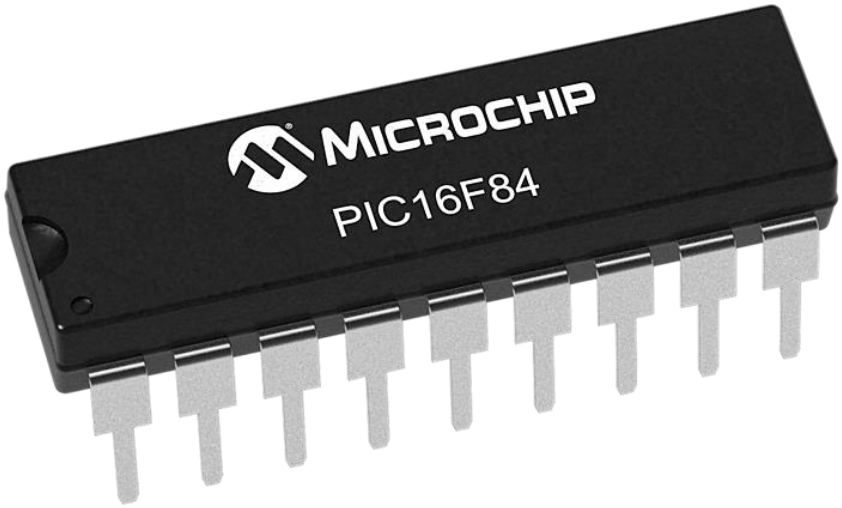
Pic Simulator Dokumentation

Juni 13

2017

Von Patrick Treyer und Sascha Hug aus dem Kurs TINF15b3 im 4.Semester

Abgabedatum:
19.06.2017



PIC Simulator

Inhalt

Vorwort

Im dritten Semester des Studiengangs Informationstechnik an der DHBW Karlsruhe wurde im Modul Rechnertechnik der Microcontroller PIC16F84 vorgestellt. Im vierten Semester galt es einen Simulator des in der Vorlesung behandelten Microcontrollers zu programmieren. Es sollen die wichtigsten Funktionen des PICs nachimplementiert werden. Während des Programmablaufs werden alle wichtigen Speicherbänke, Register und Ports auf einer grafischen Benutzeroberfläche dargestellt. Die Arbeit erfolgte in zweier Teams.

Was ist ein Simulator ?

Ein Simulator wird beispielsweise eingesetzt, wenn sich ein reales System nicht direkt beobachten lässt. Durch die Simulation kann das interne Verhalten dieses Systems dargestellt und nachvollzogen werden. Der PIC-Simulator hat vielerlei Vorteile, jedoch auch Nachteile. Zu den Vorteilen zählt vor allem, dass der Simulator ein Entwicklungswerkzeug darstellt. Assembler-Programme für den PIC können somit auf Korrektheit geprüft werden. Für Anfänger kann der Simulator zum Verstehen eines Microcontrollers beitragen, außerdem werden Kosten eingespart, da kein realer Microcontroller vorliegen muss. Andererseits ist ein Nachteil von Simulatoren, dass diese Funktionen nur einfach nachbilden. Sie entsprechen nicht in allen Fällen der realen Welt. Für den zu implementierenden PIC-Simulator steht fest, dass dieser die Hauptfunktionen unterstützen und korrekt arbeiten soll. Es ist aber auch unverkennbar, dass er nicht zu hundert Prozent den realen Microcontroller darstellen wird.

Vorteile einer Simulation

Durch Simulationen lassen sich auch gefährliche reale Gegebenheiten sicher nachstellen. So werden Simulationen zur Ausbildung von Piloten genutzt oder es werden mit Crash-Test-Dummies Verkehrsunfälle simuliert. Darüber hinaus lassen sich Abläufe verlangsamt darstellen und sind somit leichter nachvollziehbar.

Im Falle das PIC16 Mikrocontroller können Programme für diesen Controller vorab getestet und debuggt werden und somit Fehlerquellen vor dem aktiven Einsatz beseitigt werden.

Nachteile einer Simulation

Eine Simulation ist immer der Begrenztheit der Mittel unterworfen. Sei es nun Rechenleistung, Zeit oder Geld. Somit kann die Realität oftmals nur in einem vereinfachten Modell nachgebildet werden. Durch die Vereinfachung des Modells sind auch die Messergebnisse von einer gewissen

Ungenauigkeit. Die Modelle für eine Simulation werden für einen gewissen Parameterbereich entwickelt. Die Anwendung eines Simulators für Modelle außerhalb seines Parameterbereiches kann zu von der Realität verschiedenen Ergebnissen führen.

Für den PIC16-Simulator gilt, dass er fehlerfrei und möglichst genau arbeiten muss, denn Fehler innerhalb des Simulators lassen falsche Rückschlüsse auf das Programm zu. Ebenso arbeitet der Simulator verlangsamt und kann nicht die realen 2

Zeiten abbilden. Das bedeutet, dass die Laufzeit eines Programms in der Realität nicht der des Simulators entspricht.

Mikrocontroller

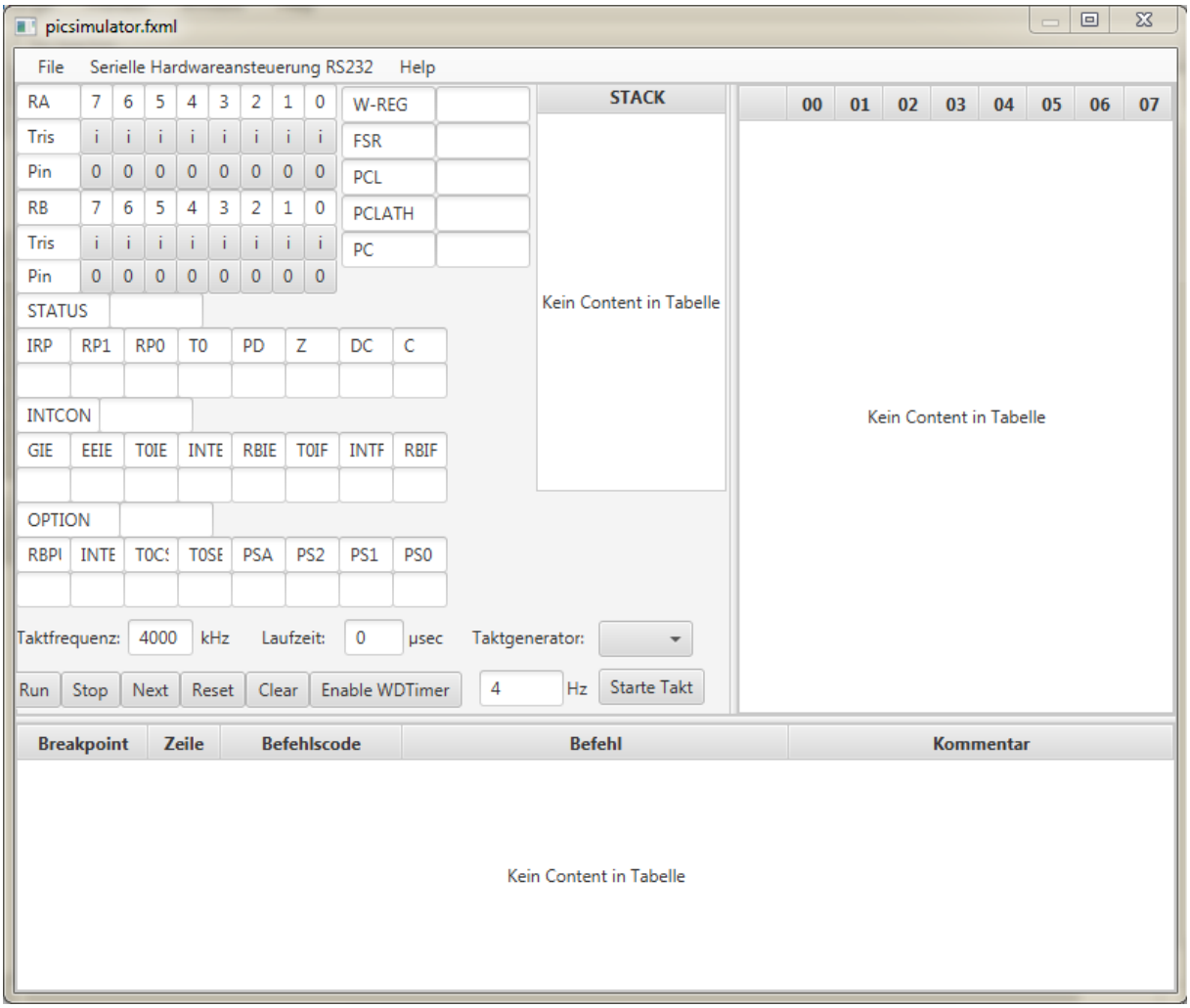
Ein Mikrocontroller ist eine Art Mikrorechnersystem, bei welchem neben ROM und RAM auch Peripherieeinheiten wie Schnittstellen, Timer und Bussysteme auf einem einzigen Chip integriert sind.

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 Prorgrammierschnittstelle zum Benutzer. Sie werden in der Regel einmal programmiert und installiert.

Die Benutzeroberfläche

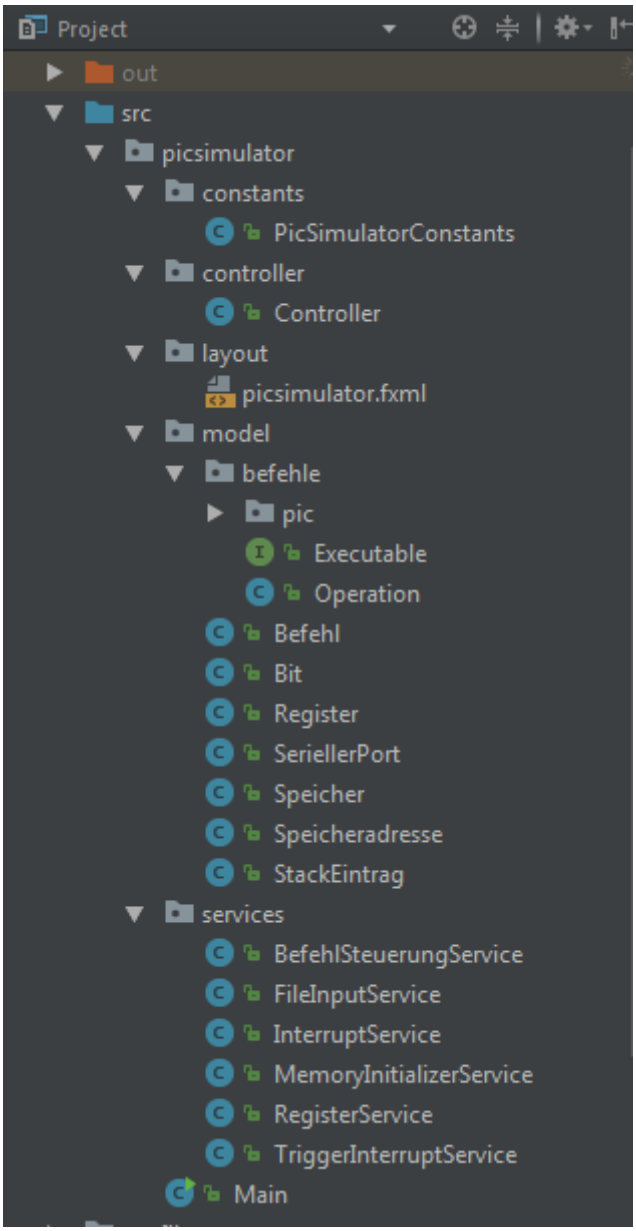
Die Benutzeroberfläche lässt sich in drei Bereiche unterteilen. Im erste Bereich, unteres Drittelwird der Programmcode angezeigt. Darüber sind die Steuerbuttons des Simulators implementiert. Auf der rechten Seite des Fensters, wird RAM-Speicher in Form einer Tabelle abgebildet.

Der Großteil der Benutzeroberfläche nehmen die SFR (Special Function Register)in Anspruch. Darin werden die Register Bänke, sowie besondere Register wie (Status, Intcon, Option)angezeigt. Auch der Stack wird durch eine Tabelle symbolisiert.


picsimulator.fxml
File
Serielle Hardwareansteuerung RS232
Help
RA 7 6 5 4 3 2 1 0
Tris i i i i i i i i
Pin 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
Pin 0 0 0 0 0 0 0 0
W-REG
FSR
PCL
PCLATH
PC
STACK
Kein Content in Tabelle
00 01 02 03 04 05 06 07
Kein Content in Tabelle
STATUS
IRP RP1 RP0 T0 PD Z DC C
INTCON
GIE EEIE T0IE INTE RBIE T0IF INTF RBIF
OPTION
RBPI INTE T0CS T0SE PSA PS2 PS1 PS0
Taktfrequenz: 4000 kHz
Laufzeit: 0 µsec
Taktgenerator:
Run
Stop
Next
Reset
Clear
Enable WDTimer
4 Hz
Starte Takt
Breakpoint
Zeile
Befehlscode
Befehl
Kommentar
Kein Content in Tabelle

Realisierung

Projektstruktur



picsimulator.xml

In der picsimulator.xml wird das Layout unserer Oberfläche beschrieben. Buttons sowie alles was der Benutzer auf der Oberfläche sieht, wird dort erstellt.

Controller

Controller, welcher direkt von der grafischen Oberfläche angesteuert wird, mit dieser interagiert und die entsprechende Logik in den Services ansteuert.

Services

Die Service-Klassen sind jeweils für die Logischen Funktionen zur Verarbeitung der Bits.

Aufgaben:

- Auswertung und Ansteuerung der Befehle
- Laden und Auswerten eines LST-Files
- Überprüfung von Interrupts
- Initialisierung des Speichers
- Änderungen die entsprechenden Bits für die Triggerung eines Interrupts

Operation (Befehle Vaterklasse):

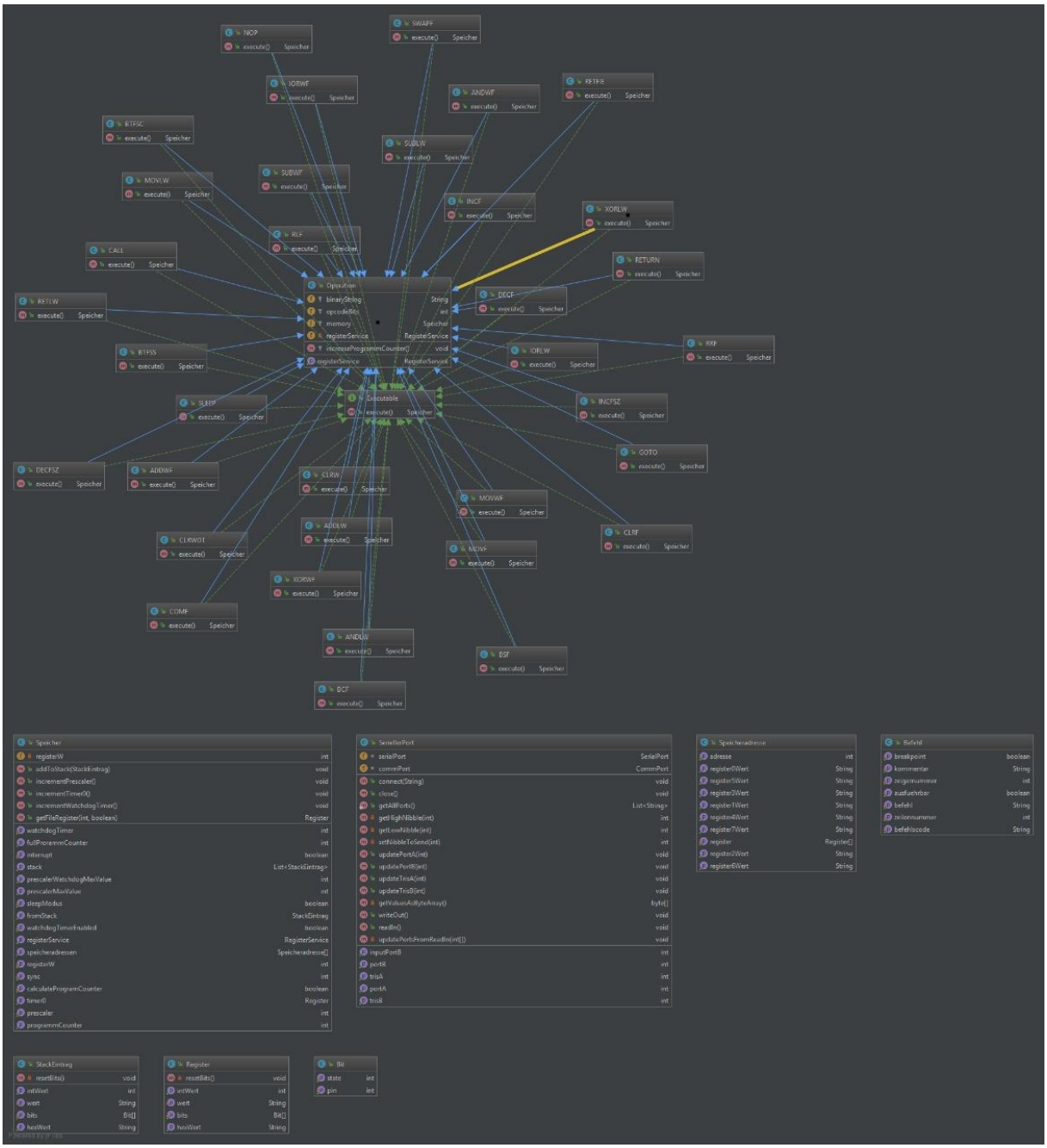
Vaterklasse der einzelnen Befehle, welche alle von dieser erben. Hier werden allgemeingültige Variablen sowie Methoden definiert, die von allen Befehlen benötigt werden.

Befehle

Im Ordner befehle ist für jeden Befehl eine eigene Klasse, die die Logik des Befehls nachstellt. Genauere Informationen können in dem Kapitel Befehlsbeschreibung eingesehen werden.

Gesamtübersicht

Hier wird die gesamt Struktur der Klassen, sowie die Verbindung untereinander dargestellt.



Funktionalitäten

Code einlesen

Den Code einlesen, haben wir in einen Service ausgelagert (FileInputService). Dieser Service importiert ein File, in dem er einen FileChooser öffnet anhand welchem der Benutzer ein File auswählen kann. Dann werden die Inhalte des Files auf den jeweiligen Datentyp, Befehl gemappt.

```
public List<Befehl> importFile() {
    try {
        FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("LST-Files", "*.LST"));
        File selectedFile = fileChooser.showOpenDialog(null);
        if (selectedFile == null) {
            return new ArrayList<>();
        }
        InputStream inputStream = new FileInputStream(selectedFile);
        BufferedReader in = new BufferedReader(new
InputStreamReader(inputStream));
        String line;
        List<Befehl> befehle = new ArrayList<>();
        while ((line = in.readLine()) != null) {
            Befehl befehl = new Befehl();
            char ch = line.charAt(0);

            if (ch != ' ' && ch != '\t' && ch != '\r' && ch != '\n') {
                befehl.setAusfuehrbar(true);
            }

            if (befehl.isAusfuehrbar()) {
                String subline = line.substring(0, 25);
                subline = subline.replaceAll("\\s+", "");

befehl.setZeigernummer(Integer.parseInt(subline.substring(0, 4), 16));
                befehl.setBefehlscode(subline.substring(4, 8));

befehl.setZeilennummer(Integer.parseInt(subline.substring(8, 13)));
            } else {
                String subline = line.substring(0, 25);
                subline = subline.replaceAll("\\s+", "");
                befehl.setBefehlscode("-");

befehl.setZeilennummer(Integer.parseInt(subline.substring(0, 5)));
                befehl.setZeigernummer(0);
            }

            String subline2 = line.substring(25);
            String[] strings = subline2.split(";");
            for (int i = 0; i < strings.length; i++) {
                if (i == 0) {
                    befehl.setBefehl(strings[0]);
                    continue;
                }
                befehl.setKommentar(strings[i]);
            }
            befehle.add(befehl);
        }
        in.close();
        return befehle;
```

```
    } catch (IOException e) {
        e.printStackTrace();
    }
    return new ArrayList<>();
}
```

Befehlsabarbeitung

Die Bearbeitung der Befehle haben wir in einen Service ausgelagert. Dieser Service wertet den übergebenen BinaryString (Opcode in einen String formatiert) aus und erkennt so, welcher Befehl ausgeführt werden muss. Somit wird eine Instanz des auszuführenden Befehls erstellt und die Ausführung angestoßen.

Wie man deutlich erkennen kann, wird in dieser Klasse eine große Verzweigung gemacht, die den Befehl ansteuern muss (ein kleiner Ausschnitt)

```
public Speicher steuereBefehl(Speicher speicher, String binaryString) {

    if (binaryString.startsWith("000111")) {
        ADDWF addwf = new ADDWF(binaryString, 6, speicher);
        return addwf.execute();
    }
    if (binaryString.startsWith("000101")) {
        ANDWF andwf = new ANDWF(binaryString, 6, speicher);
        return andwf.execute();
    }
…
```

Interrupts

Bei einem Interrupt verlässt der PIC16F seine normale Routine und springt in eine Interruptroutine, die er abarbeitet um dann wieder an die Stelle des normalen Ablaufs zurückzukehren.

Bei einem Interrupt müssen von Seiten des Simulators immer die gleichen Dinge ausgeführt werden. Der aktuelle Programm-Counter muss auf den Stack geschrieben werden und in den Programm-Counter muss die Adresse eingetragen werden, in der die Interruptroutinen liegen.

Bevor aber ein Interrupt wirklich ausgelöst wird, wird erst geprüft ob das Global-Interrupt-Enable (GIE) Bit gesetzt ist.

Es wird im Folgenden beschrieben wie die Interrupt Funktionalität für die einzelnen Interrupttypen implementiert wurde.

Da laufend geprüft werden muss, ob ein Interrupt vorliegt, wurde eine Methode checkInterrupt() geschrieben.

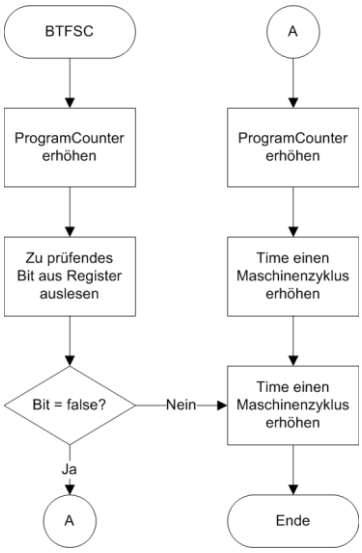
Der Service InterruptService prüft alle möglichen Interruptquellen.

Bevor auf einen RB0 Interrupt oder einen RB Port Change Interrupt überprüft wird, wird geprüft, ob im INTCON-Register das Interrupt Enable Bit (INTE) für einen RB0 Interrupt oder das RBIE für einen RB Port Change Interrupt gesetzt sind.

```
public class InterruptService {
public Speicher checkForTMR0TimerInterrupt(Speicher speicher, int cycles)
{…}
private Speicher checkForTMR0Interrupt(Speicher speicher) {…}
private Speicher incrementTimer0(Speicher speicher, int cycles) {…}
public Speicher checkForINTInterrupt(Speicher speicher) {…}
public Speicher checkForPortBInterrupt(Speicher speicher) {…}
public Speicher checkForWatchDogInterrupt(Speicher speicher, int cycles)
{…}
}
```

Befehlsbeschreibung

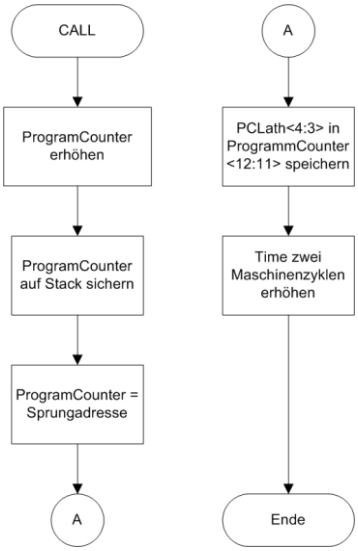
BTFSC (f,b)




```
public class BTFSC extends Operation implements Executable {

    public BTFSC(String binaryString, int opcodeBits, Speicher memory) {
        super(binaryString, opcodeBits, memory);
    }

    @Override
    public Speicher execute() {
        int registerIndex = opcodeBits + 3;
        String binBit = binaryString.substring(opcodeBits, registerIndex);
        String register = binaryString.substring(registerIndex);
        int bit = getRegisterService().binToInt(binBit);
        Bit selectedBit =
memory.getFileRegister(getRegisterService().binToInt(register),
false).getBits()[bit];

        if (selectedBit.getPin() == 0) {
            Controller.increaseRuntime();
            NOP nop = new NOP(binaryString, 14, memory);
            memory = nop.execute();
        }
```

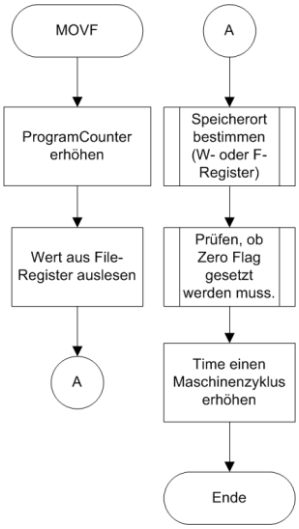
```
        Controller.increaseRuntime();
        increaseProgrammCounter();
        return memory;
    }
}
```

GOTO (k)



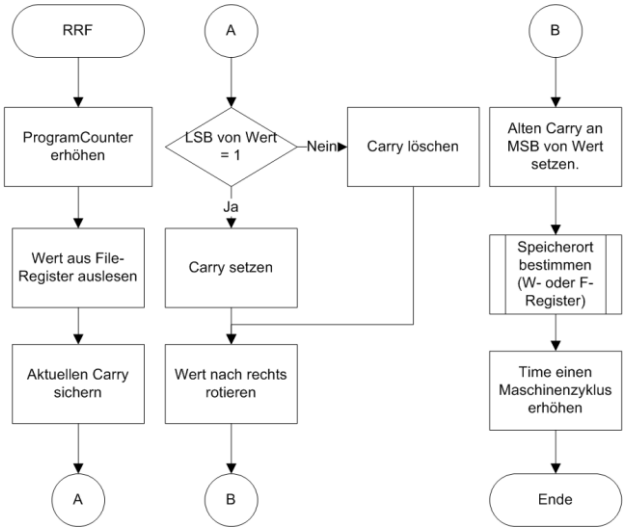

```
public class GOTO extends Operation implements Executable {

    public GOTO(String binaryString, int opcodeBits, Speicher memory) {
        super(binaryString, opcodeBits, memory);
    }

    @Override
    public Speicher execute() {
        String addresse = binaryString.substring(opcodeBits);

memory.getSpeicheradressen()[0].getRegister()[2].setWert(addresse.substring
(3));

memory.getSpeicheradressen()[1].getRegister()[2].getBits()[2].setPin(Charac
ter.getNumericValue(addresse.charAt(0)));

memory.getSpeicheradressen()[1].getRegister()[2].getBits()[1].setPin(Charac
ter.getNumericValue(addresse.charAt(1)));

memory.getSpeicheradressen()[1].getRegister()[2].getBits()[0].setPin(Charac
ter.getNumericValue(addresse.charAt(2)));

        Controller.increaseRuntime();
        Controller.increaseRuntime();
        return memory;
    }
}
```

MOVF (f,d)




```
public class MOVF extends Operation implements Executable {

    public MOVF(String binaryString, int opcodeBits, Speicher memory) {
        super(binaryString, opcodeBits, memory);
    }

    @Override
    public Speicher execute() {
        String ziel = binaryString.substring(opcodeBits, opcodeBits + 1);
        String registerAdress = binaryString.substring(opcodeBits + 1);
        int registerNr = getRegisterService().binToInt(registerAdress);

        if (memory.getFileRegister(registerNr, false).getIntWert() == 0) {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[2].setPin(1);
        } else {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[2].setPin(0);
        }

        /**
         * Prüft wohin das Ergebnis geschrieben werden soll
         */
        if (Integer.parseInt(ziel) == 0) {
            memory.setRegisterW(memory.getFileRegister(registerNr,
false).getIntWert());
        }

        Controller.increaseRuntime();
        increaseProgrammCounter();
        return memory;
    }
}
```

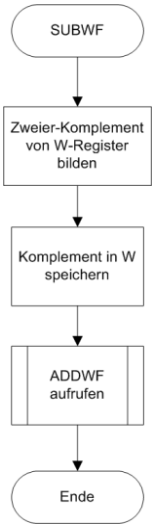
RLF(f,d)




```
public class RLF extends Operation implements Executable {

    public RLF(String binaryString, int opcodeBits, Speicher memory) {
        super(binaryString, opcodeBits, memory);
    }

    @Override
    public Speicher execute() {
        String ziel = binaryString.substring(opcodeBits, opcodeBits + 1);
        String registerAdress = binaryString.substring(opcodeBits + 1);
        int registerNr = getRegisterService().binToInt(registerAdress);

        Bit[] bits = memory.getFileRegister(registerNr, false).getBits();
        Register shiftedRegister = new Register();

        int pin = new
Integer(memory.getSpeicheradressen()[0].getRegister()[3].getBits()[0].getPi
n());
        shiftedRegister.getBits()[0] = new Bit(pin, 0);
        shiftedRegister.getBits()[1] = new Bit(bits[0].getPin(), 0);
        shiftedRegister.getBits()[2] = new Bit(bits[1].getPin(), 0);
        shiftedRegister.getBits()[3] = new Bit(bits[2].getPin(), 0);
        shiftedRegister.getBits()[4] = new Bit(bits[3].getPin(), 0);
        shiftedRegister.getBits()[5] = new Bit(bits[4].getPin(), 0);
        shiftedRegister.getBits()[6] = new Bit(bits[5].getPin(), 0);
        shiftedRegister.getBits()[7] = new Bit(bits[6].getPin(), 0);

        memory.getSpeicheradressen()[0].getRegister()[3].getBits()[0] = new
Bit(bits[7].getPin(), 0);

        /**
         * Prüft wohin das Ergebnis geschrieben werden soll
         */
        if (Integer.parseInt(ziel) == 0) {
            memory.setRegisterW(shiftedRegister.getIntWert());
        } else {
            memory.getFileRegister(registerNr,
true).setWert(shiftedRegister.getIntWert());
        }

        Controller.increaseRuntime();
        increaseProgrammCounter();
        return memory;
```

```
    }
}
```

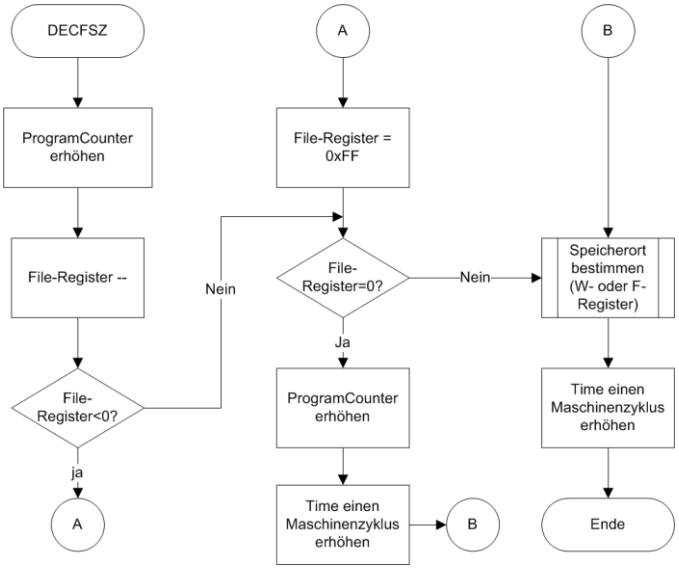
SUBWF (f,d)




```
public class SUBWF extends Operation implements Executable {

    public SUBWF(String binaryString, int opcodeBits, Speicher memory) {
        super(binaryString, opcodeBits, memory);
    }

    @Override
    public Speicher execute() {
        String ziel = binaryString.substring(opcodeBits, opcodeBits + 1);
        String registerAdress = binaryString.substring(opcodeBits + 1);
        int registerNr = getRegisterService().binToInt(registerAdress);
        int result = memory.getFileRegister(registerNr, false).getIntWert()
- memory.getRegisterW();

        /**
         *  Check Carry Flag
         */
        String wTwoCom =
getRegisterService().leftPad32(Integer.toBinaryString(memory.getRegisterW()
* -1));
        wTwoCom = wTwoCom.substring(24);
        short wC = (short) (Short.parseShort(wTwoCom, 2) & 0xFF);
        short fileC =
Short.parseShort((Integer.toString(memory.getFileRegister(registerNr,
false).getIntWert() & 0xFF)));
        short resultC = (short) (fileC + wC);
        if (Short.toUnsignedInt(resultC) > 255) {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[0].setPin(1);
        } else {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[0].setPin(0);
        }

        /**
         *  Check Digit Carry Flag
```

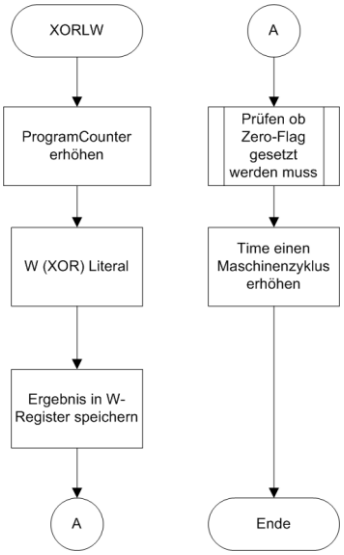
```
         */
        short wDC = (short) (Short.parseShort(wTwoCom, 2) & 0xF);
        short fileDC =
Short.parseShort((Integer.toString(memory.getFileRegister(registerNr,
false).getIntWert() & 0xF)));
        short resultDC = (short) (fileDC + wDC);
        if (Short.toUnsignedInt(resultDC) > 15) {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[1].setPin(1);
        } else {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[1].setPin(0);
        }

        /**
         *  Check Zero Flag
         */
        if (result == 0) {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[2].setPin(1);
        } else {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[2].setPin(0);
        }

        if (result < 0) {
            result = 256 - (result * -1);
        }

        /**
         * Prüft wohin das Ergebnis geschrieben werden soll
         */
        if (Integer.parseInt(ziel) == 0) {
            memory.setRegisterW(result);
        } else {
            memory.getFileRegister(registerNr, true).setWert(result);
        }

        Controller.increaseRuntime();
        increaseProgrammCounter();
        return memory;
    }
}
```

DECFSZ(f,d)




```
public class DECFSZ extends Operation implements Executable {

    public DECFSZ(String binaryString, int opcodeBits, Speicher memory) {
        super(binaryString, opcodeBits, memory);
    }

    @Override
    public Speicher execute() {
        String ziel = binaryString.substring(opcodeBits, opcodeBits + 1);
        String registerAdress = binaryString.substring(opcodeBits + 1);
        int registerNr = getRegisterService().binToInt(registerAdress);

        int aktuellerWert = memory.getFileRegister(registerNr,
false).getIntWert();
        int dekrementierterWert = aktuellerWert - 1;

        /**
         * Prüft wohin das Ergebnis geschrieben werden soll
         */
        if (Integer.parseInt(ziel) == 0) {
            memory.setRegisterW(dekrementierterWert);
        } else {
            memory.getFileRegister(registerNr,
false).setWert(dekrementierterWert);
        }

        if (dekrementierterWert == 0) {
            Controller.increaseRuntime();
            NOP nop = new NOP(binaryString, 14, memory);
            memory = nop.execute();
        } else {
            memory.getFileRegister(registerNr, true);
        }

        Controller.increaseRuntime();
        increaseProgrammCounter();
        return memory;
    }
}
```

XORLW (k)




```
public class XORLW extends Operation implements Executable {

    public XORLW(String binaryString, int opcodeBits, Speicher memory) {
        super(binaryString, opcodeBits, memory);
    }

    @Override
    public Speicher execute() {
        String reverseliteral = binaryString.substring(opcodeBits);
        String literal = new
StringBuilder(reverseliteral).reverse().toString();

        Register resultRegister = new Register();
        Register wRegister = new Register();
        wRegister.setWert(memory.getRegisterW());


resultRegister.getBits()[0].setPin(Character.getNumericValue(literal.charAt
(0)) ^ wRegister.getBits()[0].getPin());

resultRegister.getBits()[1].setPin(Character.getNumericValue(literal.charAt
(1)) ^ wRegister.getBits()[1].getPin());

resultRegister.getBits()[2].setPin(Character.getNumericValue(literal.charAt
(2)) ^ wRegister.getBits()[2].getPin());

resultRegister.getBits()[3].setPin(Character.getNumericValue(literal.charAt
(3)) ^ wRegister.getBits()[3].getPin());

resultRegister.getBits()[4].setPin(Character.getNumericValue(literal.charAt
(4)) ^ wRegister.getBits()[4].getPin());

resultRegister.getBits()[5].setPin(Character.getNumericValue(literal.charAt
(5)) ^ wRegister.getBits()[5].getPin());

resultRegister.getBits()[6].setPin(Character.getNumericValue(literal.charAt
(6)) ^ wRegister.getBits()[6].getPin());

resultRegister.getBits()[7].setPin(Character.getNumericValue(literal.charAt
(7)) ^ wRegister.getBits()[7].getPin());

        memory.setRegisterW(resultRegister.getIntWert());
```

```
        /**
         *  Check Zero Flag
         */
        if (resultRegister.getIntWert() == 0) {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[2].setPin(1);
        } else {

memory.getSpeicheradressen()[0].getRegister()[3].getBits()[2].setPin(0);
        }

        Controller.increaseRuntime();
        increaseProgrammCounter();
        return memory;
    }
}
```

Zusammenfassung

Umsetzung

Zu Beginn des Projektes wurde ein Pflichtenheft erstellt in dem die Muss,

Kann und Abgrenzungskriterien festgelegt wurden. Diese wurden soweit alle

eingehalten.

Zu den Muss Kriterien gehrten:

- Quellcode sichtbar anzeigen, einlesen und ausführen
- Einzelschritte, Start, Stopp
- Register
- Ports
- Flags anzeigen
- Interrupt
- Hilfe anzeigen
- externer Takt

Die Abgrenzungen müssen soweit weiterhin eingehalten werden. Der Simulator funktioniert nur mit einem korrekt funktionierenden Programmcode, und er übernimmt keinerlei Aufgaben eines Compilers.

Fazit

Durch verschiedene Erfahrungslevel im Bereich Software Entwicklung teilten wir die Aufgaben von Beginn an untereinander auf wie z.B. GUI Programmierung, Prozessor Programmierung und Dokumentation. Zeitaufwändigere Abschnitte wurden teilweise auch zusammen realisiert (wie z.B. Befehle ausprogrammieren). Durch Projekte in anderen Studienfächern konnten wir einige Erfahrungen zwischen den Projekten im Bereich Entwicklungsumgebung oder Versionierung (Git) austauschen. Zeitlich lag das Projekt parallel zu einem anderen im Fach Software Engineering wodurch es oftmals durch unerwartete Problemen die zunächst aufwendiger erschienen als sie wirklich waren die Zeit knapp. Jedoch konnten Zum Schluss alle zuvor im Pflichtenheft bestimmten Muss-Kriterien erfüllt werden.