargestellt. Dabei handelt es sich zum einen um die hexadezimalen Werte der Register PC (Programmzähler), W (Arbeitsregister), FSR (File Select Register), PCLATH, STATUS, TMR0, OPTION und INTCON. Zum anderen sieht man hier den Inhalt der 3 Status-Bits, die bei der Ausführung eines Befehles betroffen sein können. Zero-Bit, Carry-Bit und DigitCarry-Bit sind also je nach Programmablauf gesetzt (1) oder nicht gesetzt (0).

Spezialregister	
PC	00
W-Reg	00
Zero-Bit	0
Carry-Bit	0
DC-Bit	0
FSR	00
PCLATH	00
STATUS	18
TMR0	00
OPTION	00
INTCON	00

Abbildung 5 - Spezialregister

#### 3.1.5 Port A und Port B

In den Anzeigen RA und RB werden die Inhalte der Speicheradressen 05h und 06h noch einmal speziell visualisiert. Man erkennt anhand der Zeile „Tris“ ob es sich zurzeit beim dem entsprechenden Pin darunter um ein Eingangsbit oder ein Ausgangsbit handelt. Durch einen Klick auf den gewünschten Pin ändert dieser seinen Wert von 0 auf 1 bzw. von 1 auf 0.

RA		7	6	5	4	3	2	1	0
Tris	i	i	i	i	i	i	i	i	i
Pin	0	0	0	0	0	0	0	0	0

RB		7	6	5	4	3	2	1	0
Tris	i	i	i	i	i	i	i	i	i
Pin	0	0	0	0	0	0	0	0	0

Abbildung 6 - Portübersicht

#### 3.1.6 Stack und Laufzeit

Rechts oberhalb der Befehlsliste befinden sich noch die Anzeigen für den Stack auf dem die Rücksprungadressen gespeichert und angezeigt werden können sowie eine simple Anzeige für die bis zum aktuellen Zeitpunkt verstrichene Laufzeit des Programmes, berechnet anhand der Anzahl an abgewickelten Befehlszyklen.

Stack	Laufzeit
<div></div>	0 µs

Abbildung 7 – Laufzeitanzeige

## 4 Klassen

Zu Beginn des Projektes wurde die grundlegende Struktur des Programmes festgelegt und in entsprechende Klassen unterteilt. Diese Klassen sehen wie folgt aus:

- ❖ Alu: führt Befehle aus
- ❖ Bitoperationen: ändert bzw. prüft ein einzelnes Bit
- ❖ Codezeile: holt sich Befehlscode und Breakpoints
- ❖ Hexkonverter: wandelt einen int-Wert in einen Hex-Wert
- ❖ Interruptsteuerung: zuständig für das Interrupt-Handling
- ❖ Laufzeitzähler: zählt die Befehlszyklen
- ❖ MainWindow: alles was mit GUI-Ausgabe zu tun hat
- ❖ Parser: ermittelt Textzeile und Befehlscode
- ❖ Programmspeicher: lesen und schreiben von Speicherinhalten
- ❖ Programmzähler: zuständig für Program Counter (Startadresse, Inkrementierung)
- ❖ Ram: lesen und schreiben des RAM, Status-Bits ändern
- ❖ Runthread: lässt das Programm durchlaufen
- ❖ Stack: lesen und schreiben von Rücksprungadressen
- ❖ Steuerwerk: zentrale Schnittstelle, Befehlserkennung, Schrittsteuerung
- ❖ Timersteuerung: zuständig für TMR0 und Taktsteuerung
- ❖ W-Register: lesen und schreiben des Arbeitsregisters

## 5 Befehle

### 5.1 BTFSS (Bit Test f, Skip if Set )

**Parameter:** f, b

**Beeinflusste Flags:** -

**Befehl:** Wenn b = 1 ist, wird der nächste Befehl übersprungen.

**Implementierung:** Dafür muss aus dem aktuellen Befehl die Adresse von f extrahiert werden. Über diese kann auf den Inhalt von f und dann auf die entsprechende Stelle (b) in f zugegriffen werden. Falls diese Stelle „1“ ist, wird der „Program-Counter“ um eins erhöht.

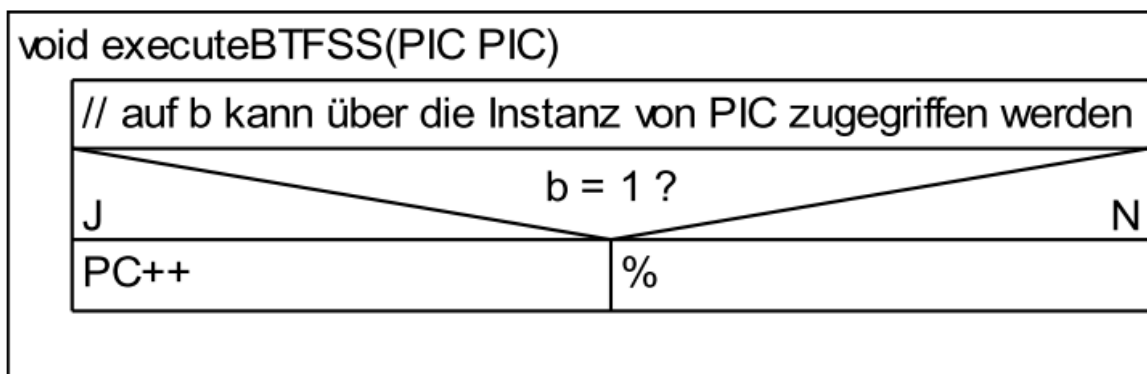


Abbildung 8 - Ablaufplan BTFSS

### 5.2 BTFSC (Bit Test f, Skip if Clear )

**Parameter:** f, b

**Beeinflusste Flags:** -

**Befehl:** Wenn b = 0 ist, wird der nächste Befehl übersprungen.

**Implementierung:** Dafür muss aus dem aktuellen Befehl die Adresse von f extrahiert werden. Über diese kann auf den Inhalt von f und dann auf die entsprechende Stelle (b) in f zugegriffen werden. Falls diese Stelle „0“ ist, wird der „Program-Counter“ um eins erhöht.

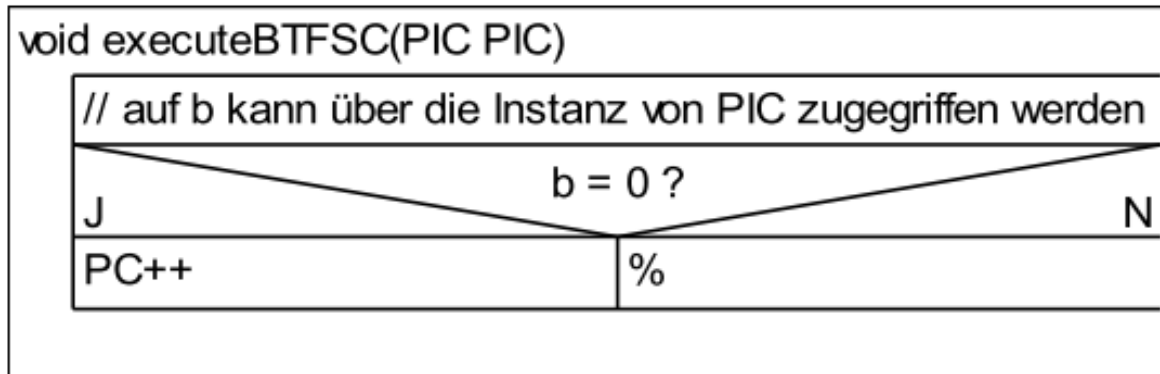


Abbildung 9 - Ablaufplan BTFSC

### 5.3 CALL (Call subroutine )

**Parameter:** k

**Beeinflusste Flags:** -

**Befehl:** Es wird ein Unterprogramm aufgerufen.

**Implementierung:** Dazu muss der aktuelle „Program-Counter“ auf den Stack abgelegt werden (unter Beachtung der Stack Grenzen und Erhöhung des Stack-Pointers). In den „Program-Counter“ muss die Start-Adresse des Unterprogramms geladen werden.

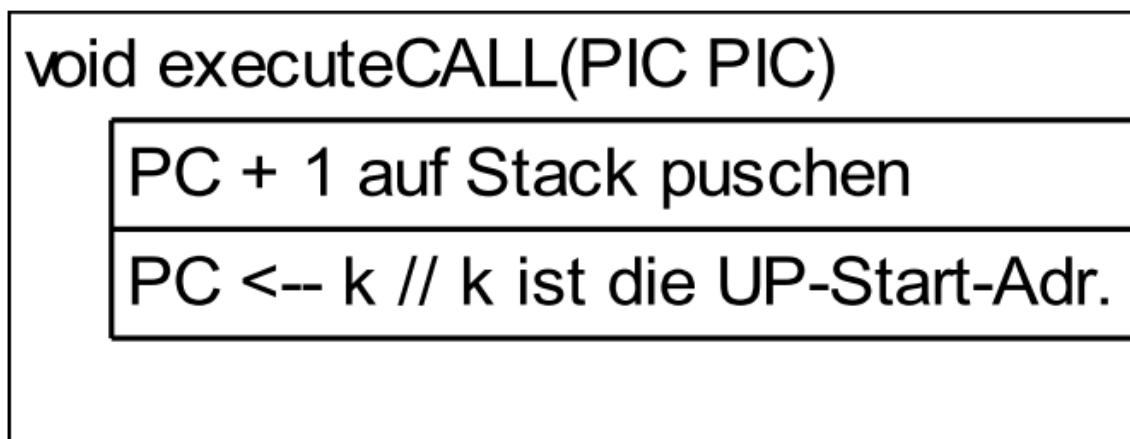


Abbildung 10 - Ablaufplan CALL

## 5.4 MOVF (Move f)

**Parameter:** f, d

**Beeinflusste Flags:** Zero

**Befehl:** Der Inhalt von f wird abhängig von d wieder nach f ( $d = 1$ ) bzw. nach w ( $d = 0$ ) geschrieben. Dabei wird das Zero-Flag gesetzt, falls in das Ziel der Wert „0“ geschrieben wird.

**Implementierung:** Es wird geprüft ob das Zero-Flag zu setzen ist und anschließend der Wert an die entsprechend Stelle geschrieben.

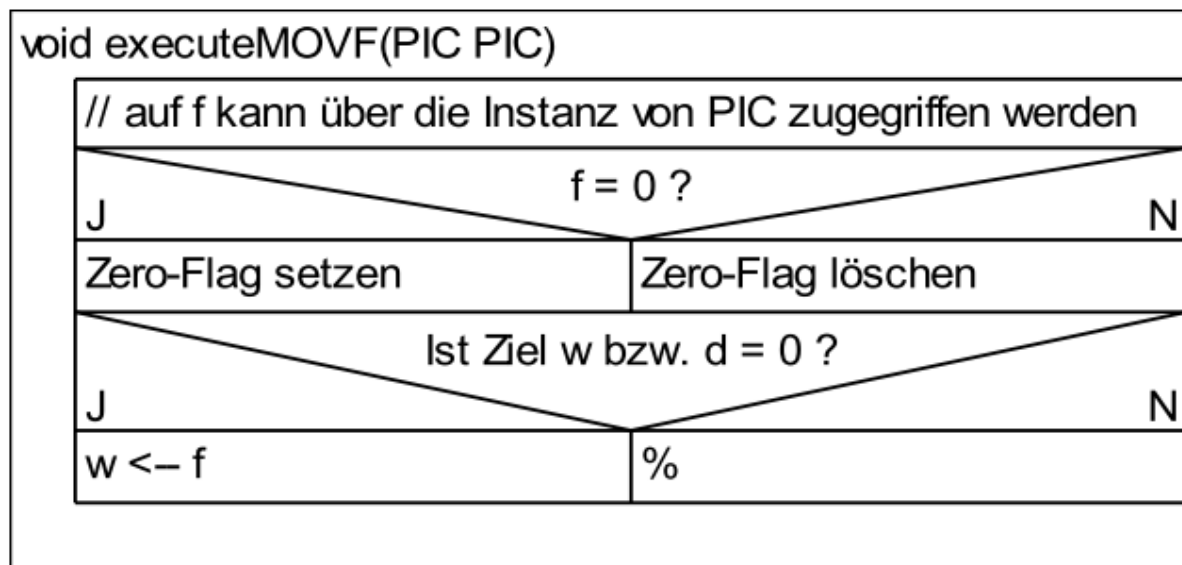


Abbildung 11 - Ablaufplan MOVF

## 5.6 RRF (Rotate Right f through Carry )

**Parameter:** f, d

**Beeinflusste Flags:** Carry

**Befehl:** F bildet mit dem „Carry-Flag“ ein „virtuelles 9 Bit Ringregister“ in dem jeweils um eine Stelle rotiert werden kann. Das Ergebnis der Rotation kann je nach Wert von d wieder nach f ( $d = 1$ ) bzw. nach w ( $d = 0$ ) geschrieben werden.

**Implementierung:** Um den Wert aus f manipulieren zu können, wird dieser zwischengespeichert und dann in der temporären Variable rotiert. Das „Carry-Flag“ wird an die niederwertigste Stelle der temporären Variable geschrieben und je nach dem welchen Wert d hat wird die temporäre Variable nach f ( $d = 1$ ) bzw. w ( $d = 0$ ) geschrieben.

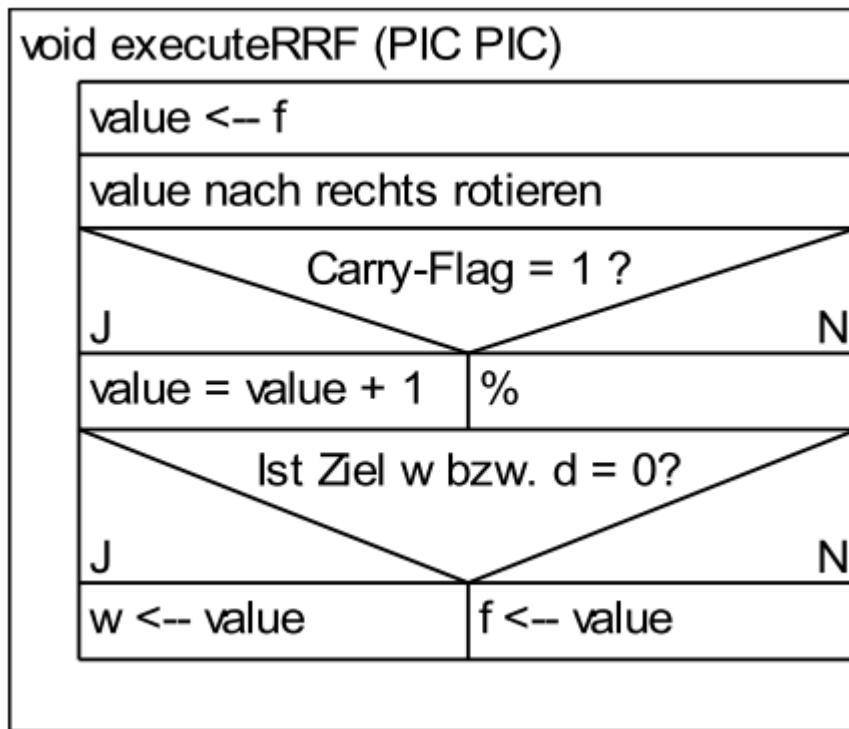


Abbildung 12 - Ablaufplan RRF

### 5.7SUBWF (Subtract W from f )

**Parameter:** f, d

**Beeinflusste Flags:** Carry, Digit-Carry, Zero

**Befehl:**  $f - w$ ; Das Ergebnis der Subtraktion wird je nach dem welcher Wert für d gesetzt wurde nach f ( $d = 1$ ) oder nach w ( $d = 0$ ) geschrieben. Das Ergebnis hat Auswirkungen auf das „Carry-Flag“, das „Digit-Carry-Flag“ und das „Zero-Flag“.

**Implementierung:** Das Ergebnis der Subtraktion wird in einer temporären Variable abgelegt. Dann wird geprüft ob und wenn ja welche Flags gesetzt werden müssen. Im Anschluss wird das temporär abgespeicherte Ergebnis in Abhängigkeit des Wertes von d wieder nach f ( $d = 1$ ) oder nach w ( $d = 0$ ) geschrieben.



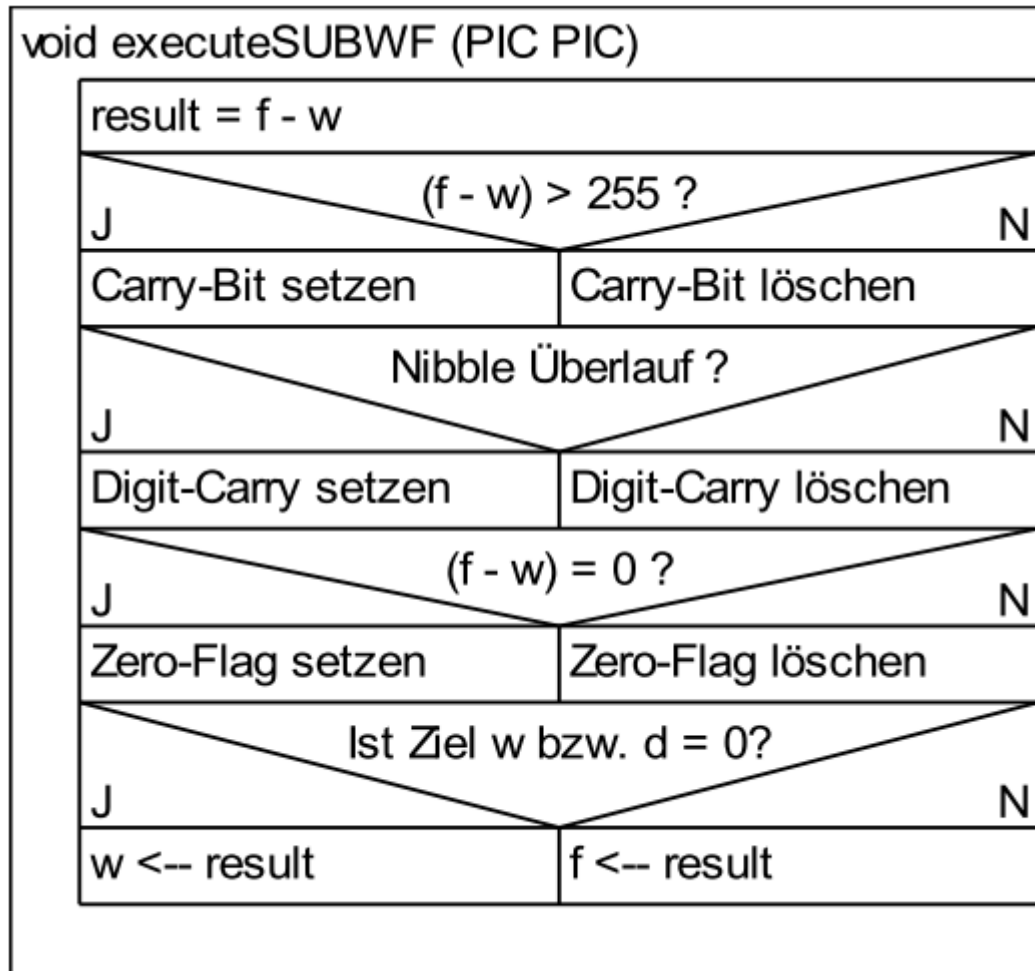


Abbildung 13 - Ablaufplan SUBWF

### 5.8 DECFSZ (Decrement f, Skip if 0 )

**Parameter:** f, d

**Beeinflusste Flags:** -

**Befehl:** F wird um eins dekrementiert. Das Ergebnis der Operation wird je nach definiertem Ziel zurück nach f (d = 1) bzw. nach w (d = 0) geschrieben. Wenn das Ergebnis „0“ sein sollte, wird der darauf folgende Befehl übersprungen. Implementierung: Der Wert aus f wird temporär zwischengespeichert. Diese temporäre Variable wird dekrementiert um anschließend, je nach definiertem Ziel wieder nach f (d = 1) bzw. nach w (d = 0) gespeichert zu werden. Wenn die temporäre Variable nach dem Dekrementieren „0“ sein sollte, wird der „Program-Counter“ inkrementiert.

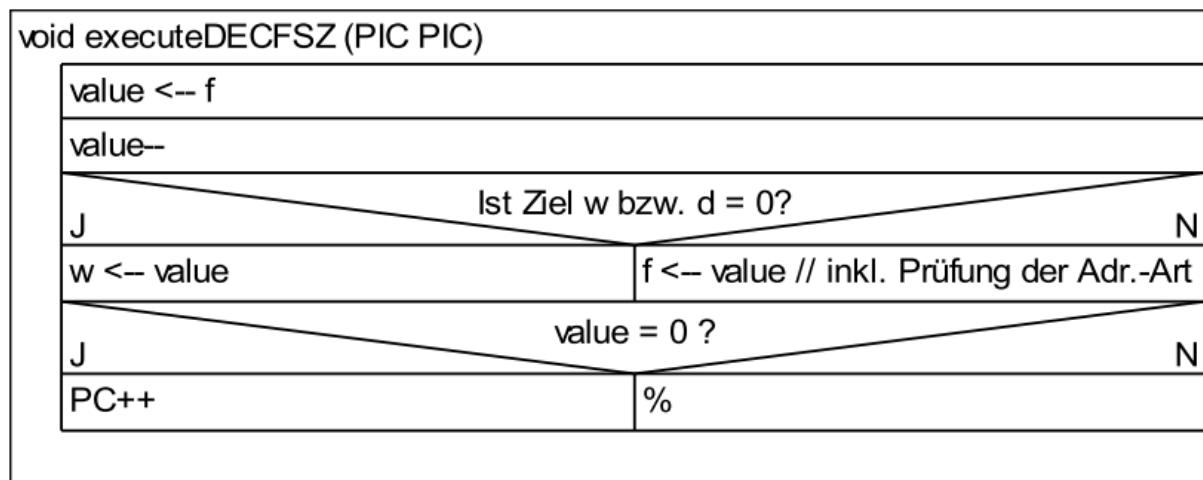


Abbildung 14 - Ablaufplan DECFSZ

## 5.9 XORLW (Exclusive OR literal with W )

**Parameter:** k

**Beeinflusste Flags:** Zero

**Befehl:** Eine konstante wird mit w XOR verknüpft, das Ergebnis wird nach w geschrieben. Sollte dieses „0“ sein, wird das „Zero-Flag“ gesetzt.

**Implementierung:** Das Ergebnis der XOR Verknüpfung von w und der Konstanten k wird nach w gespeichert. Je nachdem welchen Wert w annimmt, wird das Zero-Flag gesetzt.

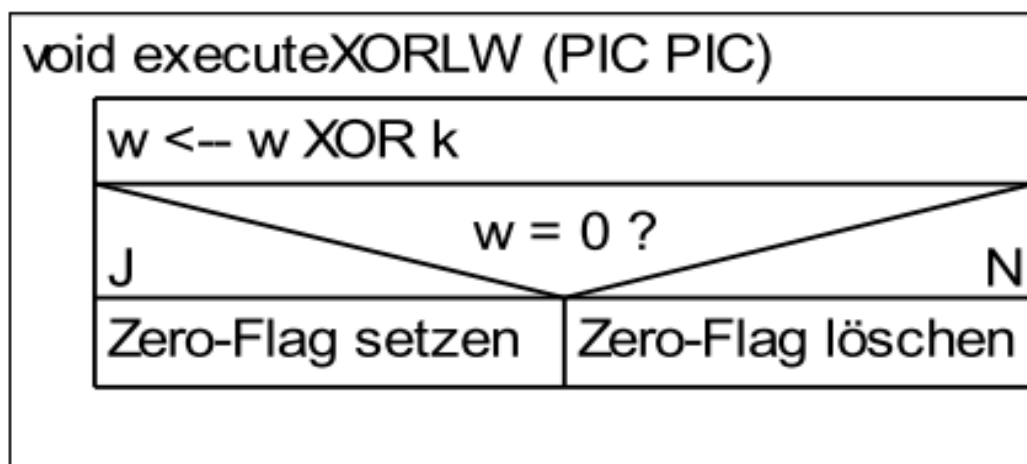


Abbildung 15 - Ablaufplan XORLW

## 6 Fazit

Dieses Projekt zur Simulation eines Mikrokontrollers verhilft zu einem deutlich besseren Verständnis der Funktionsweise eines solchen komplexen Bauteils. In diesem speziellen Fall des PIC16C84. Hierbei wurden erweiterte Erkenntnisse über Befehlserkennung und -

ausführung, Veränderung von Speicherinhalten oder auch Interrupthandling gewonnen.

Außerdem war es durchaus von Vorteil bereits erlangtes theoretisches Wissen aus vorangegangenen Semestern bei der Umsetzung verwenden zu können. Hilfreich waren dabei vor allem die Inhalte der Vorlesungen Rechnertechnik I, Digitaltechnik, Software Engineering I sowie Programmieren I und II. Dadurch konnte man das Wissen der erwähnten Fächer sinngemäß vertiefen und Erfahrung in komplexen, fächerübergreifenden Projekten sammeln.

Auffällig bei der Umsetzung war, dass das Potenzial eines objektorientierten Ansatzes wie ihn die Programmiersprache C++ nahelegt nicht unbedingt vollständig ausgeschöpft wurde, sondern die Methoden eher Schritt für Schritt implementiert wurden. Diese Methoden wurden dann im Nachhinein in separate Klassen verschoben um eine sinnvolle und übersichtliche Struktur zu garantieren.

Positiv zu erwähnen ist die Verteilung der Aufgaben innerhalb des Teams. Durch sinnvolle Aufteilung der Arbeiten und die Verwendung von Softwareentwicklungstools wie GitHub konnte meist effizient und produktiv gearbeitet werden. Die eigenen Fähigkeiten wurden durch den Beitrag an diesem komplexen Projekt nachhaltig verbessert.