

Hochschule Offenburg

- Fakultät EMI –

Dokumentation über:

**- Simulation der Funktionsweise des
Mikrocontrollers PIC 16F84 –**

Vorgelegt bei:

Stefan Lehmann

Modul:

Rechnerarchitekturen

Vorgelegt von:

Eduard Wegele

ewegele@stud.hs-offenburg.de

190549

Angewandte Informatik SS2

Linus Brüstle

lbruestl1@stud.hs-offenburg.de

190560-01

Angewandte Informatik SS2

Gliederung

Inhalt

Gliederung.....	2
Einstieg in das Simulator Projekt.....	3
Konzept eines Simulator	3
Ziel unseres Projektes	3
Anleitung und Aufbau unserer GUI	4
Implementierung.....	7
Projektmanagement: Analyse, Planung und Vorgehen	7
Realisierung des Mikrocontrollers in Java	8
Ablauf einiger ausgewählter Befehle	9
Fazit	11

Einstieg in das Simulator Projekt

Konzept eines Simulator

Die aller erste Frage, die sich einem Außenstehenden stellen mag, lautet, wie genau der Begriff "Simulator" einzuordnen ist.

Vereinfacht gesagt ist ein Simulator ein Programm, das geschaffen wurde, um gewisse Dinge aus der Realität möglichst detailgetreu abzubilden.

Beispielsweise gibt es Simulatoren für das Autofahren oder Fliegen, die zum Beispiel bereits bei Fahrschulen Anwendung finden.

Benutzt man also einen Simulator, sollte sich Alles wie in der Wirklichkeit verhalten.

Ein Simulator bietet dem Benutzer unter Anderem folgende Möglichkeiten:

- erste realistische Erfahrungen, ohne dabei einem realen Risiko ausgesetzt zu sein
- kostengünstige und effektive Methode, um verschiedenste Situationen auszuprobieren und beliebig oft wiederholen zu können
 - Fehler lassen sich früh erkennen, wodurch Optimierungen durchgeführt werden können

In einem Autosimulator kann ich also testen, wie sich das Fahrzeug bei unterschiedlichen Wetterbedingungen verhält und dieses anhand der Ergebnisse anpassen. Der Überraschungseffekt bei der ersten Nutzung wird somit möglichst klein gehalten.

Dennoch ist es wichtig zu verstehen, dass ein Simulator nicht immer die goldene Lösung ist, denn nicht alle physischen Faktoren lassen sich gut simulieren. Der praktische Einsatz kann also nicht komplett ersetzt werden.

Ziel unseres Projektes

Die Aufgabe dieses Projektes war es, einen Simulator für den Mikroprozessor PIC16F84 zu erstellen.

Der Fokus lag dabei auf der Realisierung der Funktionalität des PIC, das Ausführen eines gegebenen Programmcodes soll also vollständig möglich sein.

Das Ziel ist also, eine digitale Version des Mikroprozessors zu haben, der bei Programmausführung ein möglichst exaktes Abbild zu einem realen Mikroprozessor ist.

Beim beispielsweise Auslesen des Arbeitsspeichers (RAM) sollen wir so bei Beiden die genau gleichen Werte erhalten (jedes einzelne Bit).

Physikalische Komponente wie Einschwingverhalten, Schaltschwellen, ein realer Zeitbezug, Temperaturverhalten, Stromverbrauch, Verlustleistung, Spannungsschwankungen usw. sind **nicht** Teil dieses Projektes / Simulators.

Anleitung und Aufbau unserer GUI

Die GUI ist eine visuelle Representation des PIC16F84 Mikroprozessors, wir erhalten also Einblick in den Arbeitsspeicher, Programmspeicher, Stack und ebenso die Kontrolle über alle I/O-Pins.

Dadurch bildet sich ein tieferes Verständnis zur Arbeitsweise des PIC, da live miterlebt sowie nachvollzogen werden kann, wie ein gegebener Programmcode stückweise abgearbeitet wird.

Die GUI ist in 5 allgemeinere Bereiche gegliedert, welche sich noch jeweils genauer aufteilen lassen

- Menüleiste
- Arbeitsspeicher
- Gegebener Programmcode
- Collection
- Controller

The screenshot displays the PIC16F84 Simulator interface. It features a menu bar at the top with 'Programm laden...', 'Help', and a window title 'PIC16F84 Simulator'. The main area is divided into several sections:

- Register:** A table showing the state of various PIC registers (0x00 to 0x1D) with their bit values (Bit 7 to Bit 0).
- Memory:** A table showing memory addresses (0x00 to 0x1D) and their corresponding data values.
- Stack:** A section showing the current stack pointer (0) and the contents of the stack (0).
- Program Code:** A list of assembly instructions with their addresses (0000 to 0029). The code includes instructions like 'movlw 11h', 'andlw 30h', 'iorlw 00h', 'sublw 30h', 'xorlw 20h', 'addlw 25h', 'goto ende', and 'org 0'.
- IO Pins:** A section showing the state of the I/O pins (RA0 to RA4, RB0 to RB4) and the PIN value (0).
- Control Buttons:** A row of buttons at the bottom: 'Restart', 'Stop', 'RUN' (highlighted in green), 'Next', and 'Ignore'.

Menüleiste

- über den Reiter "Programm laden" kann der Nutzer einen der vorgegebenen Programmcodes oder einen neuen / eigenen (dergleichen Art) wählen

Visualisierung Arbeitsspeicher:

aufgeteilt in SFR und GPR

Jede Speicherzelle des RAM wird mit seiner zugehörigen Adresse sowie den einzelnen Bits abgebildet

SFR: hier sind alle 16 SFR-Register des PIC visualisiert

GPR: hier sind alle 68bytes des GPR's (aus Bank 0) visualisiert

Visualisierung des gegebenen Programmcodes:

- hier ist der vom Benutzer ausgewählte Programmcode zu sehen
- der Programmzähler (PC) = gelbe Markierung, zeigt hierbei auf den als nächstes auszuführenden Befehl

Visualisierung Collection:

aufgeteilt in Stack, W-Register, Prescaler, I/O-Pins, Laufzeitzähler, WatchDog

- der Laufzeitzähler wird angezeigt und lässt sich resetten
- der WatchDog lässt sich über den Button aktivieren / deaktivieren
- die I/O-Pins sind durch den Benutzer veränderbar, Veränderungen werden autom. in die entsprechenden Register im RAM übertragen

Visualisierung Controller

Über die hier gegebenen Buttons lässt sich der Programmverlauf steuern

- RUN: startet die autom. Befehlsabarbeitung
- Stop: stoppt die autom. Befehlsabarbeitung
- Next: führt den nächsten Befehl aus
- Ignore: überspringt den nächsten Befehl, ohne ihn auszuführen
- Restart: erzeugt einen "Neustart"

Beispielhafter Ablauf durch den Benutzer

1. „Programm laden“
2. Programm mit „RUN“ oder „Next“ starten

Zur Runtime wird an, im Programmcodefenster gesetzten Breakpoints, angehalten.
Dies unterstützt das Debugging.

Implementierung

Projektmanagement: Analyse, Planung und Vorgehen

Da uns beiden der PIC gänzlich unbekannt war, haben wir uns früh dazu entschieden, die verschiedenen Aufgaben **nicht** nach Effizienz aufzuteilen, also **nicht** inhaltlich möglichst weit voneinander entfernt zu arbeiten.

Stattdessen haben wir inhaltlich eng aneinander gearbeitet und uns zusammen in die gleichen Themen eingelest, damit beide ein vollständiges Verständnis für den Mikrocontroller erhalten können.

So wurde beispielsweise der Arbeitsspeicher und Programmspeicher auf Beide aufgeteilt, anstatt die Speicher nur einer Person zuzuteilen. Dies wäre durchaus schneller gewesen, hätte aber den entscheidenden Nachteil gehabt, dass nun dem Anderem von Uns dieses Wissen fehlt.

Wären wir beide bereits erfahrenen im Umgang mit einem Mikrocontroller gewesen, hätten wir uns für die effizientere Variante entschieden, da wir dann ja bereits alles Wissen erworben hätten.

Gegen Ende des Projektes konnten wir dies aber immer mehr umsetzen, da nun die Bauteile des PIC beiden Personen bekannt waren.

Ebenso war uns von Anfang an sehr wichtig, dass wir eine klare Struktur in unserem Projekt und Code haben. Um dies zu verwirklichen, haben wir uns einiger Hilfsmittel bedient, die uns das Leben sehr vereinfacht haben:

- Versionskontrollsystem Git
- UML
- Notion & Adobe, um Informationen über zsm.gefasste Themen d. PIC zu teilen
- Discord Server, um über verschiedenste Inhalte strukturiert zu kommunizieren
- Wöchentliche digitale Meetings, um Verschiedenes zu besprechen

Um die für uns bisher ungewohnte Größe des Projektes verarbeiten zu können, gingen wir immer sequential nach folgendem Schema vor:

1. Bauteil / Funktion des PIC auswählen
2. Dieses bisher unbekannte Thema verstehen und zusammenfassen
3. Erarbeiten einer ersten Grundlösung zur Umsetzung im Code
4. Erstellung des ersten Aufbaus in UML
5. Implementierung & gründliche Erstellung von JUnit Tests

Teamarbeit: Schritt 1, 3, 4

Individuelle Arbeit: Schritt 2, 5

Zu jeder Zeit wurde das Versionskontrollsystem Git umfassend eingesetzt, um unabhängig voneinander arbeiten zu können und doch Ergebnisse leicht zusammenzutragen.

Ohne Git wäre dieses Projekt sehr umständlich umzusetzen gewesen.

Realisierung des Mikrocontrollers in Java

für mehr Detail siehe UML (Anhang) oder Code

Da es sich um eine Harward-Architektur handelt, gab es folgende Grundbauteile in Java umzusetzen:

- Getrennter Arbeitsspeicher und Programmspeicher
- Steuerwerk sowie Rechenwerk

Eine übergeordnete Klasse „MC“ besitzt all unsere Bauteile als Klassenvariablen. Diese Klasse ist also unser Mikroprozessor, über den wir an die einzelnen Komponenten gelangen.

Weitere Klassen / Packages für die Funktionalität des PIC sind:

- Instructions
- Programm Counter
- Stack
- Timer, WatchDog und Prescaler
- Interrupts
- Utils: jegliche Hilfsmethoden, z.B. Umwandlung versch. Zahlenformate
- GUI: Visualisierung des Mikrocontrollers

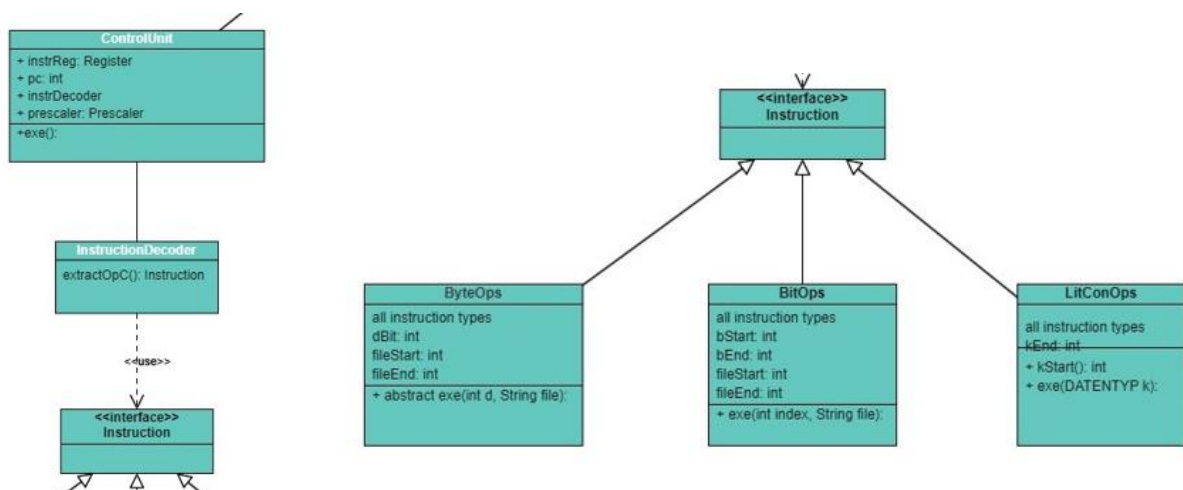
Genauere Betrachtung zur Befehlsabarbeitung

Jede der 3 Befehlsarten ByteOps, BitOps und LitConOps ist als eine Enum umgesetzt worden. Übergeordnet nutzen wir den Vorteil eines Interfaces „Instruction“, sodass diese 3 dadurch „gruppiert“ sind.

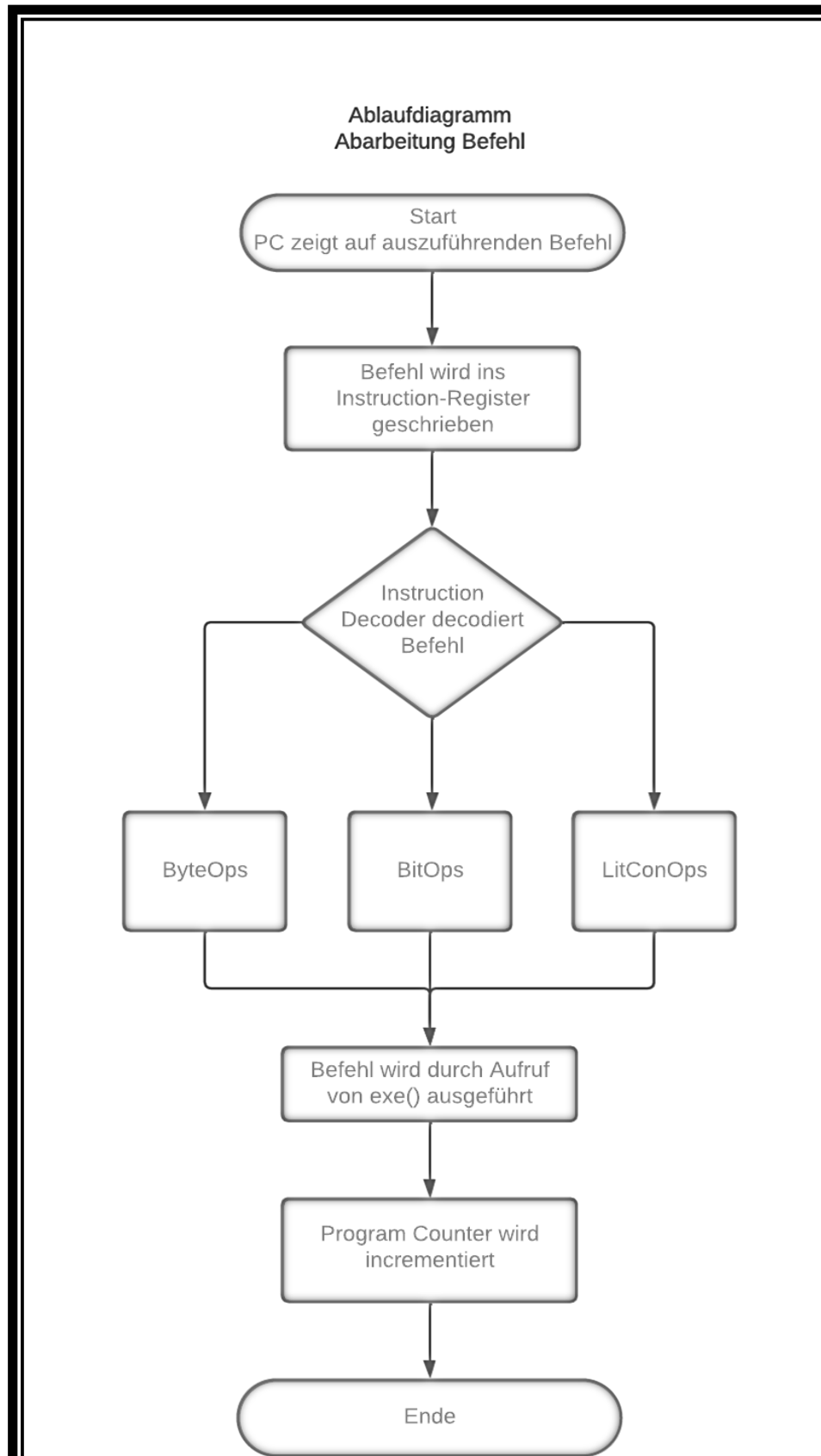
Jede Enumklasse besitzt all ihre eigenen Befehle als Methoden (MOVF, CALL, BCF).

Der Befehlsdecoder erhält den Bitbefehl aus dem Programmspeicher und liefert uns ein Enum zugehörig zum Interface „Instruction“ zurück.

Dadurch ist es uns möglich, in der ControlUnit die jeweilige Befehls-Methode des Enums aufzurufen, wodurch der richtige Befehl ausgeführt wird. Dies geschieht immer mit Beachtung der zu setzenden Flags und mit Beachtung der Interrupts.



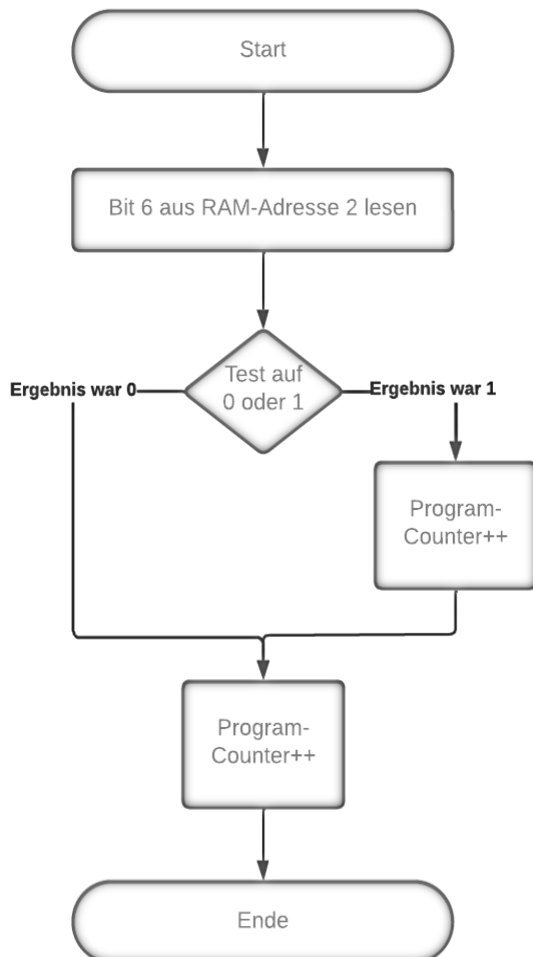
Ablauf einiger ausgewählter Befehle



Ablauf eines beliebigen Befehls

Programmsequenz
Abarbeitung BTFSS

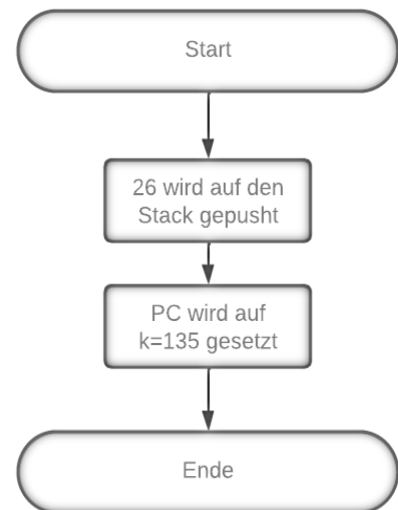
Aufruf von exe(int indexBit, int indexCell)
indexBit = 6
indexCell = 2



Programmsequenz
Abarbeitung CALL

Aufruf von exe(int[] k)
k = {0,0,0,1, 0,0,0,0, 1,1,1,1}

PC = 25



Fazit

Da wir erstmalig mit solch einem Softwareprojekt in Berührung kamen, hatten wir viel zu tun, denn von der erstmaligen Orientierungslosigkeit bis hin zum finalen Produkt war es ein weiter Weg. Dennoch hatten wir, trotz der vielen investierten Stunden, immer Spaß und sind froh über das gewonnene Wissen.

Das Umsetzen eines jeden Bauteils oder einer Funktion des PIC war jedesmal eine neue Herausforderung, da sich immer bisher unbekannte Probleme auftaten. Dennoch war es uns möglich, alle die für uns benötigten Bauteile in Code umzuwandeln und so eine möglichst detailgetreue Abbildung der Realität zu erzeugen.

Trotz aller Vorausplanung lies sich nicht immer vermeiden, dass wir rückwirkend einige Änderungen bzw. Verbesserungen vornehmen mussten, da wir durch unseren ständigen Wissenszuwachs neue Zusammenhänge erkennen konnten und besser verstanden, wie einzelne Bestandteile des PIC miteinander in Verbindung stehen.

Im Verlauf dieses Projektes haben wir durch all unsere Erfahrungen dementsprechend viel über die Softwareentwicklung lernen und unseren Blickwinkel auf das Berufsfeld erweitern können.

Wir haben unsere Kenntnis über Java in der Praxis anwenden und weiter ausbauen können, was uns ein tieferes Verständnis der Sprache erbracht hat. Auch das Schreiben vieler JUnit-Tests sowie das Lesen des jeweils anderen Codes hat uns viel gelehrt. Auch haben uns die immer neu auftretenden Probleme neue Lösungswege finden lassen, was unser Programmierlevel durchaus verbessert hat.

Des Weiteren hatten wir erste Berührungspunkte mit dem Thema Projektorganisation, sei es über Zeitmanagement, Team Koordination / Teamplay oder Planung & Strukturierung der Vorgehensweise.

Es ist wichtig, richtig miteinander zu kommunizieren, um Komplikationen möglichst schnell aus der Welt zu schaffen.

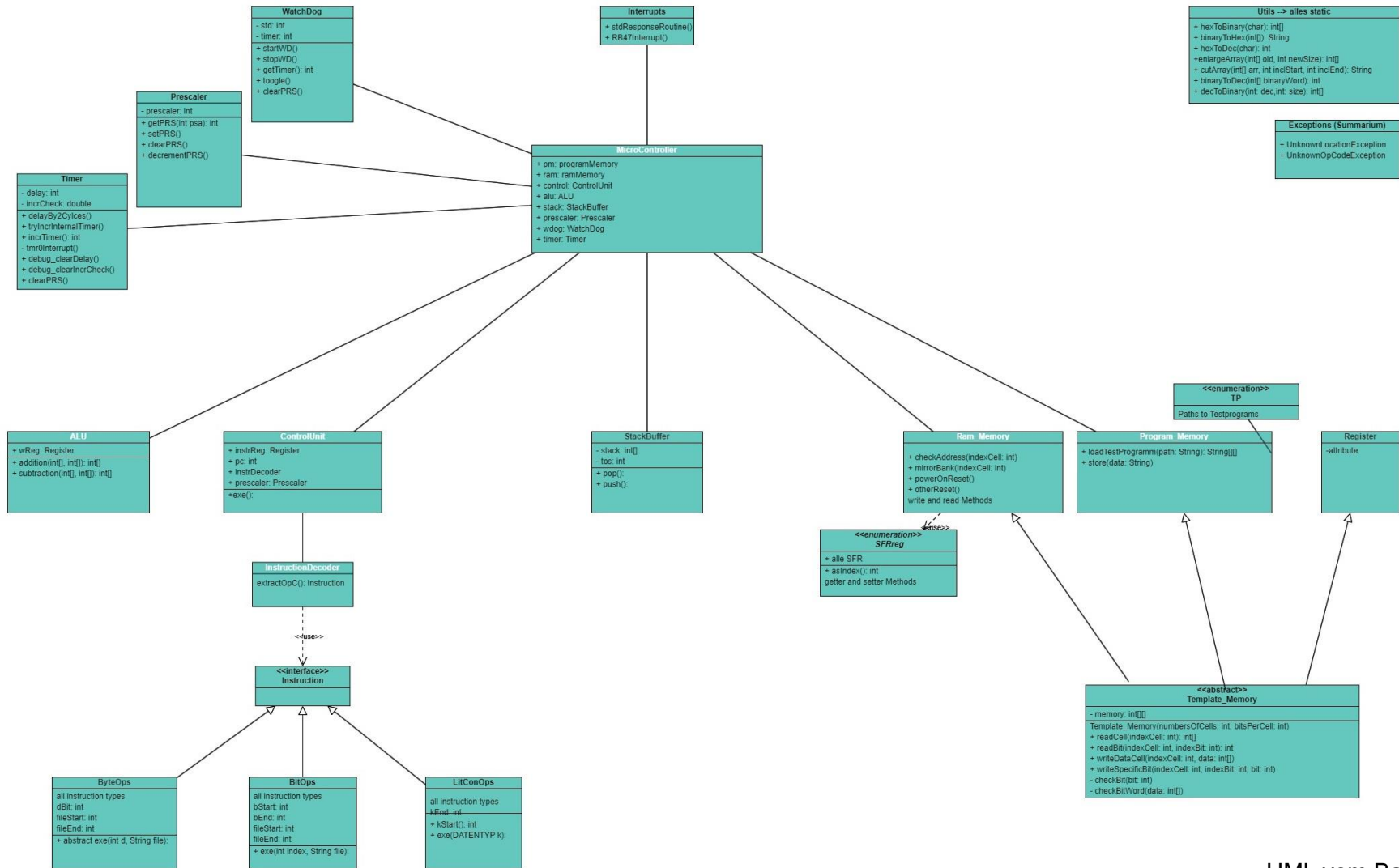
Zudem gibt einem der große Zeitaufwand dieses Projektes auch einen Einblick, wie es in der Realität später einmal sein wird, an größeren Projekten mitzuarbeiten und etwas Woche für Woche auszubauen sowie zu verbessern.

Wir kamen in Kontakt mit vielen verschiedenen Programmen, gerade im Bereich der Organisation. Aus anfänglichen Arbeitsaufteilungen in "Asana" wurde eine TODO-Liste von Microsoft, da diese für unsere Zwecke passender war.

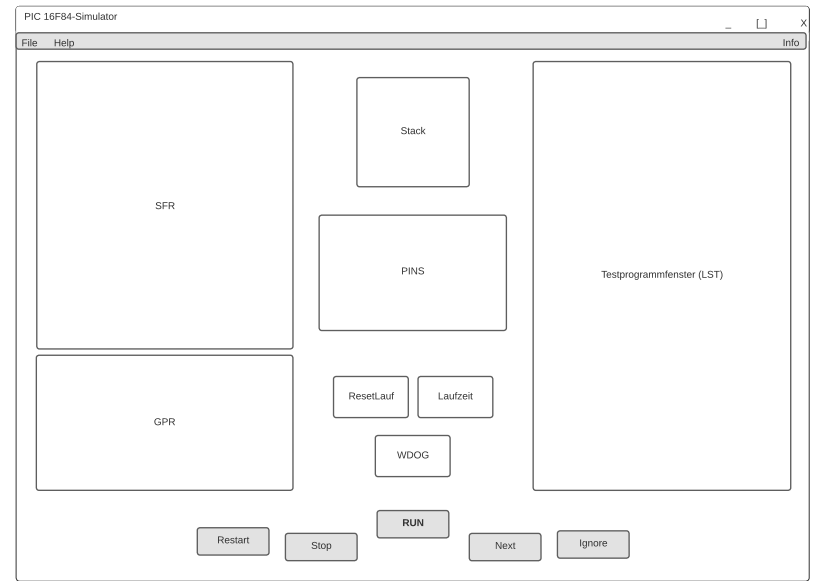
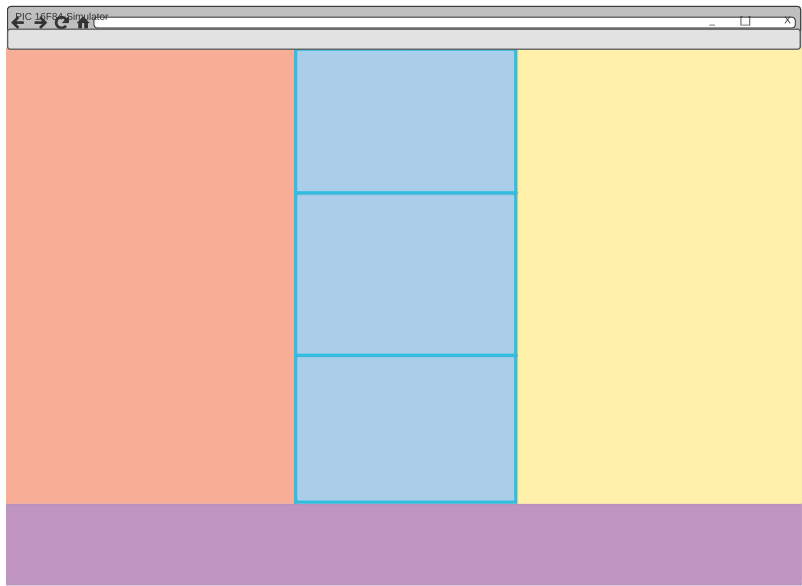
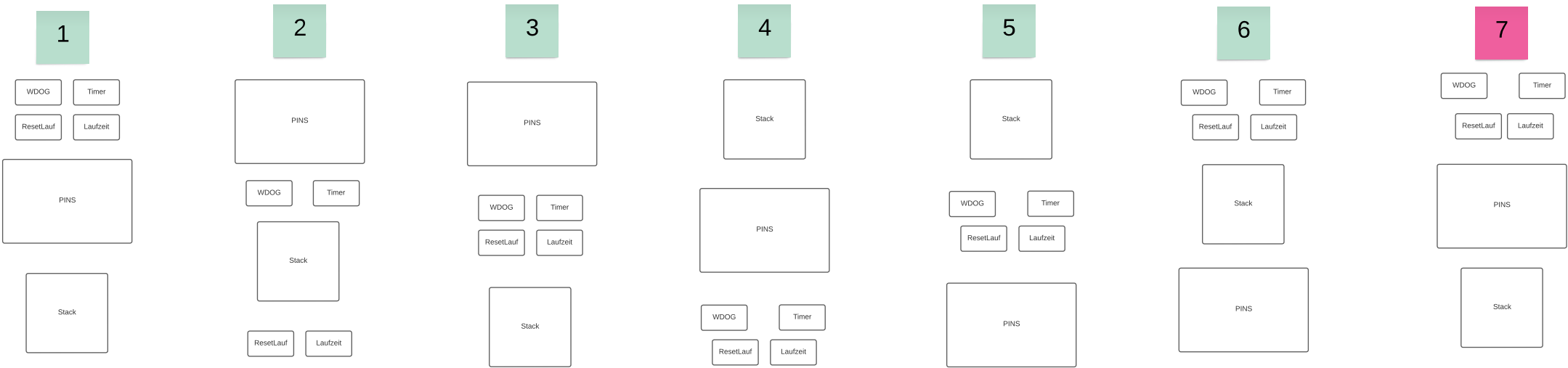
Auch unser Discord-Server ist erst mit der Zeit entstanden und wurde immer weiter ausgebaut, mit vielen verschiedenen Bereichen, in denen sich Informationen zu einem gewissen Thema finden.

Selbstverständlich wurde durch das Simulatorprojekt unser Wissen über den PIC und Mikrocontroller im Allgemeinen auf ein neues Level gehoben. Nachdem wir nun genau verstehen, was sich im Detail in solch einem Gerät abspielt, können wir dieses Wissen auf andere Themen beim Programmieren anwenden, um möglichst effektiven Code zu schreiben und neue Themen leichter verstehen zu können.

Das Simulatorprojekt zum PIC16F84 war also eine sehr wertvolle Erfahrung für uns !



UML vom Backend-Bereich



3/8

2/8


3/8

3/8

2/8

3/8

d

 PIC16F84 Simulator
Programm laden... Help

SFR

GPR

- ❑ WatchDog

WatchDog

Timer

Pins

Re...

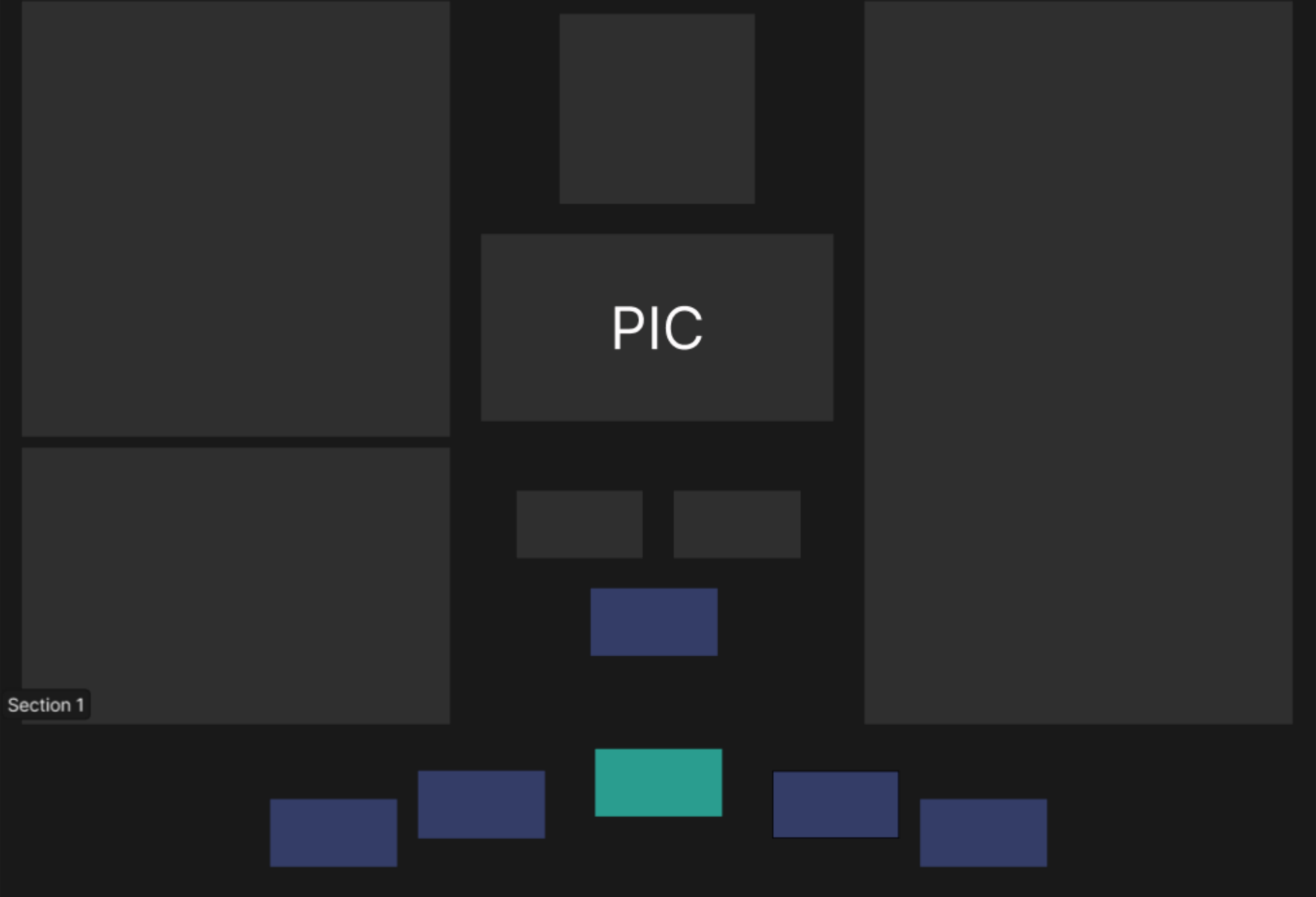
R...



N...

[illegible][illegible]

Ignore



Section 1

Register

Bit 7

Bit 6

Bit 5

Bit 4

Bit 3

Bit 2

Bit 1

Bit 0

0x00 INDF	0	0	0	0	0	0	0	0
0x01 TMR0	0	0	0	0	0	0	0	0
0x02 PCL	0	0	0	0	0	0	0	0
0x03 STATUS	0	0	0	1	1	0	0	0
0x04 FSR	0	0	0	0	0	0	0	0
0x05 PORTA	0	0	0	0	0	0	0	0
0x06 PORTB	0	0	0	0	0	0	0	0
0x08 EEDATA	0	0	0	0	0	0	0	0
0x09 EADDR	0	0	0	0	0	0	0	0
0x0A PCLATCH	0	0	0	0	0	0	0	0
0x0B INTCON	0	0	0	0	0	0	0	0
0x81 OPTION	1	1	1	1	1	1	1	1
0x85 TRISA	0	0	0	1	1	1	1	1
0x86 TRISB	1	1	1	1	1	1	1	1
0x88 EECON1	0	0	0	0	0	0	0	0
0x89 EECON2	0	0	0	0	0	0	0	0

Address

Bit 7

Bit 6

Bit 5

Bit 4

Bit 3

Bit 2

Bit 1

Bit 0

0x0C	0	0	0	0	0	0	0	0
0x0D	0	0	0	0	0	0	0	0
0x0E	0	0	0	0	0	0	0	0
0x0F	0	0	0	0	0	0	0	0
0x10	0	0	0	0	0	0	0	0
0x11	0	0	0	0	0	0	0	0
0x12	0	0	0	0	0	0	0	0
0x13	0	0	0	0	0	0	0	0
0x14	0	0	0	0	0	0	0	0
0x15	0	0	0	0	0	0	0	0
0x16	0	0	0	0	0	0	0	0
0x17	0	0	0	0	0	0	0	0
0x18	0	0	0	0	0	0	0	0
0x19	0	0	0	0	0	0	0	0
0x1A	0	0	0	0	0	0	0	0
0x1B	0	0	0	0	0	0	0	0
0x1C	0	0	0	0	0	0	0	0
0x1D	0	0	0	0	0	0	0	0

Bank0 == Bank1

STATUS

IRP

RP1

RP0

TO

PD

Z

DC

C

--	--	--	--	--	--	--	--	--

OPTION

RBPV

INTEDG

TOCS

TOSE

PSA

PS2

PS1

PS0

--	--	--	--	--	--	--	--	--

INTCON

GIE

EEIE

TOIE

INTE

RBIE

TOIF

INTF

RBIF

--	--	--	--	--	--	--	--	--

Stack

-->

0

<--

0

0

0

0

0

0

0

0

W-Reg

0

VT (T)

1

VT (W)

128

I/O

1

1

0

0

0

1

1

1

1

PIN

0

0

0

0

0

0

0

0

0

1 RA2

2 RA3

3 RA4/T0CKI

4 MCLR

5 Vss

6 RB0/INT

7 RB1

8 RB2

9 RB3

18 RA1

17 RA0

16 OSC1/CLKIN

15 OSC2/CLKOUT

14 VDD

13 RB7

12 RB6

11 RB5

10 RB4

I/O

1

1

1

0

0

1

1

1

1

PIN

0

0

0

0

0

0

0

0

0

0 μs

Reset Counter

WatchDog

RUN

Stop

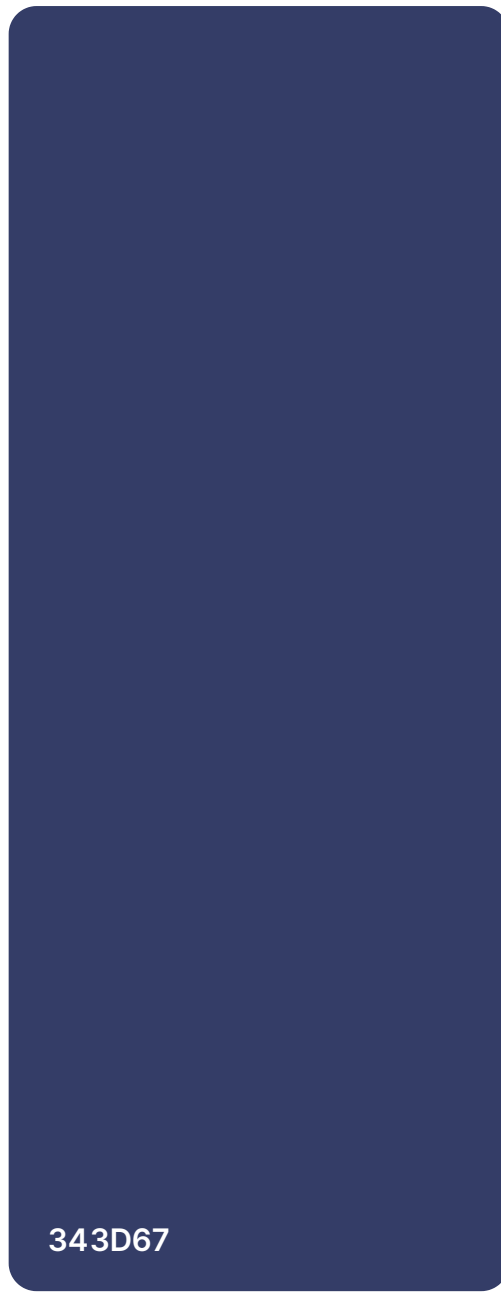
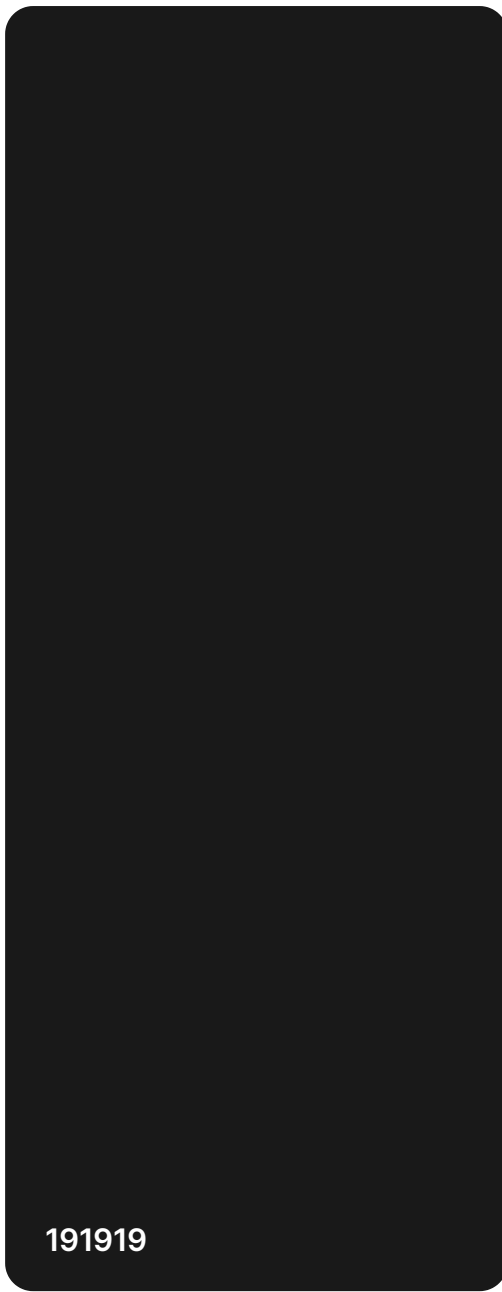
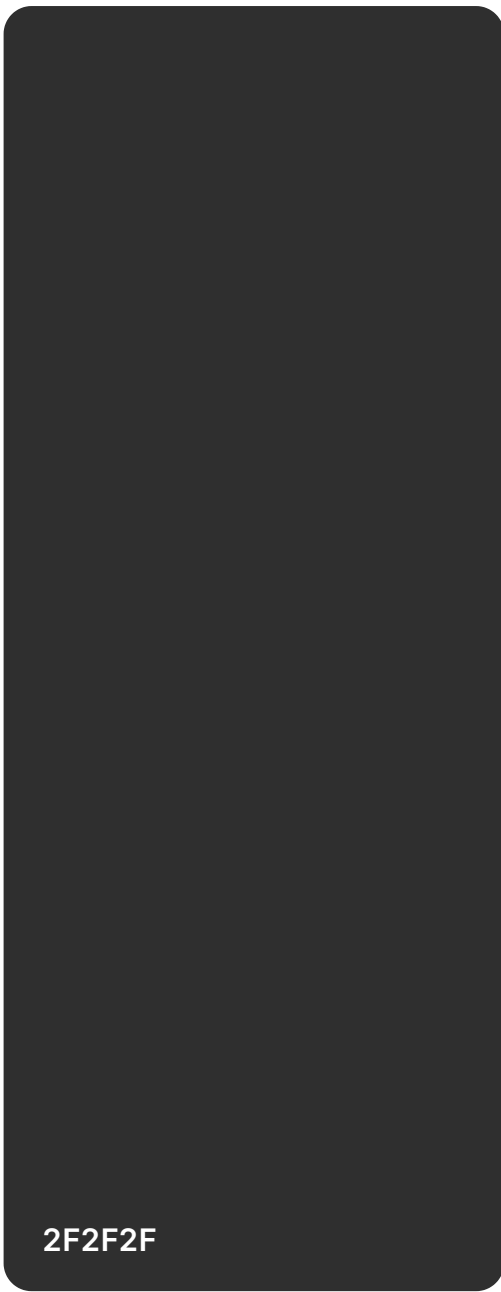
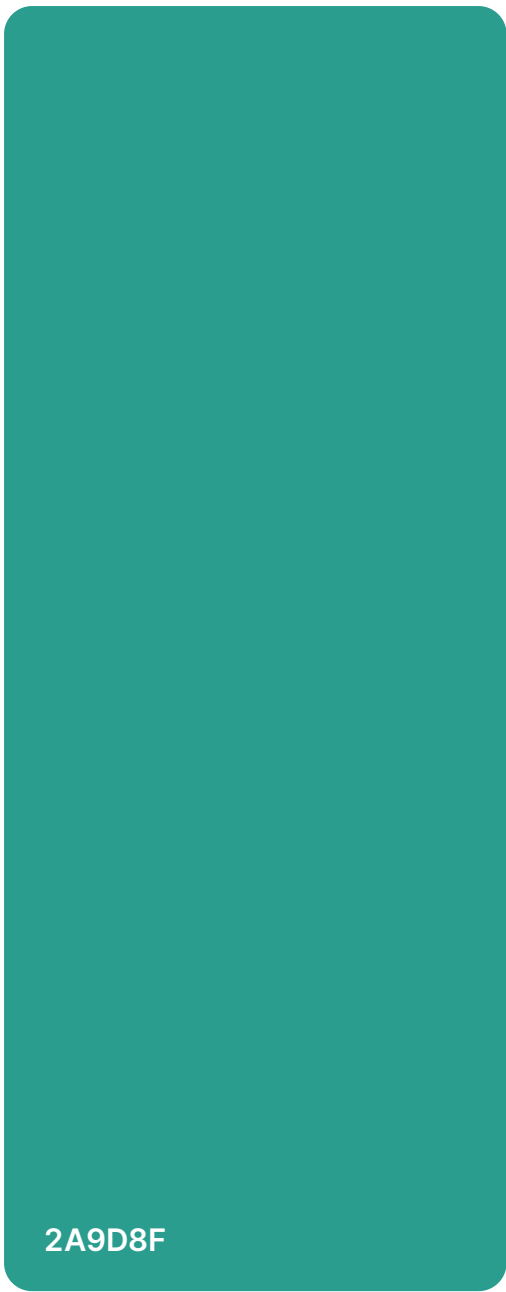
Next

Ignore

Restart

Testprogramm

00001	;TPicSim1
00002	;Programm zum Test des 16F84-Simulators.
00003	;Es werden alle Literal-Befehle geprüft
00004	;(c) St. Lehmann
00005	;Ersterstellung: 23.03.2016
00006	;19.05.2020
00007	;mod. 18.10.2018 Version HSO
00008	;
00009	list c=132 ;Zeilenlänge im LST auf 132 Zeichen setzen
00010	
00011	
00012	;Definition des Prozessors
00013	device 16F84
00014	
00015	;Festlegen des Codebeginns
00016	org 0
00017	start
0000 3011	00018 movlw 11h ;in W steht nun 11h, Statusreg. unverändert
0001 3930	00019 andlw 30h ;W = 10h, C=x, DC=x, Z=0
0002 380D	00020 iorlw 0Dh ;W = 1Dh, C=x, DC=x, Z=0
0003 3C3D	00021 sublw 3Dh ;W = 20h, C=1, DC=1, Z=0
0004 3A20	00022 xorlw 20h ;W = 00h, C=1, DC=1, Z=1
0005 3E25	00023 addlw 25h ;W = 25h, C=0, DC=0, Z=0
	00024
	00025
	00026 ende
0006 2806	00027 goto ende ;Endlosschleife, verhindert Nirwana
	00028
	00029



PIC16F84 Simulator Color Palette