Dokumentation Projektarbeit

Bombenspiel

Für die Prüfung zum
Bachelor of Science

Studiengang Informatik
Studienrichtung Angewandte Informatik
Duale Hochschule Baden-Württemberg Karlsruhe

Von
Jonas Kümmerlin, Simon Knab, Sandra Kramlich, Marco Herglotz

Abgabedatum:	22. Juni 2017
Kurs:	TINF15B4

Eidesstattliche Erklärung

Erklärung gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2015. Wir versichern hiermit, dass wir diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen als Hilfsmittel benutzt haben.

Karlsruhe, den 22. Juni 2017

Jonas Kümmerlin, Simon Knab, Sandra Kramlich, Marco Herglotz

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte vorliegt.

Copyright-Vermerk

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Motivation der Arbeit war das Vertiefen der vermittelten, theoretischen Inhalte. Erwartet wurde kein umfangreiches Projekt, sondern nur ein kleines, auf einem 8051 Mikrocomputer lauffähiges Programm, welches ein wenig über den vermittelten Lernstoff hinaus reicht und damit das Wissen um Assembler weiter festigt und vertieft.

1.2 Aufgabenstellung

Die Aufgabe war es mit dem erworbenen Wissen aus der Vorlesung „Systemnahe Programmierung“ ein kleines Programm zu entwickeln. Dieses soll dann auf einem 8051 Mikrocomputer mit simulierter Hardware laufen. Die Simulation selbst wird von der verwendeten Entwicklungsumgebung MCU-8051 IDE bereitgestellt. Der 8051 und die IDE werden in Kapitel 2.2 und 2.3 genauer vorgestellt.

Wir als Gruppe haben uns das Ziel gesetzt, das bekannte „Bombenspiel“ zu entwickeln. Hierbei wird eine Bombe mit zufälliger Zündschnur zwischen zwei Spielern hin und her geworfen. Der Spieler, bei dem die Bombe explodiert, hat verloren. Das genaue Konzept zu unserer Idee ist in Kapitel 3 ausführlich erklärt.

2 Grundlagen

2.1 Assembler

Assembler ist eine Programmiersprache der zweiten Generation und ist ein direkter Nachfolger der Programmierung in Maschinensprache, also mit Zahlencodes. Bei Assembler handelt es sich um eine hardwarenahe Programmiersprache. Je nachdem welche Hardware verwendet wird, stehen unterschiedliche Befehlssätze zur Verfügung. Daher gibt es nicht die „eine“ Programmiersprache Assembler, sondern es handelt sich um eine Gruppe von Assemblersprachen, die für die jeweilige Hardware angepasst ist.[1]

Generell handelt es sich bei Assembler um eine zeilenorientierte Sprachfamilie, das heißt eine Zeile stellt immer eine komplette Anweisung dar. Diese Anweisungen bestehen aus den hardware-spezifischen Befehlen.[2]

2.2 Der 8051 Mikrocomputer

Der 8051 Mikrocomputer wurde 1981 von Intel als 8-bit Mikrocontroller auf den Markt gebracht. Es handelt sich um ein „system on a chip“, was bedeutet, dass RAM, ROM, zwei Timer, ein Serial Port, 4 8-bit Ports und noch einiges mehr auf einem einzigen Chip verbaut sind. Die einzelnen Bestandteile sind in Abbildung 2.1 dargestellt.[3]

Mittlerweile ist der Original 8051 veraltet, dennoch existieren viele Abwandlungen (Derivate) beziehungsweise Weiterentwicklungen des 8051. Diese sind allerdings mit höherer Taktfrequenz und geringerer Taktteilung ausgestattet, sodass sie nahezu auf dem aktuellen Stand der Technik und damit deutlich leistungsfähiger als das Ursprungsmodell sind.[4]

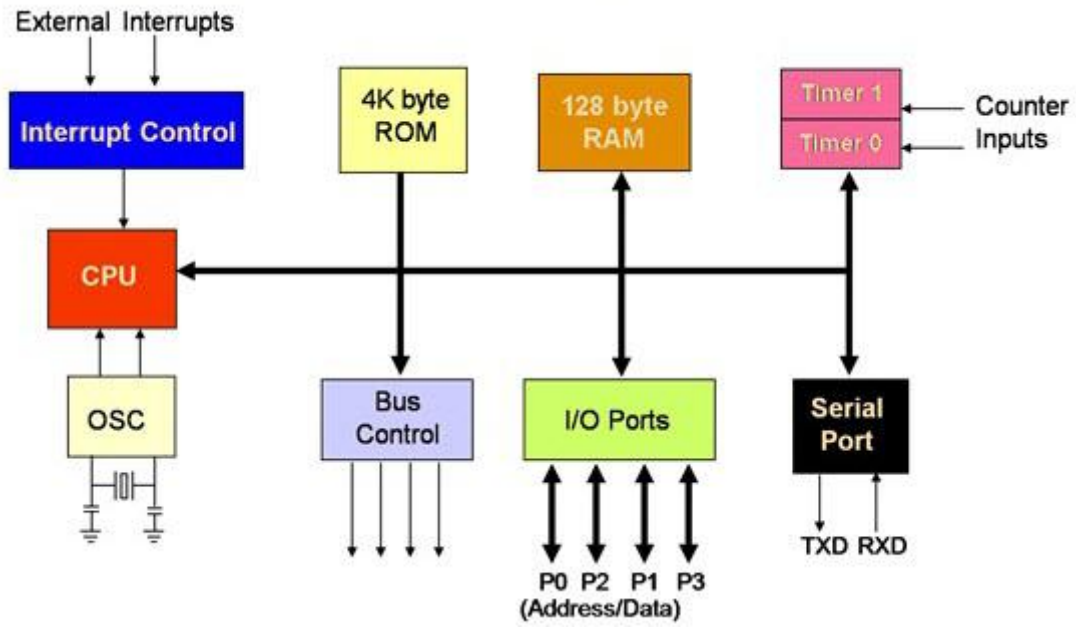


Abbildung 2.1: Blockdiagramm eines 8051 Mikrocomputers
Quelle: `https://www.tutorialspoint.com/embedded_systems/images/block_diagram.jpg`

2.3 Entwicklungsumgebung MCU-8051 IDE

Die MCU-8051 IDE ist eine integrierte Entwicklungsumgebung für Mikrocontroller und basiert auf dem 8051 Mikroprozessor. Neben Assembler können die Programme auch in C entwickelt werden.

Die IDE unterstützt den Entwicklungsprozess neben einem Quellcode Editor auch mit einem Simulator zum direkten Ausführen des Programms auf verschiedenen Mikrocontrollern. Auch eine Hardware-Anbindung in Form eines LCD Displays, eines Keypads und vielen weiteren Tools wird von der MCU-8051 IDE bereitgestellt.[5]

3 Konzept

3.1 Analyse

Anfangs soll auf dem Display eine Willkommensnachricht und die bisherige Spielstatistik angezeigt werden. Der Spielablauf beginnt dann mit dem Drücken des Knopfs (im folgenden als Buzzer bezeichnet). In diesem Moment wird die Bombe einem zufälligen Spieler gegeben, der sie dann durch Drücken des Buzzers dem anderen Spieler zuwerfen kann. Nach einer zufälligen Zeit explodiert die Bombe und der Spieler, der sie aktuell hat, verliert diese Runde. Danach wechselt das Spiel zurück in den Startbildschirm und die Statistik wird entsprechend angepasst.

Um obigen Programmablauf realisieren zu können, werden die folgenden Komponenten benötigt:

- Timer, für die „Länge“ der Zündschnur
- Zufallsgenerator, für die „Länge“ der Zündschnur
- LCD Display, zur Darstellung des aktuellen Spielzustandes
- Keypad, genauer gesagt ein einziger Knopf zur Steuerung des Spielverlaufs

3.2 Programmentwurf

Zur technischen Umsetzung ist der Programmablauf in Abbildung 3.1 geplant.

Zur Steuerung des Buzzers muss eine Interrupt Routine verwendet werden, um direkt beim Drücken des Buzzers eine Speicherstelle zu setzen. Würde der Buzzer nicht mit einem Interrupt realisiert, könnte es passieren, dass der Spieler den Buzzer bereits wieder losgelassen hat, bevor das Programm an der entsprechenden Verzweigung angekommen ist.

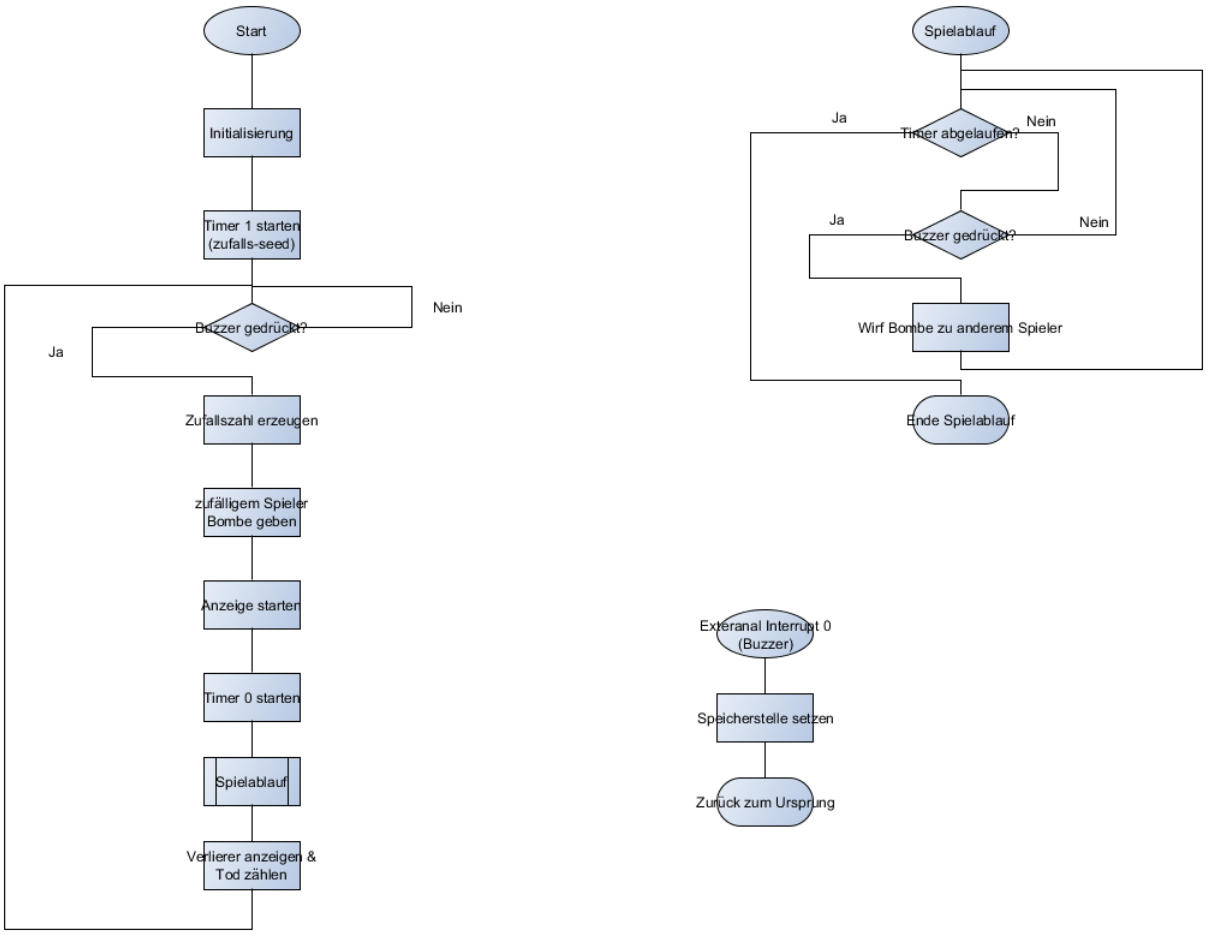
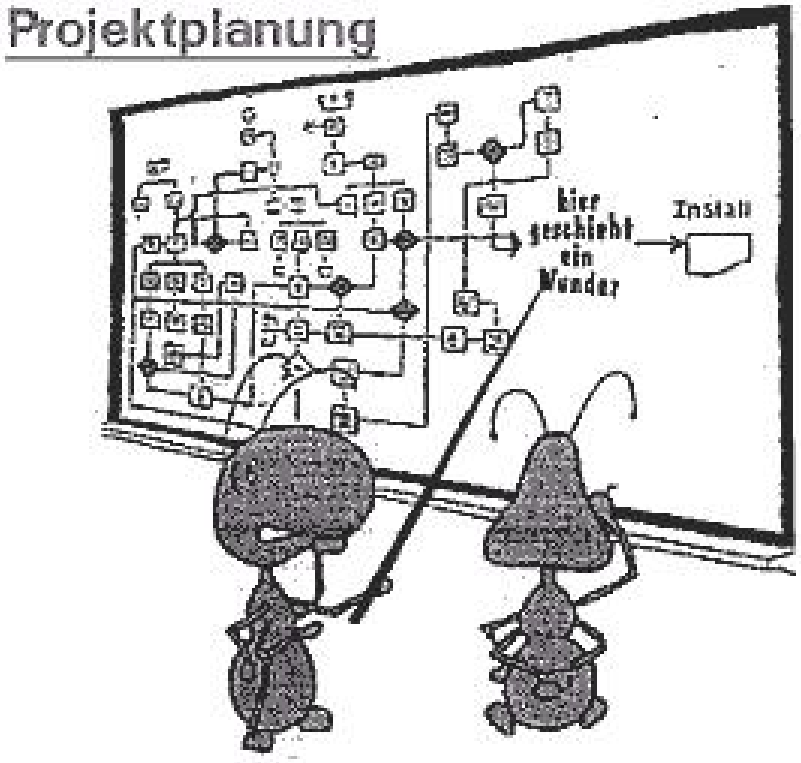


Abbildung 3.1: Flussdiagramm des Programms

4 Implementierung



Quelle: `http://www.informatik.uni-oldenburg.de/~sos/kurse09/img/projekte.jpg`

4.1 Buzzer

Der Buzzer wurde wie bereits erwähnt mit einer Interrupt Steuerung implementiert. Der Quellcode ist in Programmcode 4.1 abgebildet.

```
BUZZER_IR equ p3.2
BUZZER equ B.0

IRS_init:
setb EA          ;enable Interrupts
setb EX0         ;enable INT0
setb IT0         ;make INT0 edge triggered
ret

int0_handler:
setb BUZZER      ;set BUZZER
clr IE0
reti
```

Programmcode 4.1: Quellcode für den Buzzer Interrupt

4.2 Zufallsgenerator

Der Sprungpunkt `random_init` wird beim Starten des Programms aufgerufen um den Zufallsgenerator zu initialisieren. Für die erste erzeugte Zufallszahl wird der Wert von `Timer1` verwendet und mit dem Sprungpunkt `random_next_from_timer` angesprochen. Danach wird immer `random_next` verwendet, um eine auf der letzten Zufallszahl basierte neue Zufallszahl zu erhalten.

```
; Writes random number in RANDOM_NUM,
; uses last one as seed.
; 8-bit galois lsfr implemented according to wikipedia
random_next:
mov A, RANDOM_NUM

clr C
rrc A ; rotate right
jnc random_skipmask

xrl A,#10111000b        ; apply xor mask for x^8+x^6+x^5+x^4+1
random_skipmask:
mov RANDOM_NUM,A

ret

random_init:
mov RANDOM_NUM,#42

; setup timer 1 for random seed values
orl TMOD,#00100000b
anl TMOD,#11101111b
setb TR1

ret

random_seed_from_timer:
mov A,TL1
jnz random_seed_from_timer_nonzero      ; our LSFR won't work if all
   bits are zero
mov A,#0FFh
random_seed_from_timer_nonzero:
mov RANDOM_NUM,A
ret
```

Programmcode 4.2: Quellcode für die Zufallszahlen-Generierung

4.3 Timer

Der Timer musste deutlich komplizierter realisiert werden als gedacht. Das Spiel soll eine gewissen Zeit laufen, allerdings lassen die Timer des 8051 nur eine sehr geringe Zeitspanne zu, sodass wir dieses Problem umgehen mussten. Ein Ausschnitt des Timer-Codes ist in Programmcode 4.3 zu sehen.

```
timer_init:                 ; Subroutine to setup the timer
mov TL0,TIMER_RELOAD_VAL;   ; reset timer counter
mov TH0,TIMER_RELOAD_VAL+1;

anl TMOD,#11111101b ; intialize timer mode: timer 0 in 16 bit mode
orl TMOD,#00000001b
setb TR0;
setb EA;                    ; enable timer interrupt
setb ET0;
ret;

timer_int_handler:          ; timer interrupt handler
mov R7, A           ; save current acc
clr IE0         ; prevent INT0 from getting triggered with timer ir

mov TL0,TIMER_RELOAD_VAL;
mov TH0,TIMER_RELOAD_VAL+1;
mov A,TIMER_DEC_COUNTER
jz timer_lo_zero
dec TIMER_DEC_COUNTER
jmp timer_hi_zero
timer_lo_zero:
mov A,TIMER_DEC_COUNTER+1
jz timer_hi_zero
dec TIMER_DEC_COUNTER+1
mov TIMER_DEC_COUNTER,#0FFh
timer_hi_zero:
inc TIMER_INC_COUNTER

mov A, R7           ; reset acc to state before interrupt
reti
```

Programmcode 4.3: Quellcode-Ausschnitt für die Timerverwaltung

4.4 LCD Display

Das LCD Display ist einer der komplizierteren Komponenten aus dem Gesamtprogramm. Die beiden Macros zum Schreiben eines Strings auf das LCD Display sind in Programmcode 4.4 dargestellt. Die Komponente selbst ist allerdings deutlich größer, um auch Routinen für das Schreiben der aktuellen Statistik einfach anzubieten.

```
lcd_sendChar macro char ;call this macro like   LCD_sendChar #'A'
mov     LCD_data, char  ;Move the char to LCD port
setb    LCD_rs          ;Selected data register
clr LCD_rw              ;We are writing
setb    LCD_en          ;Enable H-> L
clr LCD_en
inc CURSOR_POS          ;save new cursor position
call    LCD_busy        ;Wait for LCD to process the data
endm

lcd_sendString macro string
local START             ;define local macro label
local EXIT              ;define local macro label
mov   dpl, #low(string)
mov   dph, #high(string)
start:                  ;local labels have to be in lowercase, idk why^^
clr   a                 ;clear Accumulator for any previous data
movc  a,@a+dptr         ;load the first character in accumulator
jz    EXIT              ;go to exit if zero
LCD_sendChar A          ;send actual char
inc   dptr              ;increment data pointer
sjmp  START             ;jump back to send the next character
exit:                   ;local labels have to be in lowercase, idk
endm
```

Programmcode 4.4: Quellcode-Ausschnitt für das LCD Display

4.5 Hauptroutine

Die Hauptroutine initialisiert zuerst sämtliche Komponenten. `gamestart` schreibt dann den `BEGIN` Text auf das Display und wartet auf das Drücken des Buzzers. Sobald dieser gedrückt wird, wird die Bombe einem zufälligen Spieler gegeben und danach der Timer auf einen zufälligen Wert gesetzt.In Programmcode 4.6 ist dieser Ablauf zu sehen.

Programmcode 4.5 zeigt die Hauptschleife in der das Programm läuft. Dabei wird zuerst überprüft, ob der Counter abgelaufen, das heißt die Bombe bereits explodiert ist. Falls nicht, wird der Buzzer überprüft und die `toggleActivePlayer` Routine aufgerufen, sofern dieser gedrückt ist. Im Anschluss wird mit `sendActivePlayerNumber` das LCD Display aktualisiert.

```
throwBomb:
; exit game on timeout
mov A,TIMER_DEC_COUNTER
orl A,TIMER_DEC_COUNTER+1
jz game_timeout

; check for button press
mov A,BUZZER
cjne A, #1, throwBomb ; not pressed

; pressed
clr BUZZER

call toggleActivePlayer
call sendActivePlayerNumber
ljmp throwBomb
```

Programmcode 4.5: Quellcode-Ausschnitt: Hauptschleife

```
gamestart:
LCD_setCursor #01h, #00h
LCD_sendString BEGIN

call waitForBuzzer

; set active player based on random number
call random_seed_from_timer
call random_next
mov A,RANDOM_NUM

mov ACTIVE_PLAYER, #01h
jb A.0,gamestart_afterplayer
mov ACTIVE_PLAYER, #02

gamestart_afterplayer:
;send Active Player to LCD
lcd_setCursor #01h, #00h
lcd_sendString ACTIVE
call sendActivePlayerNumber
call lcd_clearToEndOfLine

; setup timer
call random_next

; In the simulator: 15 ticks max
mov A,RANDOM_NUM
anl A,#00001111b
mov B,#0

; In the simulator: 10 extra ticks
; no 16bit addition necessary here
add A,#10
mov R0,A
mov A,#0

; now write the result to the timer
mov TIMER_DEC_COUNTER,R0
mov TIMER_DEC_COUNTER+1,A

clr BUZZER
```

Programmcode 4.6: Quellcode-Ausschnitt für die Hauptroutine

5 Zusammenfassung

Zusammenfassend lässt sich sagen, dass die Umsetzung des Spiels zwar komplizierter war als gedacht, alles in allem aber gut machbar war.

Das Endergebnis ist zwar spielbar, durch die Performanz des Simulators allerdings sehr langsam. Die hier dargestellten Quellcode-Ausschnitte dienen nur zur Illustration der wichtigsten Bestandteile. Der vollständige Quellcode ist in unserem Github-Repository unter https://github.com/flitzpiepe96/Systemnahe-Programmierung einsehbar.

Zum Schluss folgen noch ein paar Bilder aus dem simulierten Programmablauf.

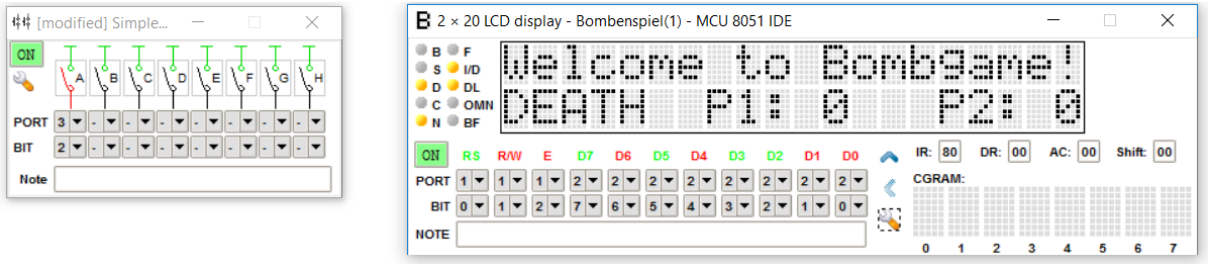


Abbildung 5.1: Willkommensnachricht

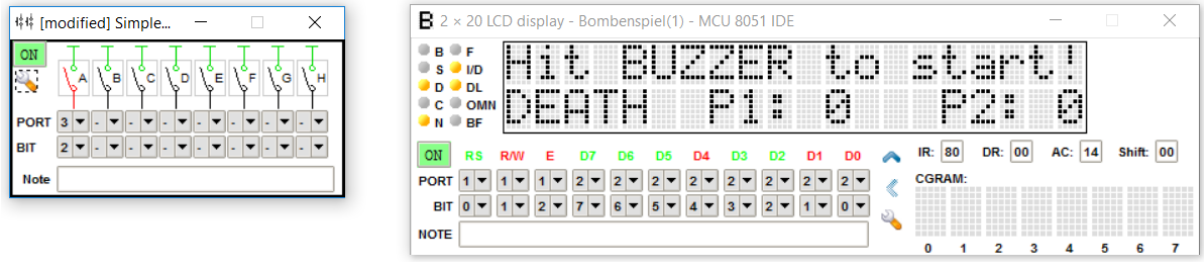


Abbildung 5.2: Startbildschirm

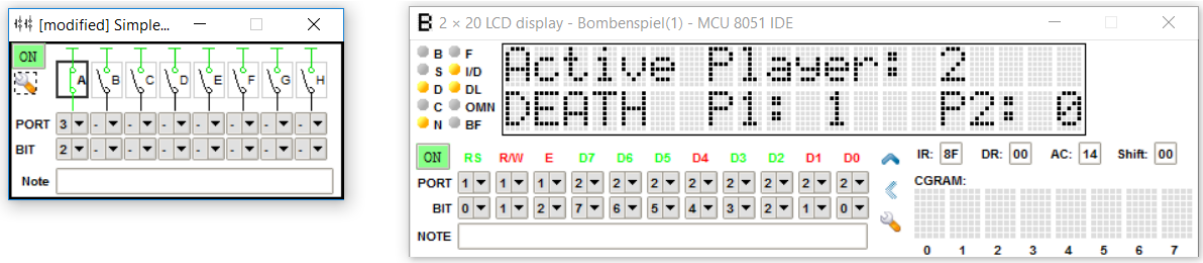


Abbildung 5.3: Spielablauf

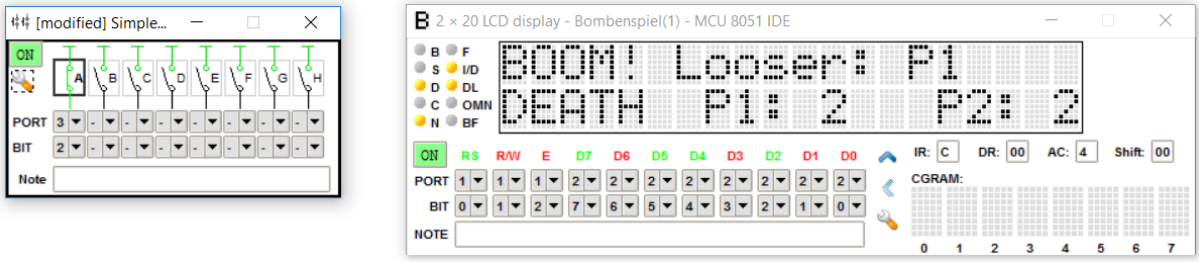


Abbildung 5.4: Ende

Literaturverzeichnis

[1] Wikipedia. Assemblersprache. `https://de.wikipedia.org/wiki/Assemblersprache`. Datum: 20.06.2017.

[2] Ernst-Wolfgang Dieterich. *Assembler: Grundlagen der PC-Programmierung.* Oldenbourg Wissenschaftsverlag GmbH, 2005. S.2.

[3] Tutorialspoint. Embedded Systems - 8051 Microcontroller. `https://www.tutorialspoint.com/embedded_systems/es_microcontroller.htm`. Datum: 20.06.2017.

[4] Mikrocontroller.net. 8051. `https://www.mikrocontroller.net/articles/8051`. Datum: 20.06.2017.

[5] Xtronic. Download MCU 8051 IDE integrated development enviroment for 8051. `https://xtronic.org/download/microcontroller/mcu-8051-ide-microcontrollers-development/`. Datum: 20.06.2017.

Abbildungsverzeichnis