Anzeigetafel für ein Fußballspiel

Daniel H. Vogt und Lukas Feltes

Matrikelnummer 9552394 und 9584715

Dokumentation eines 8051 Projekts

eingereicht im Rahmen der
Vorlesung *Systemnahe Programmierung*
des Studiengangs

Angewandte Informatik

an der DHBW Karlsruhe

im Juni 2018

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, am 14. Juni 2018

Daniel H. Vogt und Lukas Feltes

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Dieses Projekt findet im Rahmen der Vorlesung Systemnahe Programmierung an der Dualen Hochschule Baden-Württemberg statt. Ziel dieser Vorlesung ist es Kenntnisse über die systemnahe Programmierung zu vermitteln und so das Schreiben systemnaher Programme zu lernen. Innerhalb der Vorlesung wird Assembler für die 8051 Mikrokontroller verwendet. Um die erlernten Grundlagen dieser Vorlesung auch praktisch einsetzen zu können, wird ein Projekt erstellt und in einer virtuellen Entwicklungsumgebung ausgeführt. Dafür wurde die MCU-8051 IDE verwendet.

Bei der Gestaltung des Projektes können die Studenten eigene Ideen umsetzen. Aufgrund unserer eigenen großen Sportleidenschaft haben wir uns dafür entschieden eine Fußballanzeigetafel mit Spielstand und Zeitanzeige zu implementieren.

1.2 Aufgabenstellung

Die zu implementierende Anzeigetafel sollte sowohl den aktuellen Spielstand, sowie die bereits gespielte Zeit anzeigen (siehe Abb. 1.1). Dabei soll die Zahl der erzielten Tore oder Punkte über angeschlossene Knöpfe steuerbar sein. Optional zu diesen Implementierungen soll zusätzlich eine Hupe die bei Ablauf der Spielzeit ertönt implementiert werden. Der aktuelle Spielstand und die Zeit soll per Knopfdruck zurück setzbar sein.

Für die Implementierung der Anzeigetafel werden nur die innerhalb der Entwicklungsumgebung vorgegebenen Hilfsmittel genutzt.

Abbildung 1.1: Abbildung Anzeigetafel https://pimage.sport-thieme.de/detail-fillscale-min-height/stramatel-anzeigetafel-ffbffc/131-5400

Kapitel 2

Grundlagen

2.1 Assembler

Der erste bekannte Assembler wurde zwischen 1948 und 1950 geschrieben.

Eine Assemblersprache, kurz auch Assembler genannt, ist eine Programmiersprache, die auf den Befehlssatz einer bestimmten Prozessorarchitektur ausgerichtet ist. Assemblersprachen sind maschinenorientierte Programmiersprachen. Anstelle eines schwer verständlichen Binärcodes der Maschinensprache können Befehle durch verständliche Symbole in Textform dargestellt werden. Zur Übersetzung eines Assemblerprogramms in Maschinencode wird eine Übersetzungssoftware genutzt. Der Quelltext in Assemblersprache wird auch als Assemblercode bezeichnet.

Die verschiedenen Computerarchitekturen nutzen dabei eigene Maschinensprachen und damit auch eigene Assemblersprachen. Diese Sprachen unterscheiden sich der verschiedenen Architekturen in Anzahl und Typ der Operationen. Jedoch haben alle Architekturen dieselben grundlegenden Operationen. [1]

2.2 Der 8051 Mikrocontroller

Die **8051**-Familie der Mikrocontroller ist eine Prozessorarchitektur von Intel. Sie ist mittlerweile im Originalen veraltet, aber es gibt weitere Varianten, die teilweise durchaus auf aktuellem Stand sind.

Der 8051 ist ein Mikrocontroller bei dem Befehls- und Datenspeicher logisch getrennt sind, auch wenn diese über einen einzigen Bus adressiert werden. Ob es sich dabei um eine Harvard oder eine von Neumann-Architektur handelt, ist umstritten.[2]

Ein paar der verwendeten Befehle des 8051 Befehlssatzes sind folgende:

```
1  mov A, R7  ;Register in Akku laden
2  mov R0, A  ;Akku in Register laden
3  setb C   ;Set C - for demo purpose
```

```
4  JB 042d, bc2 ;springe bei gesetzer BAdr
5  jmp @A+DPTR  ;Brechneter jump
6  ret    ;reti for interrupt
7
8
```

[3]

2.3 Entwicklungsumgebung MCU-8051 IDE

In Zeiten von Emacs, Vim (und Intellij) ist es schwer sich in eine IDE einzufinden, welche viele Bugs hat und langsam arbeitet. Glücklicherweise ist man nicht gezwungen in dieser IDE zu programmieren. Es reicht wenn man dieses Werkzeug nur zum Compilieren und Simulieren nutzt.

Der Editor dieser IDE unterstützt einen nur sehr wenig beim entwickeln. Die Autovervollständigung ist mangelhaft, da Symbole teilweise nicht gefunden werden. Die Vorschläge für die Vervollständigung erfolgen oft ohne den Kontext oder sind unzureichend vollständig.

Das Laden und Erstellen von virtueller Hardware ist zu kompliziert und langsam. Die virtuelle Hardware ist mit Abstand das langsamste an der IDE. Konkurrenzprodukte sind um ein Mehrfaches schneller. Komplexe Programme lassen sich damit nur langsam simulieren, große Timer-Intervalle sind zu vermeiden (da es Tage dauern könnte, bis es zu einem Ereignis kommen könnte). Der “Fade out Intervall„ der virtuellen Hardware ist unberechenbar und unabhängig von der Ausfuhrgeschwindigkeit und somit nicht sehr realitätsnah.

Gut an der IDE sind die gut strukturieren Anzeigen. So lassen sich schnell die Registerinhalte und andere relevante Hardwareinformation auf einen Blick erkennen. Auch die Möglichkeit das Programm Schrittweise oder auch rückwärts ablaufen zu lassen ist hilfreich.

Kapitel 3

Konzept

3.1 Analyse

Zu Beginn des Spiels muss die Zeitanzeige auf null gesetzt werden, also 00:00 anzeigen. Auch die Anzeige für den Spielstand muss null zu null gesetzt werden. Das Spiel und somit der Ablauf der Zeit soll dann per Knopfdruck gestartet werden. Die aktuelle Zeit sowie der Spielstand sollten dabei stets im Hauptspeicher hinterlegt sein. Neben der Option das Spiel per Knopfdruck zu starten, soll man die Zeit anhalten können. Zudem können beide Anzeigen zurück auf 0 gesetzt werden. Auch das Zählen der Tore ist über die Steuerung möglich. Dabei kann die Zahl der Tore sowohl um eins erhöht als auch um eins erniedrigt werden.

Zur Anzeige der Zahlenwerte dienen 7 Segment Anzeigen. Diese dienen jeweils als Pärchen als verschiedene Anzeigen. Die Aufgaben eines Pärchens ist einmal Anzeigen eines Spielstandes oder das anzeigen der aktuellen Zeit.

Zur Steuerung wird ein Simple Keypad genutzt. Dieses hat acht Knöpfe, die jeweils mit den oben beschriebenen Funktionen belegt werden und so die Anzeigen steuern können.

3.2 Programmentwurf

Die Ergebnisse der Analyse sollen nun in einen Programmentwurf umgesetzt werden. Zu Beginn werden dafür den verschiedenen Elementen, der in der Analyse beschrieben Hilfsmittel, bestimmte Speicherbereiche zugewiesen.

Tabelle 3.1: My caption

Speicheradressbereich	Inhalt
60h-63h	Punkte
64h-67h	Zeit
2Fh	Funktionen

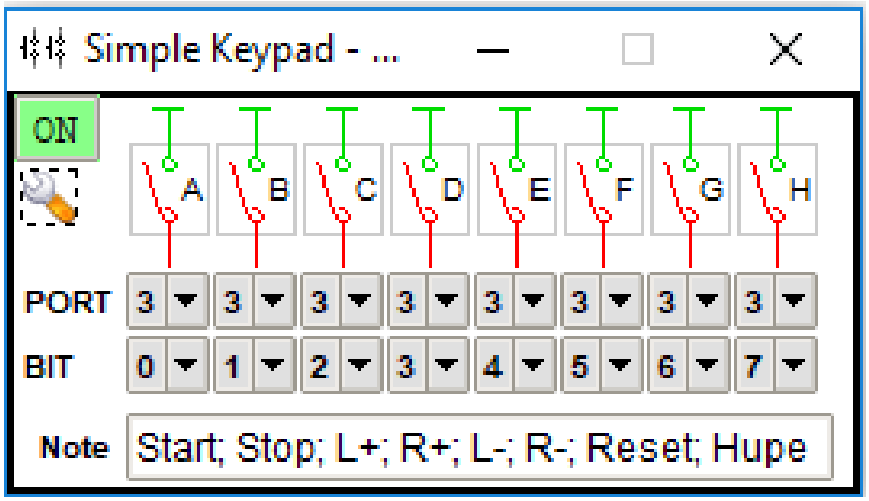


Abbildung 3.1: Abbildung Simple Keypad Eigener Screenshot

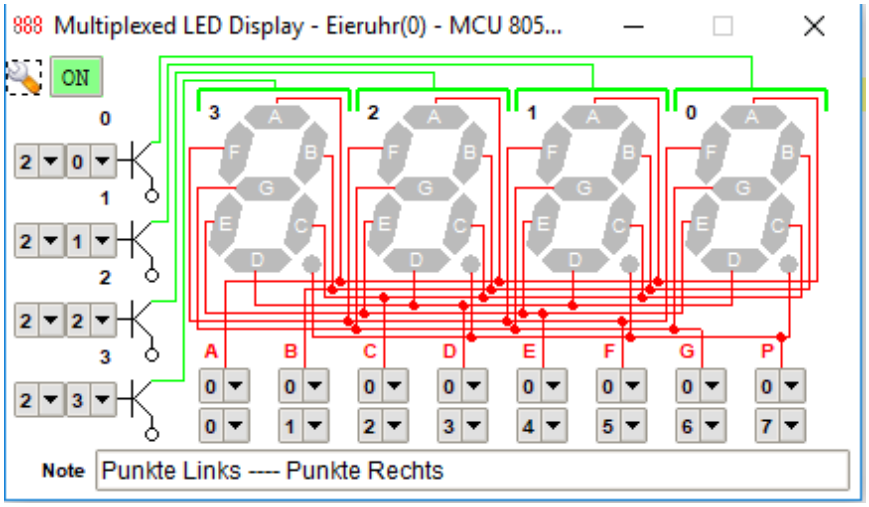


Abbildung 3.2: Abbildung Spielstand Eigener Screenshot

Neben der Festlegung der Speicherbereiche, wird die Hardware über die I/O Ports verbunden. Dabei werden die folgenden Ports für die jeweiligen Anzeigen genutzt:

- Minuten und Sekunden anzeige für ein Fußballspiel (Port0)
- Punkte Anzeige für jedes Team (Port1)
- Ansteuerung der jeweiligen 7 Segmentanzeige (Port2)
- Simple Keypad zur Steuerung (Port3)

Wenn das Programm aufgerufen wird, kann der Ablauf der Zeit mit dem

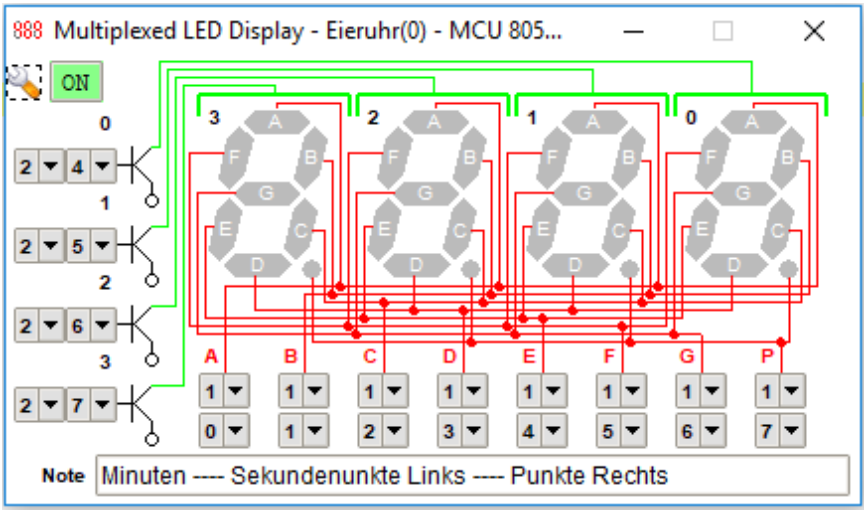


Abbildung 3.3: Abbildung Spielzeit Eigener Screenshot

Simple Keypad eingeleitet werden. Dafür muss der erste Knopf auf dem Keypad einmal gedrückt werden. Er wird somit geschaltet und nimmt den Wert 0 an. Anschließend muss er noch einmal betätigt werden, um die Zeit zu starten, da nur beim Schalten von 0 auf 1 eine Veränderung des Wertes festgestellt wird. Das gleiche gilt für die anderen Knöpfe des Keypads. Auch sie müssen je von 0 auf 1 geändert werden um eine Veränderung zu bekommen.

Um das Hochzählen der Zeit genau kontrollieren zu können wird ein Timer implementiert der während dem Ablauf des Programmes automatisch hochzählt. Die Zahl wie oft der Timer hochzählt, ist dabei so gewählt, dass genau eine Sekunde vergeht bevor die Anzeige noch einmal hochzählt. Dieser Wert ist aufgrund der Geschwindigkeit der IDE bei unserem Programm kleiner gewählt.

Das Hauptprogramm läuft eigentlich unabhängig vom Timer selbst. Allerdings kommt es sobald der Timer fertig hochgezählt hat, es also zu einem Überlauf kommt, zu einem Interrupt, durch den das Programm unterbrochen wird. Erst nachdem dieser Interrupt vom Timer beendet wird, kann das Programm weiterlaufen.

Kapitel 4

Implementation

Im folgenden Kapitel wird die Implementation des Programmes beschrieben. Da eine komplette Beschreibung des Codes den Rahmen sprengen würde, werden hier nur ausgewählte Abschnitte des Programmes beschrieben.

Zu Beginn des Programmes werden die beschrieben Speicheradressen festgelegt und den Variablen Werte zugewiesen. Im Anschluss daran wird der Timer initialisiert.

Zu Beginn des Ablauf des Hauptprogrammes werden alle Zahlen der 7 Segmentanzeigen neu gezeichnet. Dieses Zeichnen wird im Folgenden genauer beschrieben.

```
 1 zeigeAlleSemente:
 2 push 00h  ;Sichere R0
 3 push 01h  ;Sichere R1
 4 push 02h  ;Sichere R2
 5 mov R0, #00d  ;aktuelles segment
 6 mov R1, #00d  ;Pointer für wert von segment
 7
 8 mov A, #01d
 9 cpl A
10 mov R2, A ;selectiertes segment... wird rotiert
11
12 ;clear all 7segs
13 mov P2, #00h
14 mov P0, #0FFh
15 mov P1, #0FFh
16 mov P2, #0FFh
17
18 loopAlleSegmente:
19 CJNE R0, #08d, doFuerSegment     ;für alle 8 segmente
20 pop 02h   ;Wiederherstellen von R2
21 pop 01h   ;Wiederherstellen von R1
22 pop 00h   ;Wiederherstellen von R0
23 ret
24
25 doFuerSegment:
26 ; Lade Segment mit entsprechendem Wert
```

```
27 mov p2, R2  ;aktivate segment mit P2
28
29
30
31 mov A, R0 ;get wert für
32 ;60h start adresse für die Sement werte s.o.
33 Add A, #060h  ;aktuelles Segment?
34 mov R1, A
35 mov A, @R1
36
37 mov dptr, #numbers  ;hole richtigs 7Seg code
38 movc a,@a+dptr
39 mov p0, a ;gib 7seg code auf P0 aus
40 mov p1, a ;gib 7seg code auf P1 aus
41
42 mov p2, #0FFh ;deaktivte 7Seg
43 mov p1, #0FFh ;deaktivte 7Seg
44 mov p0, #0FFh ;deaktivte 7Seg
45
46
47 ; Select nächstes Segment
48 mov A, R2
49 RL A
50 mov R2, A
51
52 ;continue loop
53 Inc R0
54
55 ljmp loopAlleSegmente
```

Zu Beginn des Aufrufs von ZeigeAlleSegemente (4) werden die Register 0 bis 2 auf dem Stack gespeichert. In die Register 0 und 1 wird 0 geschrieben. Anschließend wird dann der Wert 1 in den Akku geladen. Anschließend wird das Komplementär dieses Wertes gebildet. Somit sind alle Stellen der 8 bit Zahl außer der rechten 1. Die Binär-Zahl ist also 11111110. Dieser Wert wird in das Register 2 geladen.

Nach dieser Initialisierung findet ein Loop durch die 8 Stellen der Anzeige statt. Dafür wird anfangs verglichen ob schon bereits alle Werte neu geschrieben wurden, indem der Wert von Register 0 mit 8 verglichen wird. R0 speichert also die jeweilige Stelle die adressiert wird. Zu Beginn eines Ablaufes, wird das passende Segment mithilfe des in Register 2 gespeicherten Wertes ausgewählt und so auf 0 gesetzt. Anschließend wird mithilfe des Registers 0 der passenden aktuelle Wert aus dem Speicher ausgewählt. Das passiert in dem der Wert aus R0 um 60 erhöht wird, so adressiert er die passende Speicherstelle. Im Anschluss wird der Wert aus der Datenbank ausgelesen. Anschließend werden alle Anzeigen wieder deaktiviert, damit die Werte neu gezeichnet werden können.

Am Ende eines Durchlaufes wird der in R2 gespeicherte Wert nach links rotiert, so dass die 0 innerhalb der Zahl um eine Stelle nach links rückt und beim nächsten Mal die passende Zahl in der Anzeige im nächsten Durchlauf

ausgewählt wird. Zudem wird der "Counter"Register 0 um eins erhöht.

Nach dem Ablauf des Neu Zeichnens der Zahlen, finden Überprüfungen statt, ob einer der Knöpfe gedrückt worden ist. Dabei werden nur dann Aktionen ausgeführt, wenn der Knopf gedrückt wurde. Eine wichtige Funktion ist dabei die passende Auswahl des Spielstandes. Dabei wird wenn der Knopf gedrückt wurde, folgende Funktion aufgerufen:

```
1  countupPunkte:
2  push 00h
3  push 01h
4  cjne R1, #00d, runtercount
5  ;Count Hoch
6  INC @R0
7  cjne @R0, #010d, fertigcountPunkte
8  mov @R0, #00d
9  INC R0
10 INC @R0
11 cjne @R0, #10d, fertigcountPunkte
12 mov @R0, #00d
13 DEC R0
14 mov @R0, #00d
15 ljmp fertigcountPunkte
16
17
18 runtercount:
19 ;Count Runter
20 DEC @R0
21 cjne @R0, #0FFh, fertigcountPunkte
22 mov @R0, #09d
23 INC R0
24 DEC @R0
25 cjne @R0, #0FFh, fertigcountPunkte
26 mov @R0, #09d
27 DEC R0
28 mov @R0, #09d
29
30
31 fertigcountPunkte:
32 pop 01h
33 pop 00h
34 ret
```

Zu Beginn des Aufrufs der Funktion werden die Werte aus Register 0 und 1 auf den Stack abgelegt. Anschließend wird überprüft, ob der Wert hoch oder runtergezählt werden soll. Dies geschieht durch einen Abgleich mit Register 1 indem vor Aufruf der Funktion der Wert 0 bzw. 1 gespeichert wurde. Abhängig davon wird nach einem Abgleich die passende Methode aufgerufen. Innerhalb der Mehtoden wird dann der Wert auf der Anzeige um eins erhöht. Dabei muss darauf geachtet werden, dass der Wert passend erhöht wird und nicht ein falscher Wert angezeigt wird. Dabei wird beim Herunterzählen geprüft ob die Zehner und Einerstelle passend (nicht größer

als 9) ist. Gleichartig wird beim Runterzählen darauf geachtet, dass der Wert nicht kleiner als 0 wird.

Neben diesen Überprüfungen gibt es weitere Funktionen, die die Zeit anhalten und alle Werte auf 0 setzen.

Kapitel 5

Zusammenfassung

Zusammenfassend lässt sich festhalten, dass das Projekt erfolgreich umgesetzt wurde. Die geplanten Maßnahmen zur Steuerung der Anzeigetafel sind dabei umgesetzt worden. Die Anzeigetafel zeigt die abgelaufene Zeit und den aktuellen Spielstand an. Zudem können über das Keypad gewünschte Befehle ausgeführt werden. Bei der Implementation wurden weiterhin mögliche Fehler wie die Anzeige einer falschen Zeit oder ein negativer Spielstand beachtet und in der Implementierung abgefangen.

Insgesamt lässt sich also festhalten, dass das Projekt Kenntnisse über Assembler vermittelt hat und die Erstellung eines Assemblerprogramms somit gelungen ist.

Quellenverzeichnis

Online-Quellen

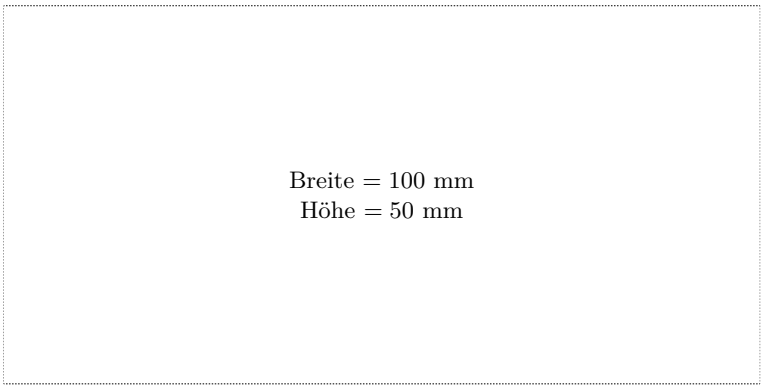
[1] URL: https://de.wikipedia.org/wiki/Assemblersprache (besucht am 14. 06. 2018) (siehe S. 3).

[2] URL: https://www.mikrocontroller.net/articles/8051 (besucht am 14. 06. 2018) (siehe S. 3).

[3] URL: https://moodle.dhbw.de/pluginfile.php/7731/mod_resource/content/0/FOSA--8051-Lokal-V31-5-05.pdf (besucht am 14. 06. 2018) (siehe S. 4).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —