Inhaltsverzeichnis

[Abbildungsverzeichnis 2](#_Toc484465486)

[1 Einleitung 3](#_Toc484465487)

[1.1 Was ist Simulation 3](#_Toc484465488)

[1.2 Vor- und Nachteile einer Simulation 3](#_Toc484465489)

[1.3 Mikrocontroller PIC16F84A 4](#_Toc484465490)

[2 Werkzeugen 5](#_Toc484465491)

[2.1 Entwicklungsumgebung IntelliJ 5](#_Toc484465492)

[2.2 GUI – Javafx 5](#_Toc484465493)

[2.3 Versionsverwaltung Githup 5](#_Toc484465494)

[3 Programmstruktur und Aufbau 6](#_Toc484465495)

[3.1 Benutzeroberfläche 6](#_Toc484465496)

[3.2 Programmstruktur 6](#_Toc484465497)

[3.3 Variablen 6](#_Toc484465498)

[3.4 Beschreibung einiger Befehle 6](#_Toc484465499)

[3.5 Interrupfunktion 6](#_Toc484465500)

[4 Zusammenfassung 7](#_Toc484465501)

[4.1 Umsetzung 7](#_Toc484465502)

[4.2 Fazit 7](#_Toc484465503)

# Abbildungsverzeichnis

# Einleitung

Die Zielsetzung des unseren Projektes ist es einen Simulator für einen PIC16F84A Mikrocontroller zu entwickeln. Dabei sollen verschieden Debug-Funktionen des Mikrocontrollers ebenfalls implementiert werden.

## Was ist Simulation

Simulation ist das Nachbilden eines dynamischen Prozesses in einem System mit Hilfe eines experimentierfähigen Modells, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Eine Simulation findet meistens nicht in Echtzeit statt (wie z.B. bei einer Emulation) sondern wird zu analytischen Zwecken langsamer als in der Realität nachgebildet.

## Vor- und Nachteile einer Simulation

Vorteile einer Simulation sind meist offensichtlich:

* Viele reale Systeme können mit analytischen Modellen nicht genau genug beschrieben werden, während in einem Simulationsmodell die realen Abläufe sehr gut nachgebildet werden können.
* Simulation bietet die Möglichkeit, Systeme in ganz unterschiedlichen Umgebungen und unter unterschiedlichen Bedingungen zu analysieren.
* Produktentwicklung und -optimierung erfolgen im Simulationsmodell. Versuche können verkürzt oder vermieden werden.
* Simulation erlaubt es Systeme auch in sehr kurzen oder sehr langen Zeitintervallen zu beobachten, wenn der Simulator die Zeit entsprechend streckt oder staucht.
* Durch Simulationen können Sie die Konsequenzen Ihrer Entscheidungen vor deren Realisierung analysieren. Problem können frühzeitig erkannt und vermieden werden.
* Funktion und Vorteile Ihrer technischen Systeme können mit Hilfe von Simulationen detailliert und optisch ansprechend demonstriert werden.

Doch eher seltener werdend die Nachteile einer Simulation beachtet:

* Bei der meisten Simulationsmodelle Resultate können nur geschätzt werden. Die dazu verwendeten Methoden basieren fast immer auf zwar plausiblen aber nicht beweisbaren Annahmen, so dass für einzelne Modelle falsche Resultate ermittelt werden.
* Ein hoher Datenbedarf. In der Simulation bedeutet dies oft, dass zwar detaillierte Modell erstellt werden, die benötigten Parameter aber nur auf Grund von sehr wenigen Beobachtungen geschätzt werden und in manchen Fällen nur auf Grund von nicht validierten Annahmen gesetzt werden.
* Simulationsmodelle sind aufwändig und teuer in der Entwicklung, deswegen meist durch begrenzte Ressourcen eingeschränkt. Oftmals wird deswegen nur ein vereinfachtes Modell der Wirklichkeit eingesetzt.

## Mikrocontroller PIC16F84A

Ein Mikrocontroller ist eine Art Mikrorechnersystem, bei welchem neben ROM und RAM auch Peripherieeinheiten wie Schnittstellen, Timer und Bussysteme auf einem einzigen Chip integriert sind.

Die Hauptanwendungsgebiete sind die Steuerungs-, Mess- und Regelungstechnik, sowie die Kommunikationstechnik und die Bildverarbeitung. Mikrocontroller sind in der Regel in Embedded Systems, in die Anwendung eingebettete Systeme, und somit in der Regel von außen nicht sichtbar. Ebenso verfügen sie, im Gegensatz zum PC, nicht über eine direkte Bedien- und Programmierschnittstelle zum Benutzer. Sie werden in der Regel einmal programmiert und installiert.

Der Pic16F84 ist ein 8bit Mikrocontroller mit Reduced-Instruction-Set-Computing-Architektur (RISC). Dadurch wird auf komplexe Befehle verzichtet und es kann mit allen Befehlen auf alle Register zugegriffen werden. Durch die Harvard Architektur des Pic16 sind Daten und Programmbefehle in zwei verschiedenen Registern, was den Programmbefehlen ermöglicht 14-Bit groß zu sein, während der separate Datenbus nur 8-Bit groß ist. Ein großer Vorteil dieser Architektur ist, dass fast alle Anweisungen des Pic16 nur einen Instruction Cycle benötigen. Der Pic16F besitzt einen Stack mit Speicherplatz für acht Adressen und zwei externe sowie zwei interne Interruptquellen. Darüber hinaus besitzt der Pic16F ein großes Register, welches in zwei Bänke unterteilt ist. Das Umschalten der Bänke erfolgt im Programmcode. Die Speicherbereiche können auch direkt über ihre Registeradresse angesprochen werden.

# Werkzeugen

## Entwicklungsumgebung IntelliJ

Da wir uns für die Sprache Java entschieden haben, wäre für uns die beste Lösung IntelliJ als Entwicklungsumgebung zu nehmen.

**IntelliJ IDEA** ist eine freie Entwicklungsumgebung (IDE) für Java, Kotlin, Groovy und Scala des tschechischen Unternehmens Jetbrains. Neben einer kostenpflichtigen und proprietären Ultimate-Edition gibt es auch eine Community-Edition, die unter einer Open-Source-Lizenz verfügbar ist. Die IDE kann durch zahlreiche Plugins erweitert werden. Intellij IDEA bietet einige, teils sehr fortschrittliche Funktionen wie Syntax-Highlighting, Code-Vervollständigung oder Refactoring, um das Programmieren produktiver zu gestalten. Zu den Features gehören die Unterstützung von Java EE, Ant, Apache Maven, JUnit, ein GUI-Editor, Tools zur Versionskontrolle, insbesondere Git, sowie hauptsächlich verschiedene Möglichkeiten zum automatischen Erstellen und Modifizieren von Code. Wie die meisten Entwicklungsumgebungen bietet IntelliJ eine Vielzahl, dem Entwickler nützlicher, Funktionen.

## GUI – JavaFX. UI-Designer Scene Builder.

Die Benutzeroberfläche vom Simulator wurde mit dem Framework JavaFX erstellt. **JavaFX** ist ein Framework zur Erstellung plattformübergreifender Java-Applikationen. Es ist eine Java-Spezifikation von Oracle und setzt sich zum Ziel, das professionelle Erstellen und Verteilen von interaktiven, multimedialen Inhalten und grafischen Benutzeroberflächen (GUIs) über sämtliche Java-Plattformen hinweg zu erleichtern.

Java Application muss man nicht mit der Hand bauen. Wichtig ist es jedoch zu wissen, dass die grundsätzliche Möglichkeit dazu besteht, d.h. das gesamte UI kann im Java-Code erzeugt werden. Zur Verfügung steht auch ein recht komfortabler UI-Designer, namens Scene Builder. Dieser kann direkt in die Entwicklungsumgebung (IntelliJ), eingebunden werden *(*JavaFX Application*(*Abbildung 1*).*

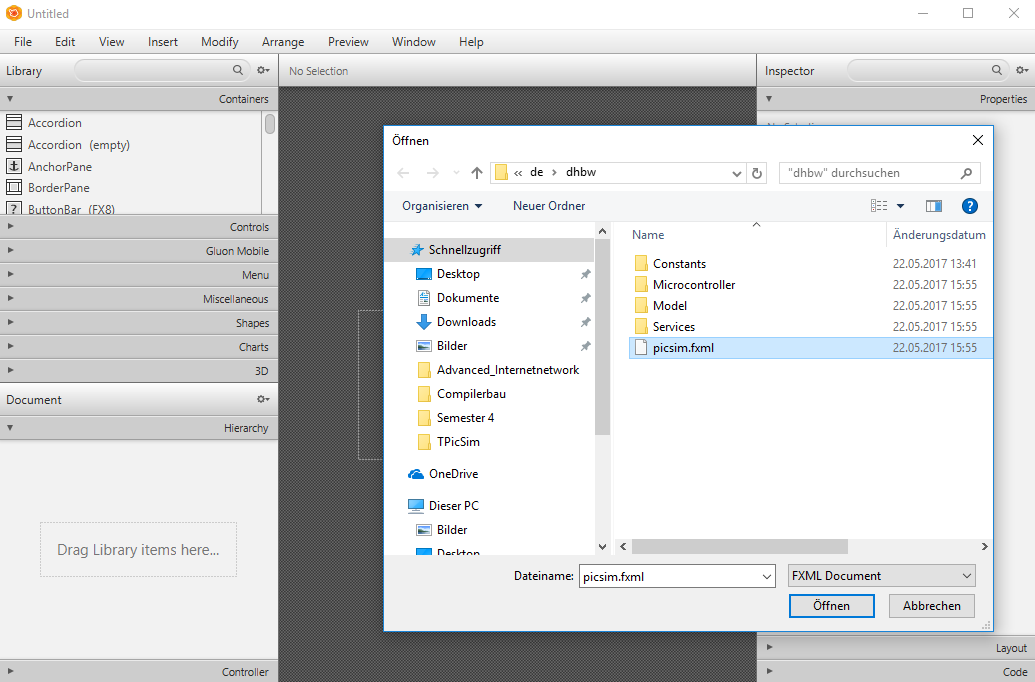


Abbildung 1. JavaFX Application.

## Versionsverwaltung Githup

Git ist eine freie Software zur verteilten Versionsverwaltung von Dateien, die ursprünglich für die Quellcode-Verwaltung des Linux-Kernels entwickelt wurde. GitHub kann mit den meisten Funktionen kostenlos genutzt werden, wenn der Quellcode öffentlich gemacht wird.  Im Kern bietet GitHub Speicherplatz im Internet an, der es ermöglicht, per git verwalteten Quellcode zu teilen. Erst damit ist es möglich, dass der Quellcode und seine Historie für mehrere Entwickler zugänglich ist und es bei der Bearbeitung nicht zur Überschneidung von Quellcode kommt, der später nicht mehr manuell auseinander dividiert werden kann. Über einen Push Befehl kann das Remote Repository mit dem lokalem Repository überschrieben werden. Wird ein Fetch Befehl ausgeführt wird, dass Remote Repository mit dem lokalem Repository verglichen und

zusammengeführt (merge Befehl) werden. Im Projekt wurde GitHub, ein webbasierter Hosting-Dienst für Software-Entwicklungsprojekte verwendet.

# Programmstruktur und Aufbau

## Benutzeroberfläche

Beim Starten der Applikation öffnet sich das User Interface des Simulators. Über diese Oberfläche ist es dem Benutzer möglich die Simulation zu steuern.

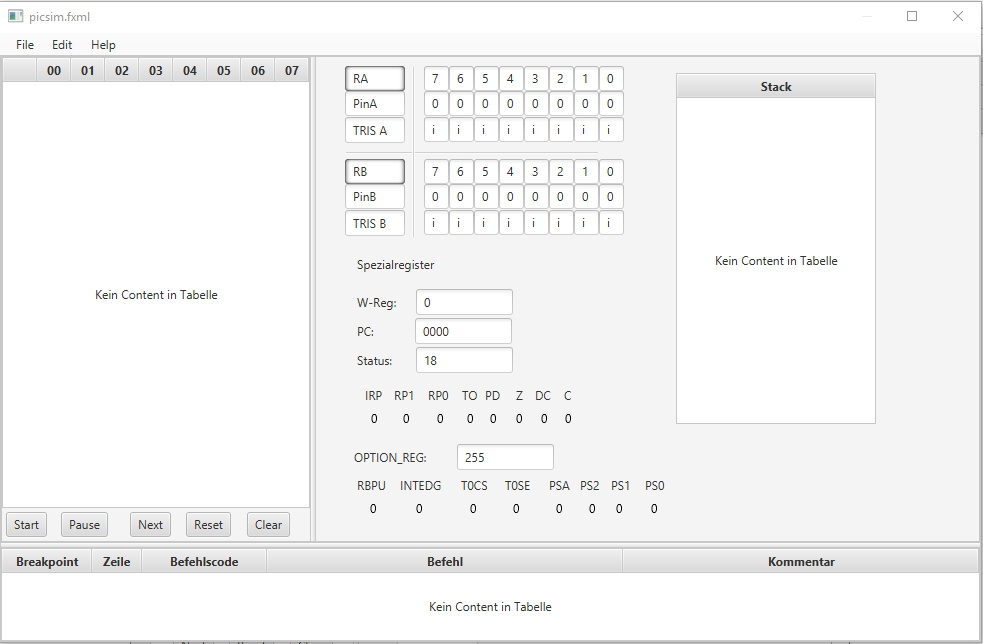


Abbildung 2. GUI

Links oben, über den Menü-Eintrag File> "Open" kann man eine zuvor fertig compilierte **\*.lst** Datei in den Simulator laden.

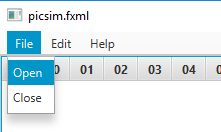


Abbildung 3. Datei laden

Nach dem man die Datei öffnet, wird der Programmcode der \*.lst als Liste angezeigt.

Der Punkt, an welchem sich das Programm befindet, wird blau markiert.

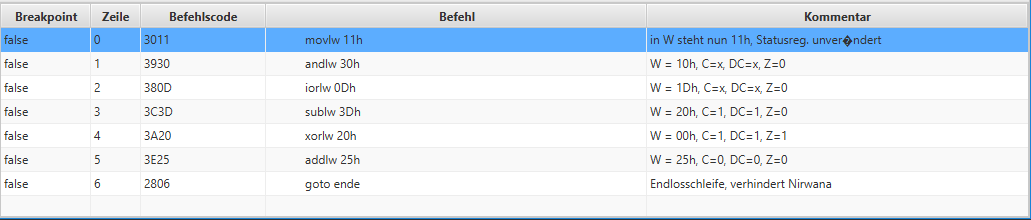


Abbildung 4. Visualisieren des Programmcodes

Mit den Buttons Start, Pause, Next, Reset, Cear lässt sich der Simulationsablauf steuern. Mit Next lässt sich jeweils nur ein Programmschritt ausführen, mit Start wird das Programm komplett abgearbeitet. Mit Pause stoppt der Simulationsvorgang, mit Reset wird er komplett zurückgesetzt und mit Clear wird der Programmcode aus der Content- Tabelle gelöscht.



Abbildung 5. Kontroll-Buttons

In der Mitte des User Interfaces werden die Register angezeigt. Die Werte des Spezialfunktionsregisters werden im Hexadezimalsystem darunter dargestellt. Die Speicherinhalte werden in Form einer Tabelle am oberen Rand des User Interfaces dargestellt. Die Werte werden im Hexadezimalsystem wiedergegeben.

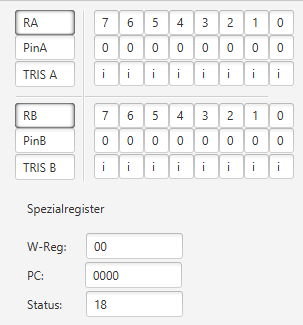


Abbildung 6. Register und Spezialfunktionsregister

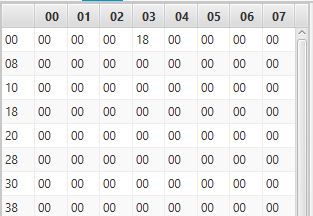


Abbildung 7. Die Speicherinhalte

## Programmstruktur/ Programmstruktur

## Beschreibung einiger Befehle

Bei den byteorientierten File Register Befehlen handelt es sich meist um Operationen zwischen einem Fileregister und dem Akkumulator. Das Ergebnis wird entweder in das W-Register(d = 0) oder ins Fileregister (d = 1) geschrieben. Bei Addition und Subtraktion werden das Carryflag, DigitCarry und Zerobit gesetzt/bzw gecleart. Die meisten Operationen überprüfen jedoch lediglich ob das Ergebnis 0 ist.

**MOVF**

Der Befehl MOVF schreibt den Wert des bestimmten Registers entweder in sich selbst,

wenn d = 1 oder in das W-Register, wenn d = 0. Der Befehl ist z.B. dann sinnvoll wenn

nach Setzen des Zerobits das Register getestet werden soll. Für diesen Befehl wird eine

Hilfsvariable benötigt, da das Ergebnis entweder in W oder im Register selbst geschrieben

wird.

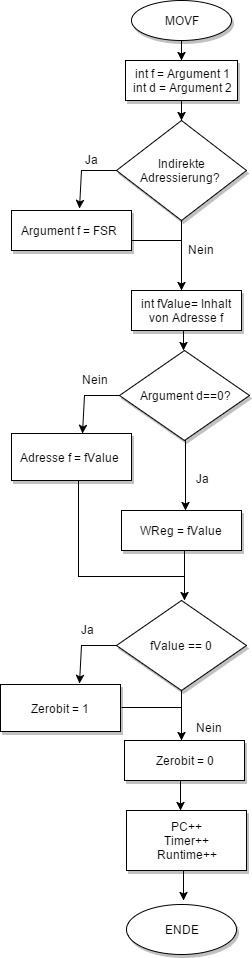


Abbildung 8. MOVF

|  |
| --- |
| public class MOVF extends Instruction {  public MOVF(int instruction, int opcode, int argument1, int argument2){  super(instruction, opcode, argument1, argument2);  }   @Override  public void execute(){  copyFormerValues();   int d = argument1;  int f = argument2;   // Indirekte Addressierung  if(f == Const.*IND*)  f = memory.getAddress(Const.*FSR*);  int fValue = memory.getAddress(f);   if (d == 0)  memory.setRegisterW(fValue);  else  memory.setAddress(f, fValue);   // ZeroBit prüfen  if(fValue == 0)  setZeroFlag();  else  clearZeroFlag();   incrementProgramCounter();  incrementRuntime();  incrementTimer0();  copyCurrentValues();  compareValues();  }   @Override  public void displayDebugInfo(){  System.*out*.println(String.*format*("%04X", instruction) + ": MOVF" + " d: " + argument1 + "," + " f: " + argument2);  } |

**SUBWF**

Der Befehl SUBWF subtrahiert von einem Fileregister den Inhalt des W-Registers. Das

Ergebnis wird entweder ins W-Register oder ins Fileregister geschrieben. Das Besondere

an SUBWF: Das Carryflag wird hier gesetzt wenn das Ergebnis 0 ist oder wenn es keinen überlauf gibt! Auch hier wird eine Hilfsvariable benötigt.

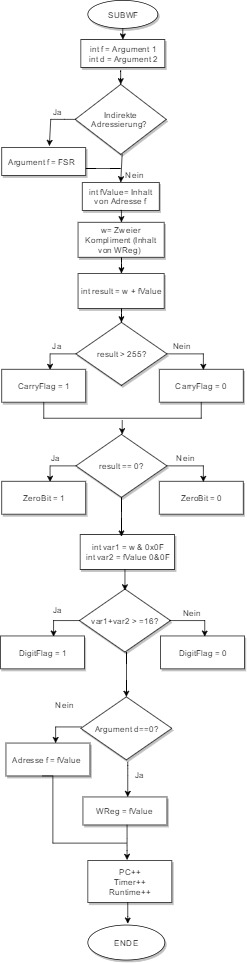
****

Abbildung 9. SUBWF

**DECFSZ**

Dieser Befehl dekrementiert den Inhalt des bestimmten Fileregisters und schreibt das

Ergebnis entweder ins Fileregister oder in W. Wird die 0 dekrementiert so ist das Ergebnis

255. Ist das Ergebnis 0 wird ein NOP ausgeführt und der nächste Befehl übersprungen.

Ansonsten wird der nächste Befehl ausgeführt.

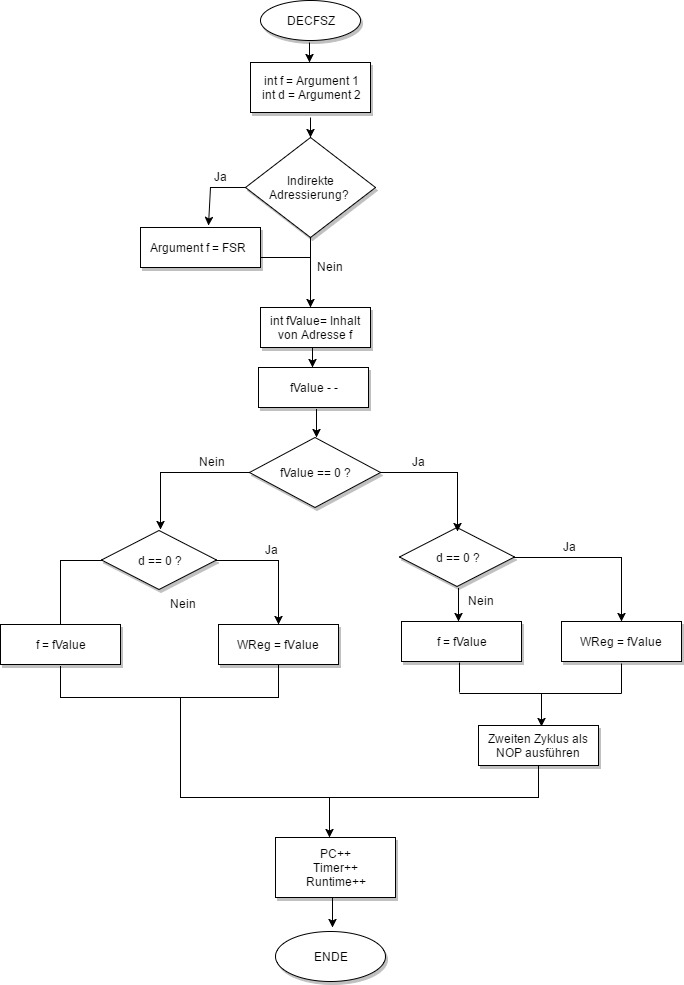
****

Abbildung 10. DECFSZ

**RRF**

Die Rotate Right Funktion shiftet ein den inhalt eines Registers um eine Stelle nach

rechts. Dies entspricht einer Ganzzahldivision durch 2. Das Ergebnis wird abhängig von d entweder in W oder ins Fileregister geschrieben.

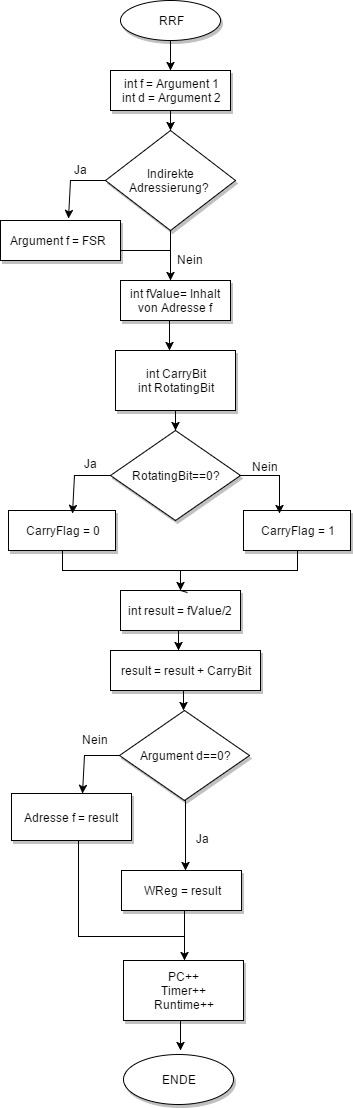


Abbildung 11. RRF

**XORLW**

XORLW führt eine Exklusivoderverknüpfung des W-Register mit einem Literal aus. Ist

das Ergebnis 0 wird das Zeroflag gesetzt. Der Befehl dauert 1 Zyklus und das Ergebnis wird in W gespeichert. XORLW führt eine Exklusivoderverknüpfung des W-Register mit einem Literal aus. Ist das Ergebnis 0 wird das Zeroflag gesetzt. Der Befehl dauert 1 Zyklus und das Ergebnis wird in W gespeichert.

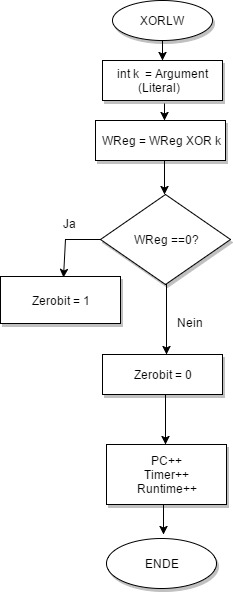


Abbildung 12. XORLW

**BSF**

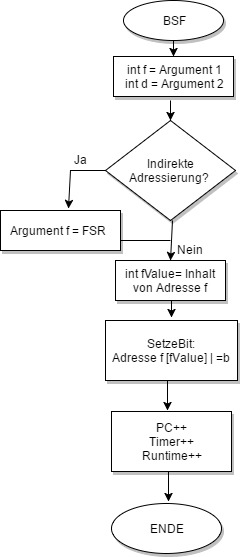
****

Abbildung 13. BSF

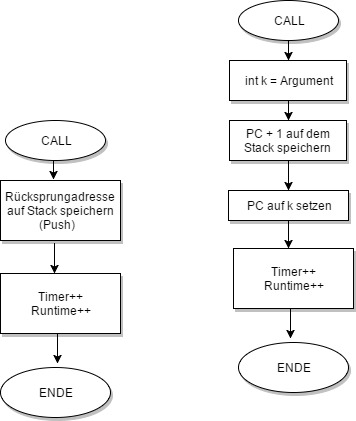
**CALL**  


Abbildung 14. CALL

## Interrupfunktion

# Zusammenfassung

## Umsetzung

## Fazit