PicSimulator Dokumentation

von Damian P. und Emil B.

Inhaltsverzeichnis

[Simulator 2](#_Toc485497779)

[Benutzeroberfläche 2](#_Toc485497780)

[Aufbau des Programmes 4](#_Toc485497781)

[Funktionen 4](#_Toc485497782)

[Code einlesen 4](#_Toc485497783)

[Befehl bearbeiten 5](#_Toc485497784)

[Breakpoints 6](#_Toc485497785)

[Interrupts 6](#_Toc485497786)

[RS232 6](#_Toc485497787)

[Befehle 6](#_Toc485497788)

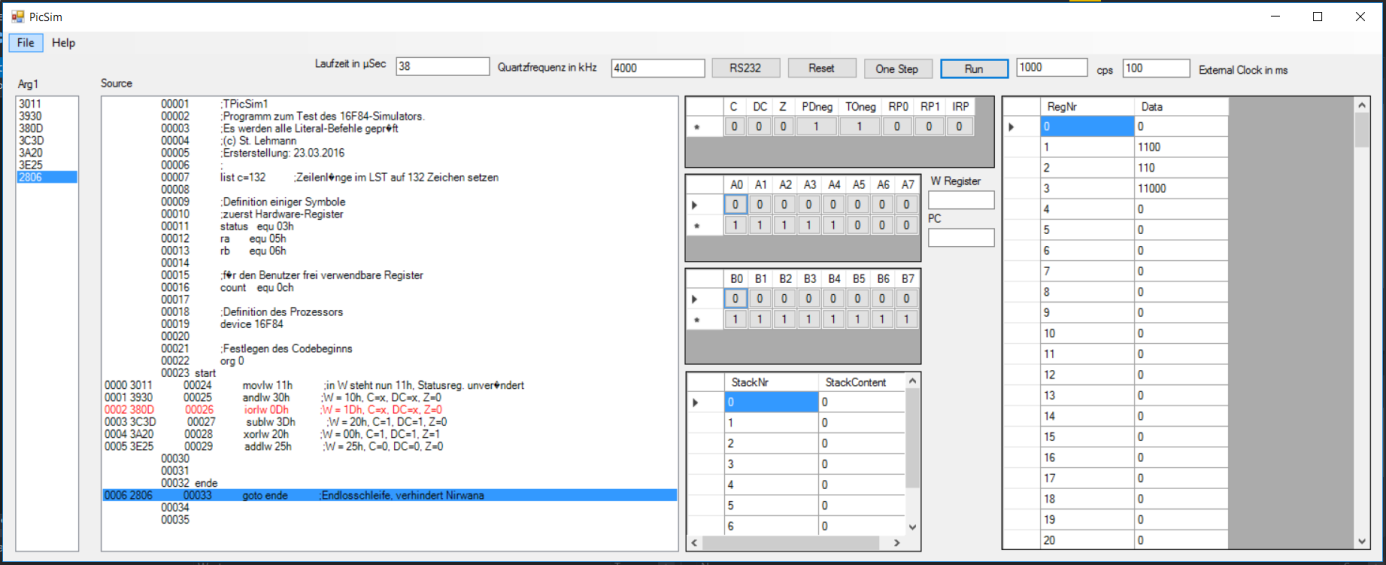
[ADDLW/SUBLW/ADDWF/SUBWF 6](#_Toc485497789)

[BTFSC/BTFSS 8](#_Toc485497790)

[ANDWF/ANDLW / IORWF/IORLW / XORWF/XORLW 9](#_Toc485497791)

# Simulator

# Benutzeroberfläche



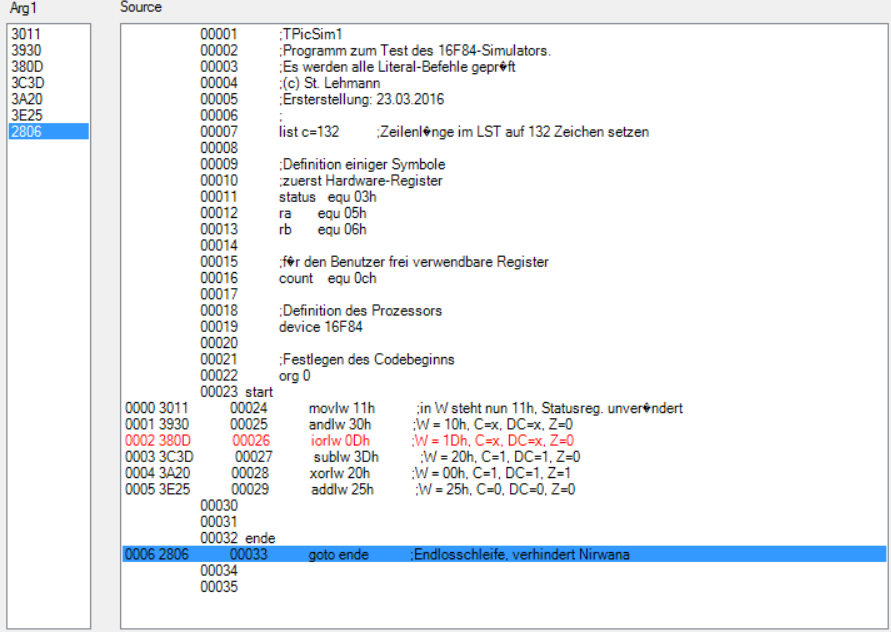
3

1

2

Die Benutzeroberfläche ist in 3 Bereiche unterteilt auf welche im Folgenden genauer eingegangen werden soll.

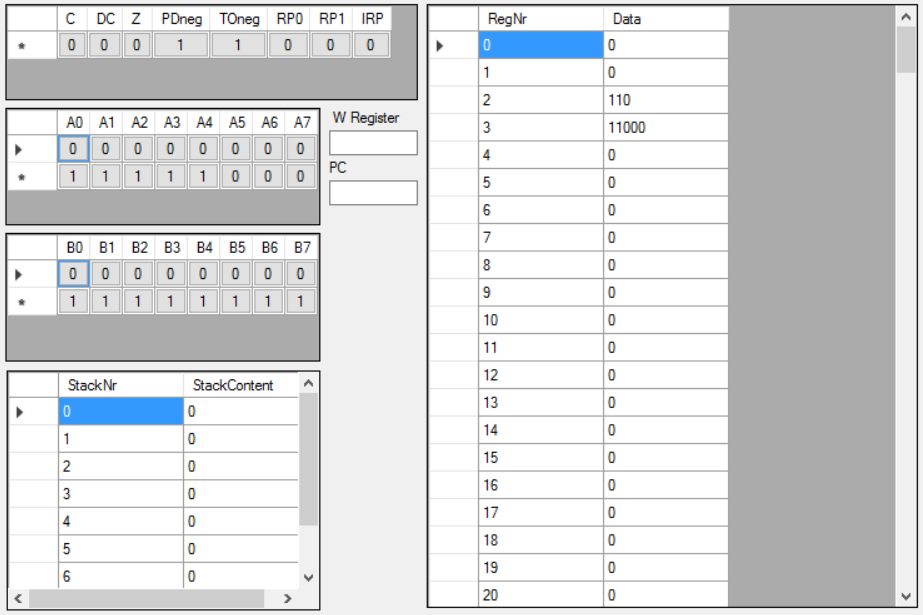
Der erste Bereich beinhaltet die Programminformationen. In 1 kann man die verschiedenen Befehle in ihrer Hexadezimalen Darstellung erkennen und in 2 ist der erweiterte Programmcode dargestellt. Außerdem erkennt man durch die blaue Markierung welcher Befehl der nächste ist welcher ausgeführt wird. Durch einen Doppelklick in 2 auf eine beliebige Zeile wird dort ein Breakpoint gesetzt, was man an der roten Einfärbung erkennen kann.



1

2

Im zweiten Bereich werden die Register und der Stack dargestellt. Im ersten Feld wird das Statusregister angezeigt und in den Feldern 1 und 2 werden Port A und B und darunter die Passenden Tris Werte angezeigt. Die Werte von Port A und B lassen sich mit einem Klick auf diese ändern. In Feld 4 wird der Stack angezeigt mit der NR und dem passenden Inhalt. In Feld 5 werden die Werte vom W Register und des PCs dargestellt und in Feld 6 sind alle Register in Binärschreibweise vorhanden.



6

5

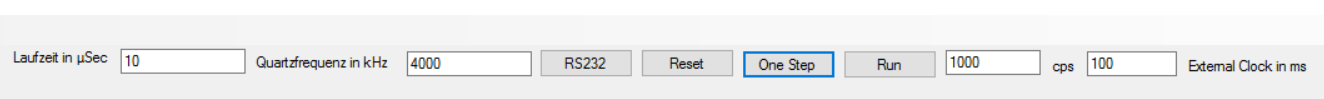
4

3

2

1

Im dritten Bereich sind die Interaktionsmöglichkeiten dargestellt welche für den Benutzer vorhanden sind. Im Feld 1 kann man die Quarzfrequenz festlegen und dort wird die Laufzeit basierend auf der Quarzfrequenz angezeigt. Feld 2 beinhaltet verschiedenen Buttons um ein Programm zu starten, es Schritt für Schritt durch zu gehen, es zu resetten oder die Schnittstelle RS232 anzusprechen. Im Feld 3 kann man die Geschwindigkeit einstellen mit welcher der Simulator Befehle ausführen soll und in Feld 4 kann man die Externe Clock einstellen.



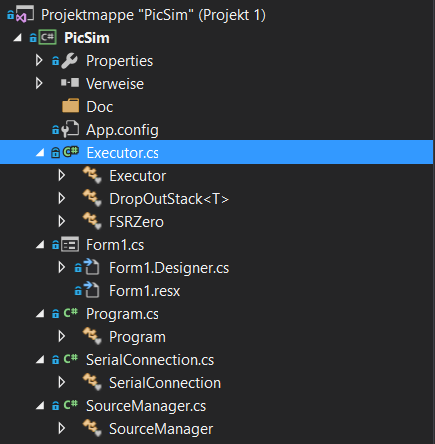
4

3

2

1

# Aufbau des Programmes

Das Programm ist in verschiedene Bestandteile unterteilt. Auf die genauere Funktionsweise soll im Anschluss eingegangen werden.

So ist die Form1 und Program Klasse dafür Zuständig die Benutzeroberfläche dar zu stellen.

Die SourceManager Klasse ist zuständig den Programmcode eines PIC Programmes ein zu lesen und diesen in ein Array um zu wandeln.

Die Executor Klasse wiederum beinhaltet die Logik des PIC Simulators, da hier alle Befehle und Routinen abgebildet sind.

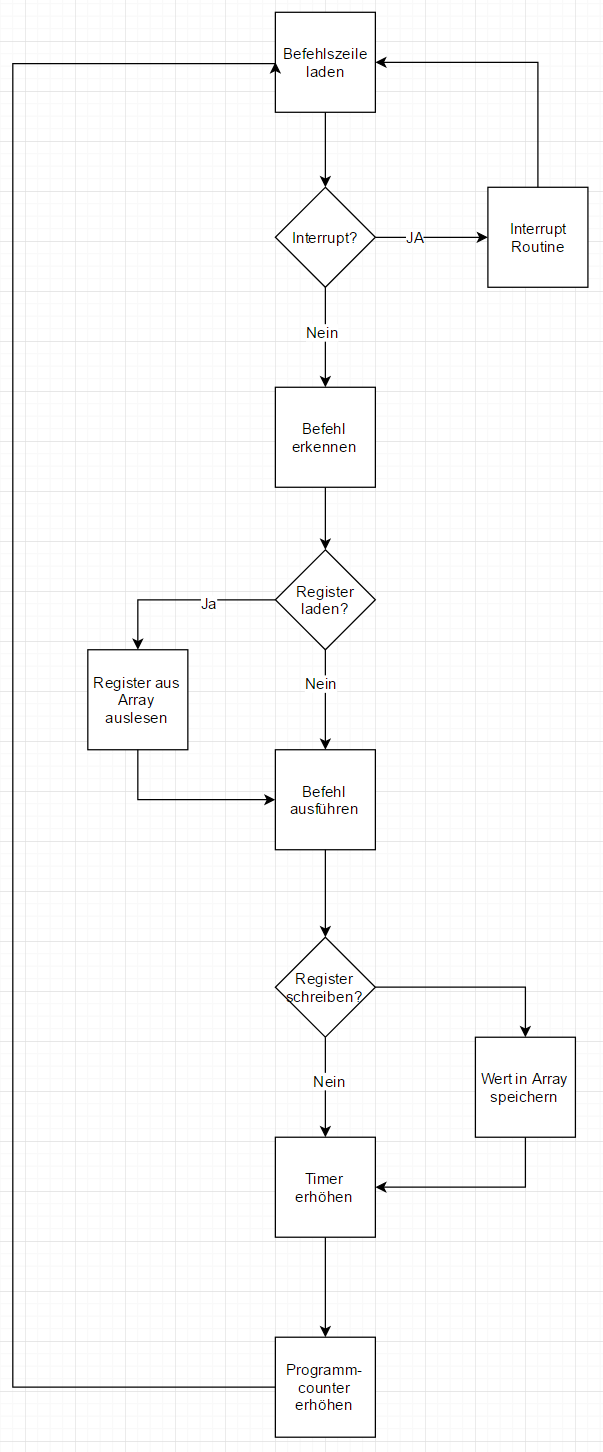
Und die SerialConnection Klasse ist zuständig für die Verbindung mit der RS232 Schnittstelle und das senden/empfangen der Daten über diese.

Das Programm wurde in C# in der Entwicklungsumgebung VisualStudio erstellt.

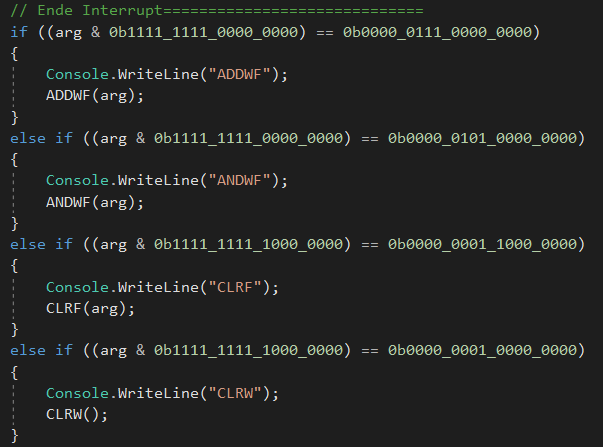
# Funktionen

## Code einlesen

## Befehl bearbeiten



Jeder Cycle und dadurch jeder Befehl durchläuft eine gewisse Prozedur welche bei jedem Befehl grundlegend gleich ist. Zuerst wird aus dem ProgramCounter der Aktuelle wert gelesen. Dann wird der Executor mit der passenden Programmzeile zum ProgrammCounter aufgerufen. Im Executor wird dann zuerst geprüft ob ein Interrupt stattgefunden hat und falls dies der Fall ist wird die Interrupt Routine ausgeführt. Wenn kein Interrupt stattgefunden hat wird im nächsten Schritt der Befehl erkannt. Dazu wird die aus dem Array stammende Zeile verundet und es wird geprüft um welchen Befehl es sich handelt, ein Beispiel zu ein paar Befehlen ist in der Abbildung zu sehen.



Wenn nun der Befehl erkannt wurde wird als nächstes der Befehlsspezifische Code ausgeführt welcher den Befehl darstellt. Dabei werden auch die Status Bits geprüft und eventuell gesetzt aber dies soll später bei den Befehlen erklärt werden. Wenn der Befehl das lesen oder beschreiben eines Registers beinhaltet wird dazu eine Methode aufgerufen welche anhand der Bank und der Adresse das passende Register aus dem Register Array ausliest oder beschreibt. Am Ende von jedem Befehl wird der Timer aufgerufen. Dabei übergibt der Befehl wie viele Cycels er gebraucht hat. Der Timer Checkt nun ob der Interne Timer aktiviert ist und wird dann abhängig von der Prescaler Einstellung erhöht. Nachdem der Timer erhöht wurde wird der Programmcounter nach jedem Befehl um 1 erhöht und die Routine läuft von vorne los.

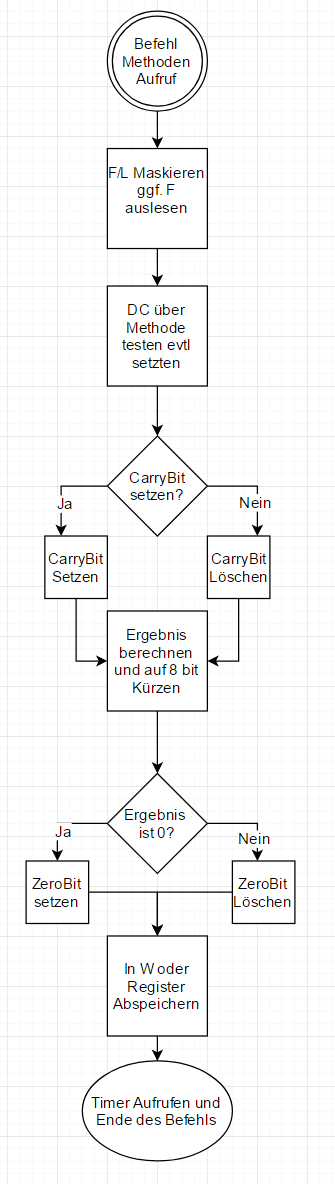
## Breakpoints

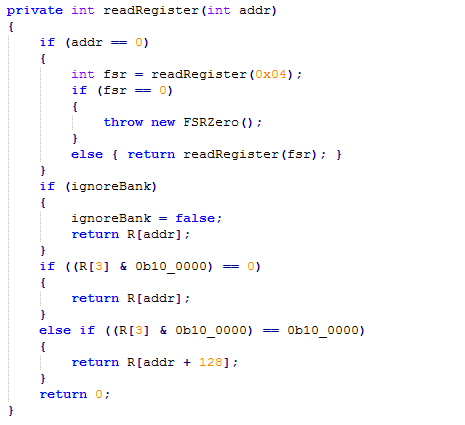
## Interrupts

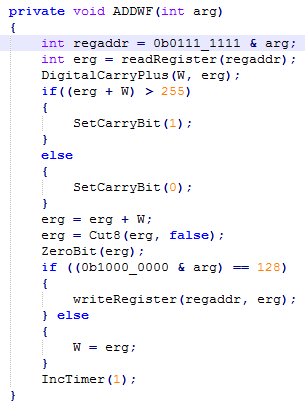
## RS232

# Befehle

## ADDLW/SUBLW/ADDWF/SUBWF

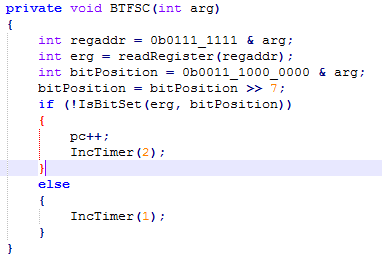


ADDLW, SUBLW und ADDWF, SUBWF sind sehr ähnlich aufgebaut, da es sich jeweils um eine einfache Rechenoperation handelt, welche jeweils alle drei StatusBits beeinflussen können. Zuerst wird aus dem Befehlscode die Zahl (L) oder das Register (F) heraus maskiert. Anschließend wird, wenn nötig der wert von F aus dem Array, durch die Funktion readRegister, ausgelesen. readRegister ist eine eigene Methode auf die mehrere Befehle zugreifen. In dieser Methode werden Ausnahmen wie die Indirekte Adressierung und die Bank berücksichtigt. Sobald L oder F ausgelesen wurden wird der aktuelle Befehl auf das DigitalCarry geprüft. Dazu werden einer Methode die beiden Operanten der Berechnung übergeben und diese prüft darauf ob das DigitalCarry gesetzt werden soll. Anschließend wird geprüft ob das CarryBit gesetzt oder gelöscht werden muss, wobei das eigentliche setzten oder löschen über eine andere Methode stattfindet. Darauf wird das Ergebnis in eine Variable gespeichert und es wird durch eine weitere Funktion auf ein Byte gekürzt, dass die PIC Register jeweils ein Byte speichern können. Danach wird das Ergebnis der Funktion einer Methode übergeben, welche überprüft ob das Ergebnis null ist und wenn dies der Fall ist das ZeroBit setzt bzw. löscht. Abschließend wird, wenn nötig, geprüft in welches Register das Ergebnis gespeichert werden soll und die Timerfunktion wird mit den passenden Cycle werten aufgerufen.



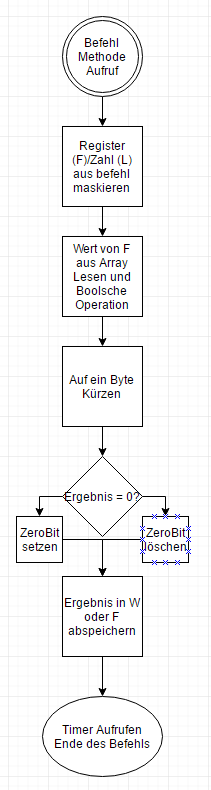
## BTFSC/BTFSS

Die Methoden der Befehle BTFSC und BTFSS beginnen mit dem Maskieren des Registers (F) und dem Auslesen des Wertes aus dem Array zu F. Anschließend wird die Bit Position aus dem

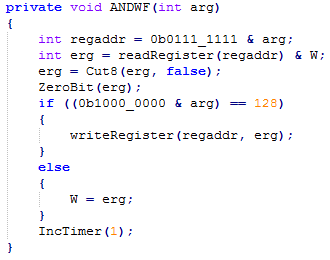


Befehl heraus maskiert und bestimmt. Mit der Information des Wertes von F und der Bit Position wird eine Methode aufgerufen, welche prüft ob das Bit an der Position gesetzt ist. Dann wird abhängig vom Ergebnis der ProgrammCounter um eins erhöht und der Timer mit dem Wert zwei aufgerufen oder nur der Timer mit dem Wert eins aufgerufen.

## ANDWF/ANDLW / IORWF/IORLW / XORWF/XORLW

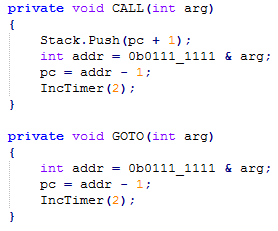


Zuerst wird das Register (F) bzw. die Zahl (L) aus. Daraufhin wird die Boolsche Operation mit W und dem Register bzw der Zahl ausgeführt. Um die Länge von einem Byte zu sichern wird danach das Ergebnis auf acht Bit gekürzt. Anschließend wird das Ergebnis der Operation darauf geprüft ob es 0 ist und abhängig davon wird das ZeroBit gesetzt bzw. gelöscht. Abschließend wird geprüft ob wohin das Ergebnis gespeichert werden soll und die Timer Methode wird mit dem Wert Eins aufgerufen.

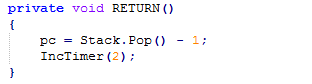


## CALL/GOTO

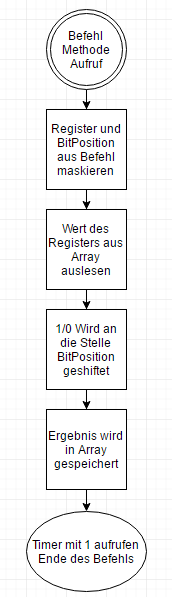
Der GOTO und der CALL Befehl sind sehr ähnlich. Bei beiden wird zuerst der neue Wert für den ProgramCounter aus dem Befehl maskiert. Bei der CALL Funktion wird jedoch zuerst der nächste ProgrammCounter Wert auf den Stack gepusht. Dazu wird eine Methode verwendet, welche einen Stack darstellt welcher die Funktionen Pushen und Pullen beinhaltet. Anschließend wird der ProgramCounter auf den vorher ermittelten Wert minus Eins gesetzt, da er noch einmal am Ende eines jeden Befehles erhöht wird. Abschließend wird die Timer Methode mit dem Wert Zwei aufgerufen.



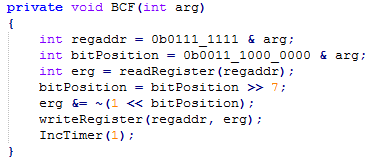
Bei einem RETURN wird der Eintrag aus dem Stack gepullt. Daraufhin wird der ProgramCounter mit diesem Wert minus Eins überschrieben und der Timer wird mit dem Wert Zwei aufgerufen. Bei einem RETFIE wird zusätzlich noch das Globale Interrupt Enable Bit im INTCON Register gesetzt, bei einem RETLW wird die Zahl aus dem Befehl maskiert und anschließend in W gespeichert.

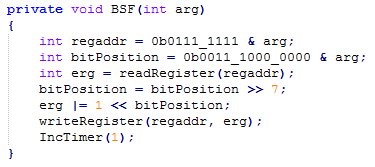


## BCF/BSF

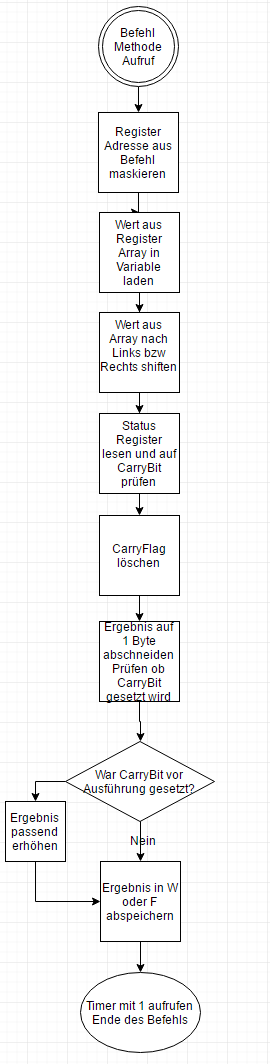


Zuerst wird die BitPosition und die Register(F) Adresse aus dem Befehl maskiert und in Variablen gespeichert. Anschließend wird der Wert des Registers ausgelesen und die BitPosition wird aus dem Gespeichertem Wert bestimmt. Danach wird eine Eins oder eine Null an die Bit Stelle des Wertes von F geshiftet, welche durch die BitPosition gegeben ist. Abschließend wird das Ergebnis wieder in das Register Array geschrieben und die Timer Methode wird mit dem Wert Eins aufgerufen.

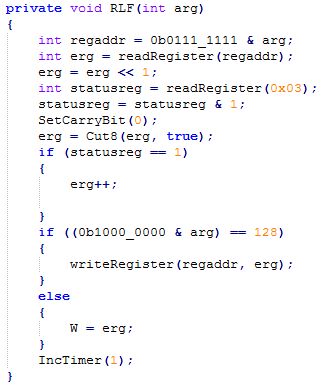




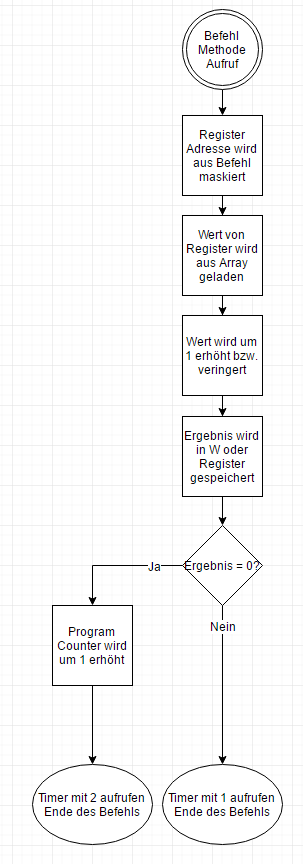
## RLF/RRF



RLF und RRF beginnen damit, dass die Register (F) Adresse aus dem Befehl maskiert wird. Daraufhin wird der Wert von F aus dem Array in eine Variable geladen. Danach wird dieser Wert nach rechts bzw. nach links geshiftet und es wird geprüft ob das CarryBit gesetzt ist. Dazu muss jedoch zuerst das OPTION Register aus dem Array gelesen werden. Die Information über das CarryBit wird gespeichert und es wird auf Null gesetzt. Anschließend wird das Ergebnis auf ein Byte gekürzt und es wird geprüft ob das CarryBit neu gesetzt werden muss. Daraufhin wird abgefragt ob das CarryBit vor dieser Operation gesetzt war und gegebenen Falls wird das Ergebnis so modfiziert, als wäre eine Eins rein geshiftet worden. Zum Schluss wird geprüft ob das Ergebnis in W oder in F gespeichert werden soll und danach wird die Timer Methode mit dem Wert Eins aufgerufen.



## DECFSZ/INCFSZ



Am Anfang des Befehls wird die Registeradresse aus dem Befehl maskiert und der Wert des Registers (F) aus dem Array in eine Variable gespeichert. Anschließend wird dieser Wert um Eins erhöht oder verringert. Dann wird geprüft ob das Ergebnis in W oder in F gespeichert werden soll. An diesem Punkt würden die Befehle INCF bzw. DECF den Timer aufrufen und wären fertig. Bei DECFSC und INCFSC wird jedoch noch geprüft ob das Ergebnis Null ist. Wenn dies der Fall ist wird der ProgramCounter um Eins erhöht und der Timer wird mit dem Wert Zwei aufgerufen. Falls nicht wird lediglich der Timer mit dem Wert Eins aufgerufen.

