Dokumentation

Rechnerarchitekturen

Praktikum

PIC16F8X Simulator

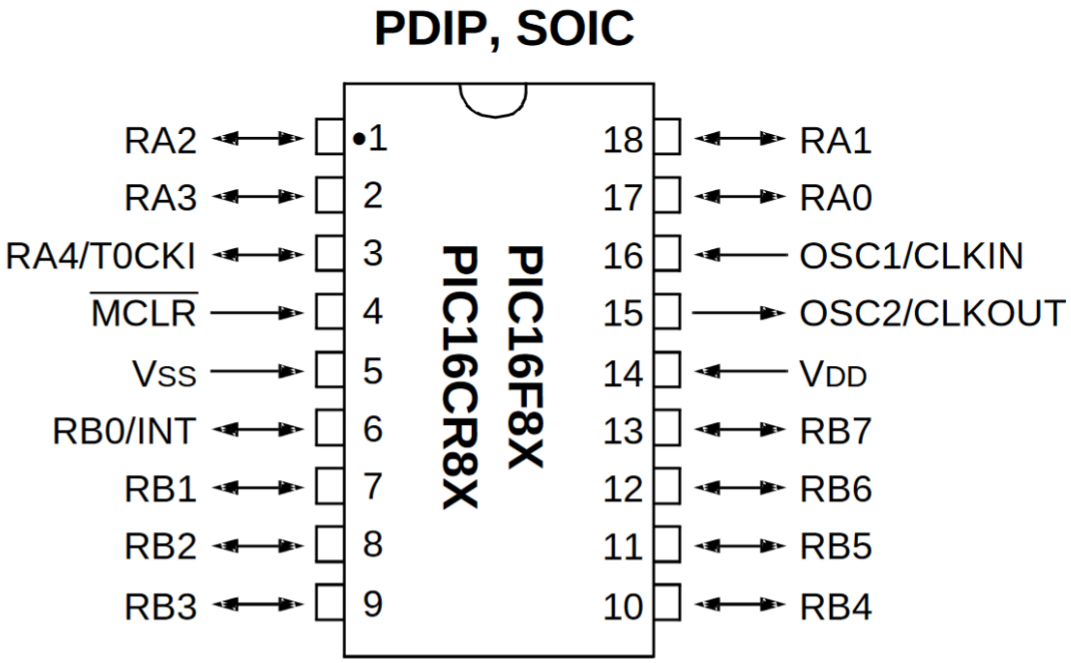


Abbildung 1 PIC16F8X Datenblatt

Stefan Lehmann Rechnerarchitekturen

Florian Grunwald, Niklas Studer

Angewandte Informatik

3. Semester

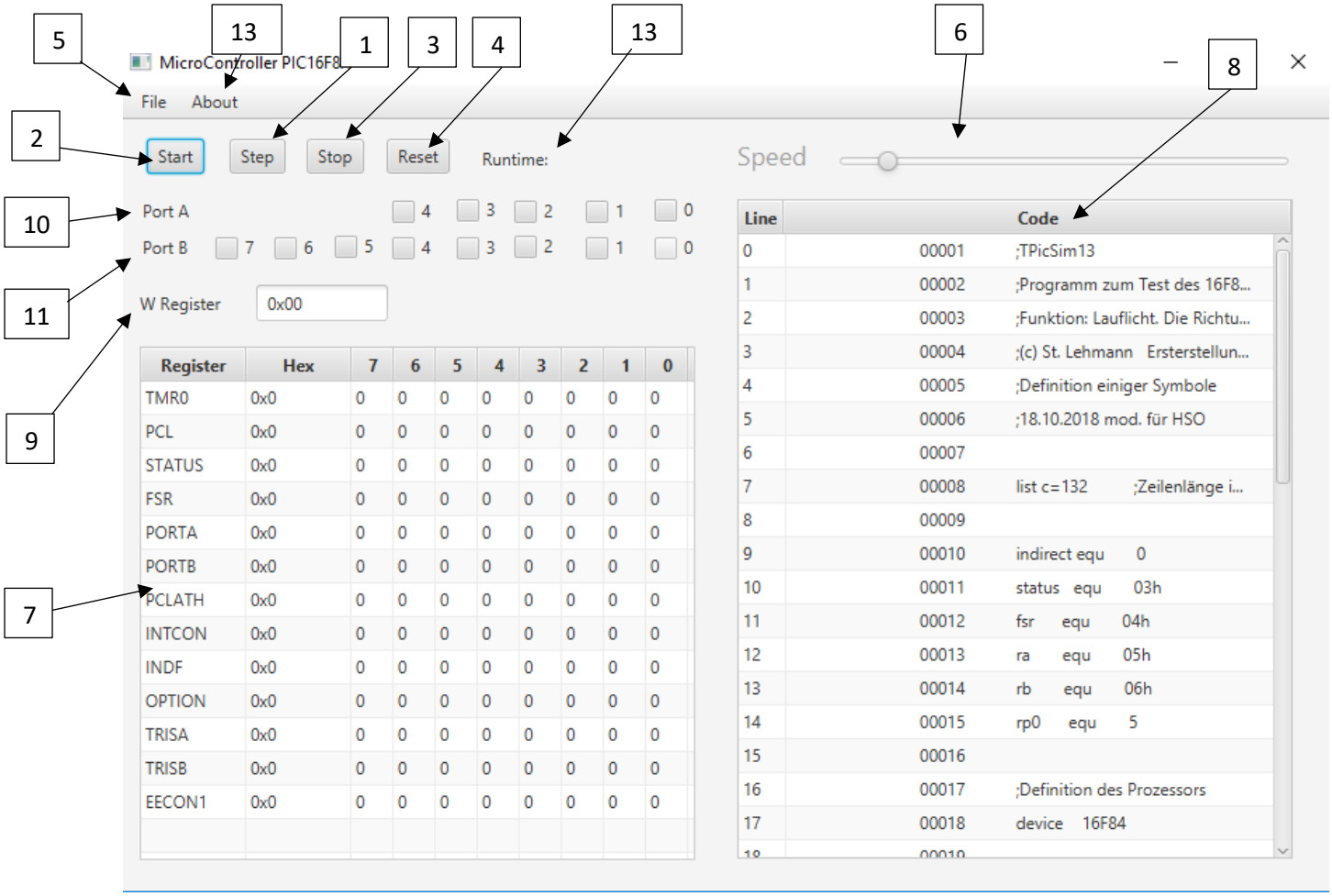
Inhaltsverzeichnis

2 Funktionen

- Die LST Dateien werden korrekt eingelesen
- Alle Befehle Werden von dem Simulator ausgeführt, bis auf CLRWDT und SLEEP
- Bankumschaltung funktioniert
- Indirekte Adressierung funktioniert
- Sprungbefehle mit Berücksichtigung des PCLATH sind möglich
- Port Interrupts sind verfügbar
- Timer Interrupt ist verfügbar

- Die Laufzeit wird angezeigt
- Beeinflussung der Ports durch Checkboxen
- Anzeigen der LST Datei in einer Tabelle
- Markierung des Nächsten Befehls in der LST Datei
- Anzeigen der Register in einer Tabelle
- Start, Stop, Step, Reset, Button verfügbar
- Öffnen der Dokumentation über das Programm

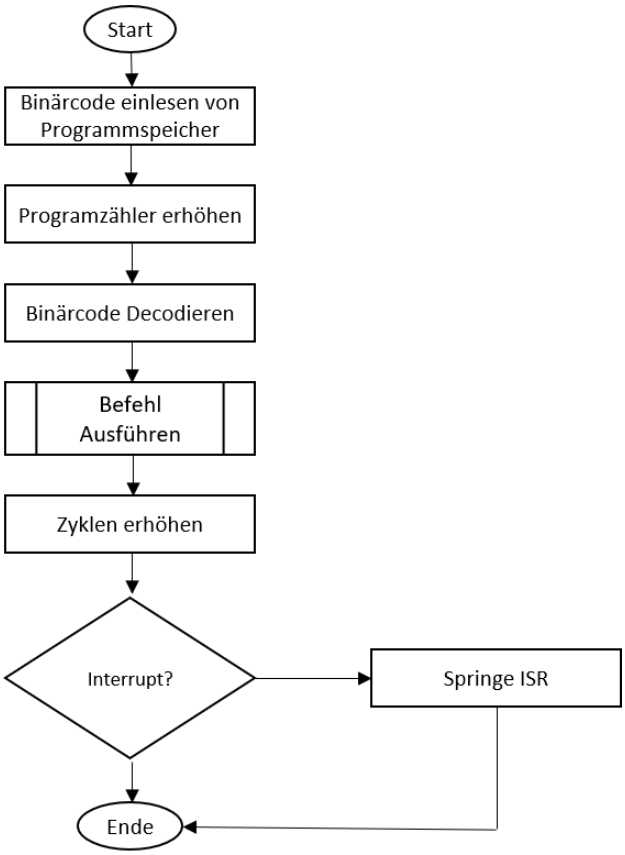
3 Userinterface



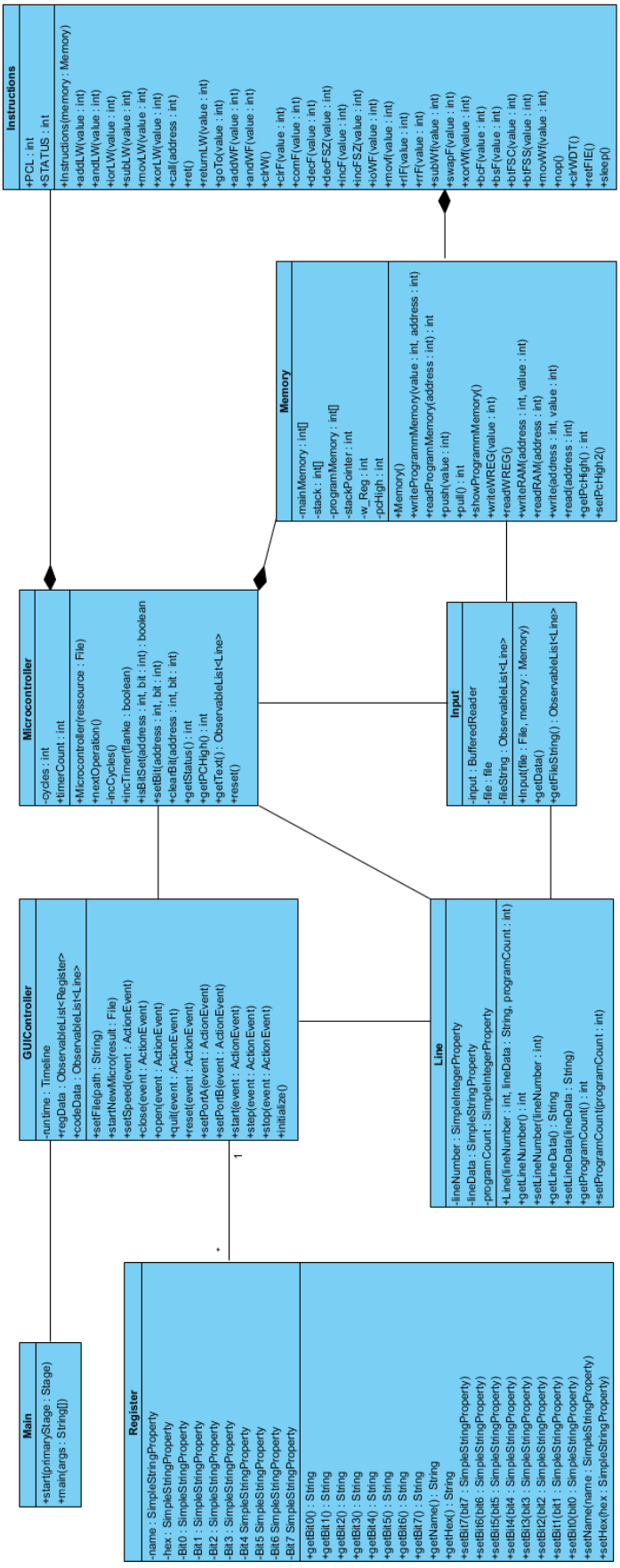
1. Step Button: Führt den nächsten Befehl aus
2. Start Button. Startet die Automatische Befehlsausführung
3. Stop Button: Stoppt die Automatische Befehlsausführung
4. Reset Button: Resettet den Microcontroller
5. File Menü: Öffnet eine neue LST Datei
6. Speed Slider: Einstellung der Geschwindigkeit
7. Register Tabelle: Zeigt die Einzelnen Register im Microcontroller an (Hex & Bits)
8. Quellcode Tabelle: Zeigt den Inhalt des LST Fiels an mit Zeiger auf den nächsten Befehl
9. W-Register: Zeigt den Inhalt des W-Registers an
10. Port A: Checkboxen als Input für den Port A
11. Port B: Checkboxen als Input für den Port B
12. Runtime: Zeigt die aktuelle Reale Zeit seit Programstart an
13. About: Zeigt Dokumentation als PDF an

4 Struktur

4.1 NextStep() PAP



4.2 Klassendiagramm



5 Programmdecoder

Der Programm Decoder wurde bei uns in einem Switch Case in der Klasse Microcontroller realisiert. Der Binärcode wird zuerst in die 4 Grund Befehlsarten aufgeteilt. In den jeweiligen Befehlsarten wird der Binärcode jeweils nochmal durch ein Switch analysiert.

```
switch (Binärcode & 0x3000) {
        0x0000: // 00 Befehle
            switch ((Binärcode & 0x0F00)) {
            0x0700: // ADDWF, 0x0500: // ANDWF, 0x0900: // COMF
            0x0300: // DECF, 0x0B00: // DECFSZ, 0x0A00: // INCF
            0x0F00: // INCFSZ, 0x0400: // IORWF, 0x0800: // MOVF
            0x0D00: // RLF, 0x0C00: // RRF, 0x0200: // SUBWF
            0x0E00: // SWAPF, 0x0600: // XORWF
            0x0100: // CLRF //CLRW
                if ((Binärcode & 0x3FFF) == 0x0100) {
                    // CLRW
                } else {
                    // CLRF
                }
            0x0000: // MOVWF //NOP //CLRWDT //RETFILE //RETURN //SLEEP
                switch ((Binärcode & 0x00FF)) {
                0x0000: // NOP, 0x0064: // CLRWDT, 0x0009: // RETFIE
                0x0008: // RETURN, 0x0063: // SLEEP, default: // MOVWF
        0x1000: // 01 Befehle
            switch (Binärcode & 0x0C00) {
            0x0000: // BCF, 0x0400: // BSF, 0x0800: // BTFSC, 0x0C00: // BTFSS
        0x2000: // 10 Befehle
            if ((Binärcode & 0x0800) > 0) {
                // GOTO} else { // call
        0x3000: // 11 Befehle
            switch (Binärcode & 0x0F00) {
            0x0E00: // ADDLW,0x0900: // ANDLW, 0x0800: // IORLW, 0x0000: //
            MOVLW, 0x0C00: // SUBLW, 0x0A00: // XORLW, 0x0400: // retLW
```

6 Indirekte Adressierung

In der Methode readRAM() wird jedes Mal wenn auf den RAM Speicher zugegriffen wird geprüft ob der Zugriff auf das Indirekt Register(Adresse 0x00) erfolgt. Erfolgt ein Zugriff, wird der Wert aus dem FSR Register zurückgegeben.

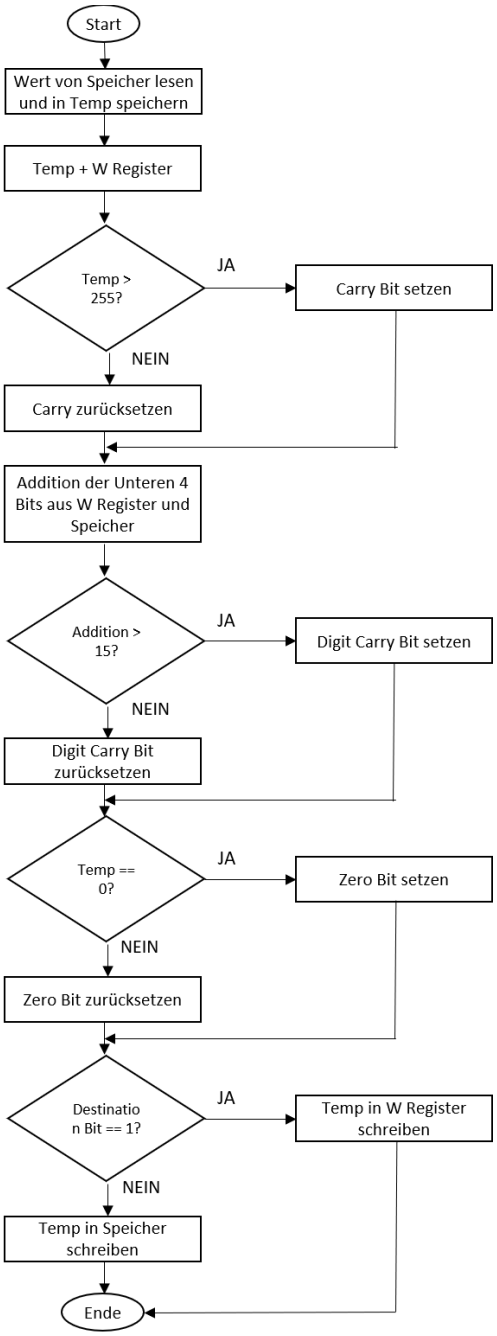
```
public int readRAM(int address) {
    if(address == 0) {
        return this.mainMemory[this.mainMemory[4]];
    }else {
        if((this.mainMemory[3] & (1<<5)) > 1) {
            if(address == 0 || address == 2 || address == 3 || address == 4 || address == 10 || address == 11) {
                return this.mainMemory[address];
            }else {
                return this.mainMemory[address+128];
            }
        }else {
            return this.mainMemory[address];
        }
    }
}
```

7 Befehle

7.1 Rechenbefehle

Am Beispiel addWF:

7.2 PAP



7.3 Code

```
public void addWf(int value) {
    int temp = this.memory.readRAM(value & 0x7F);
    temp += this.memory.readWREG();
    if(temp > 255) { //set Carry Bit
        this.memory.writeRAM(STATUS,this.memory.readRAM(STATUS) | 0x01);
    }else{
        this.memory.writeRAM(STATUS,this.memory.readRAM(STATUS) & 0xFE);
    }
    // set DC
    if(15 < ((this.memory.readWREG() & 0x0F)+(this.memory.readRAM(value & 0x7F) & 0x0F))) {
        this.memory.writeRAM(STATUS,this.memory.readRAM(STATUS) | 0x02);
    }else{
        this.memory.writeRAM(STATUS,this.memory.readRAM(STATUS) & 0xFD);
    }
    if(temp == 0) { //set Zero Bit
            this.memory.writeRAM(STATUS,this.memory.readRAM(STATUS) | 0x04);
        }else{
            this.memory.writeRAM(STATUS,this.memory.readRAM(STATUS) & 0xFB);
    }
    if((value & 0x80) > 0) {
        this.memory.writeRAM(value  & 0x7F,temp & 0xFF);
    }else {
        this.memory.writeWREG(temp & 0xFF);
    }
}
```

7.4 Logische Befehle

Am Beispiel ioWf:

```
public void ioWf(int value) {
    int temp = this.memory.readRAM(value  & 0x7F) | this.memory.readWREG();
    if(temp == 0) {
        this.memory.writeRAM(STATUS,this.memory.readRAM(STATUS) | 0x04);
    }else{
        this.memory.writeRAM(STATUS,this.memory.readRAM(STATUS) & 0xFB);
    }
    if((value & 0x80) > 0) {
        this.memory.writeRAM(value & 0x7F,temp & 0xFF);
    }else {
        this.memory.writeWREG(temp & 0xFF);
    }
}
```

7.5 Sprungbefehle

Am Beispiel Call:

Als erstes wird der Wert vom Programcounter in den Stack geschrieben, dann wird die übergebene Adresse in das PCL Register geschrieben und die oberen Bits der Adresse in die PcHigh Variable.

```
public void call(int address) {
    this.memory.push((this.memory.getPcHigh() << 8) + this.memory.read(PCL));
    this.memory.write(PCL,address & 0xFF);
    this.memory.setPcHigh(address>>8);
}
```

7.6 Bitbefehle

Am Beispiel btFSC:

Der Übergebene Wert wird um sieben nach Links geschiftet um die Bit Position zu speichern. Die Adresse entspricht den hinteren Bits des übergebenen Wertes. Mit der logischen AND der Maske 0x7F werden die Bits extrahiert. Es wird geprüft ob das Bit an der Stelle der Bit Position gesetzt ist indem die Adresse mit dem logischen AND des Wertes eins um die Entsprechende Bitposition geschiftet wird. Sollte dies gleich Null sein, wird der Programzähler um eins erhöht.

```
public void btFSC(int value) {
    int bitPosition = value >> 7;
    int address = value & 0x7F;
    if(0 == (this.memory.readRAM(address) & (0x1 << bitPosition))){
        this.memory.write(PCL, this.memory.readRAM(PCL)+1);
    }
}
```

8 Fazit