

EdSim51 ist eine Simulation des Intel 8051 und eines Evaluation Boards mit diverser Peripherie.

1. Einstieg in die 8051 Programmierung

a) Machen Sie sich mit der EdSim51 Benutzeroberfläche und dem Schaltplan vertraut.

b) Schalten Sie die LED0 an:

Die LED0 liegt in Durchlassrichtung zwischen High (+3.3V) und Port Pin P1.0

Entsprechend muss der Pin P1.0 auf Low (Masse) gesetzt werden. Geben Sie dazu in EdSim51 folgenden Assembler-Befehl ein und führen diesen mit „Run“ aus: **CLR P1.0**

Schalten Sie nun auch die LED7 an.

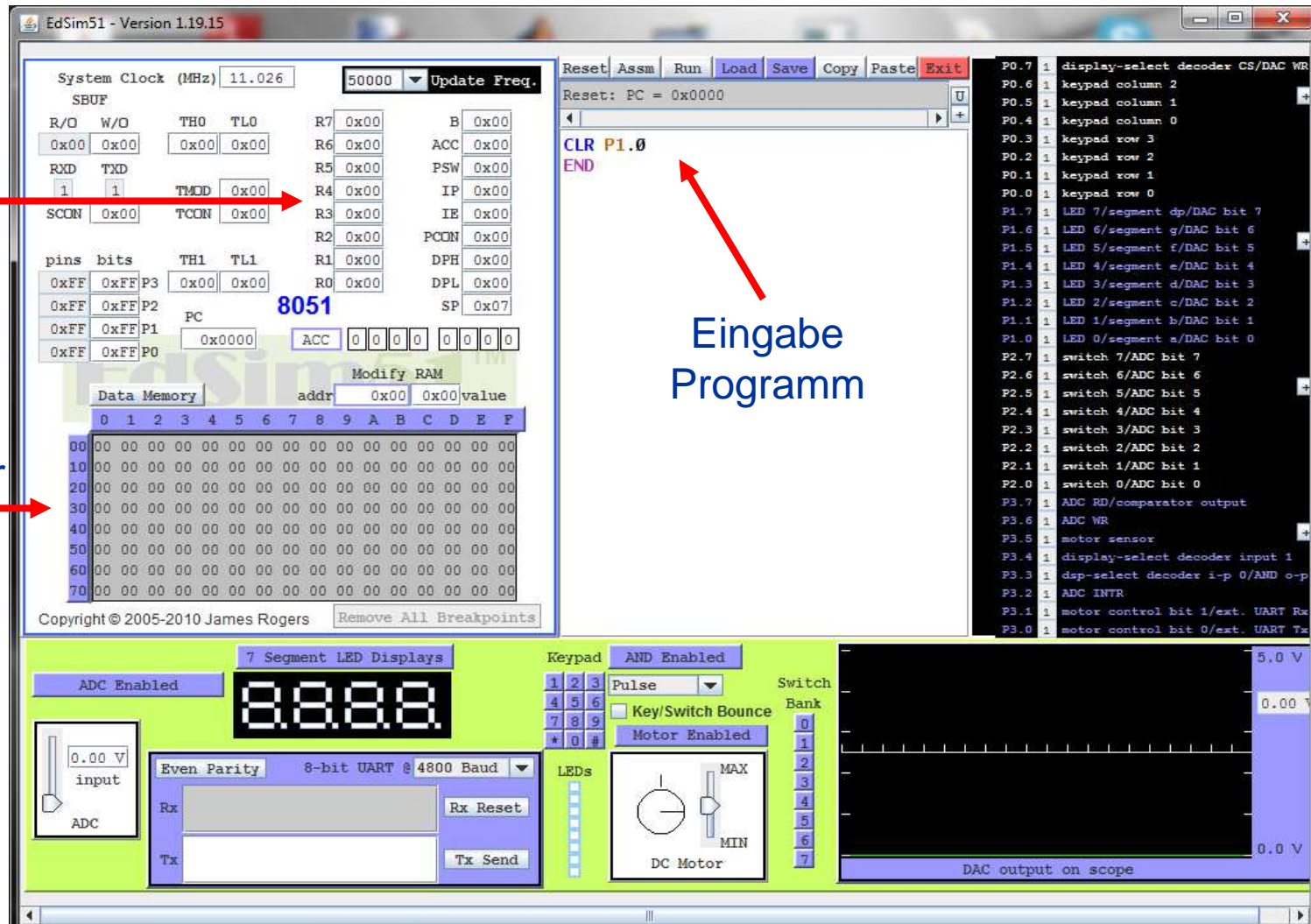
Register

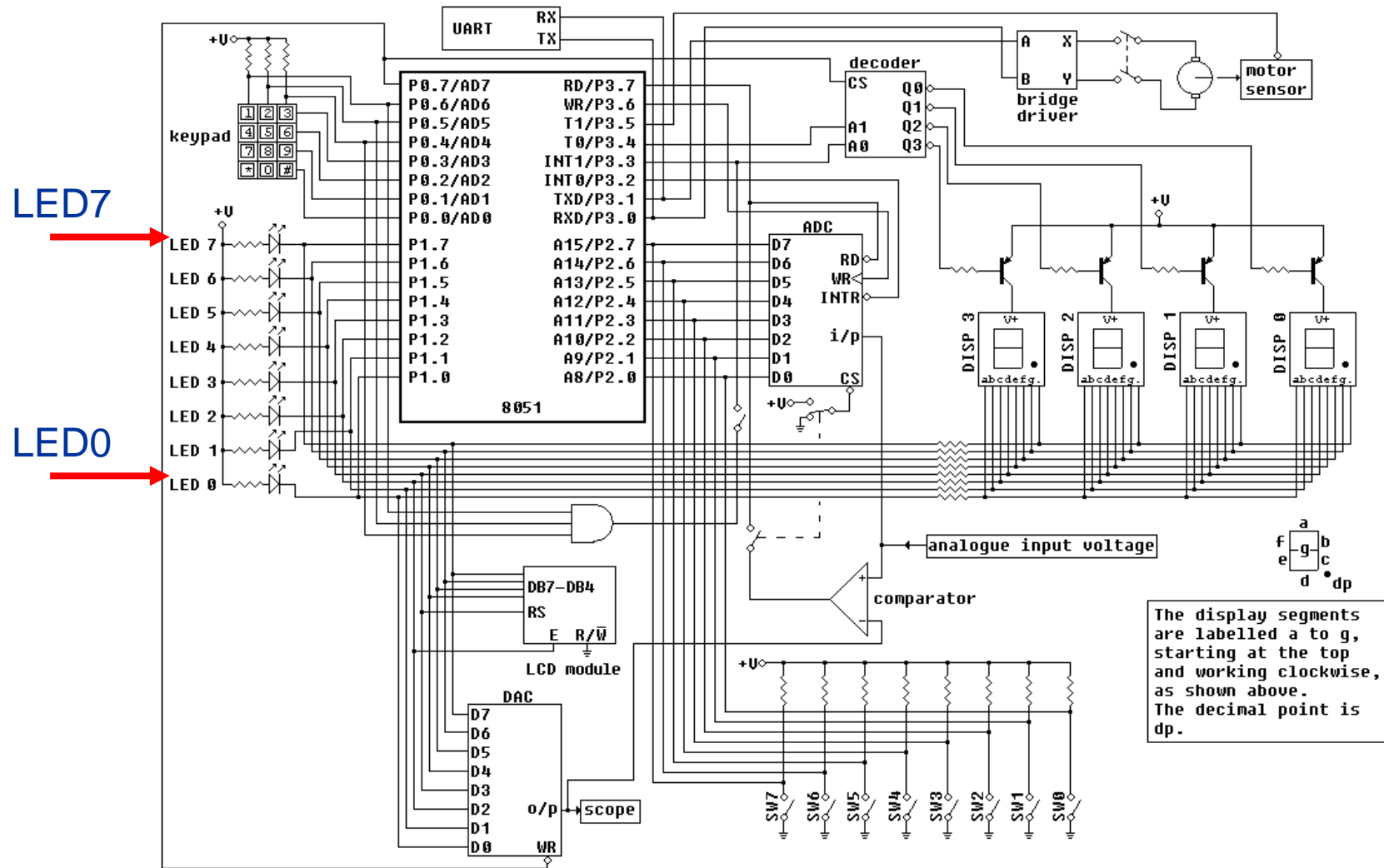
Speicher

Eingabe
Programm

Pins

Peripherie





c) Erstellen Sie mit einem Editor das Assembler Programm aus der Vorlesung.

Problem:

Der Schalter SW0 soll die LED0 ein- und ausschalten.

Lösung:

Die LED0 liegt in Durchlassrichtung zwischen High (+3.3V) und Port Pin P1.0
Der Schalter SW0 liegt zwischen Port Pin P2.0 und Low (Masse).

d) Führen Sie das Programm in EdSim51 aus.

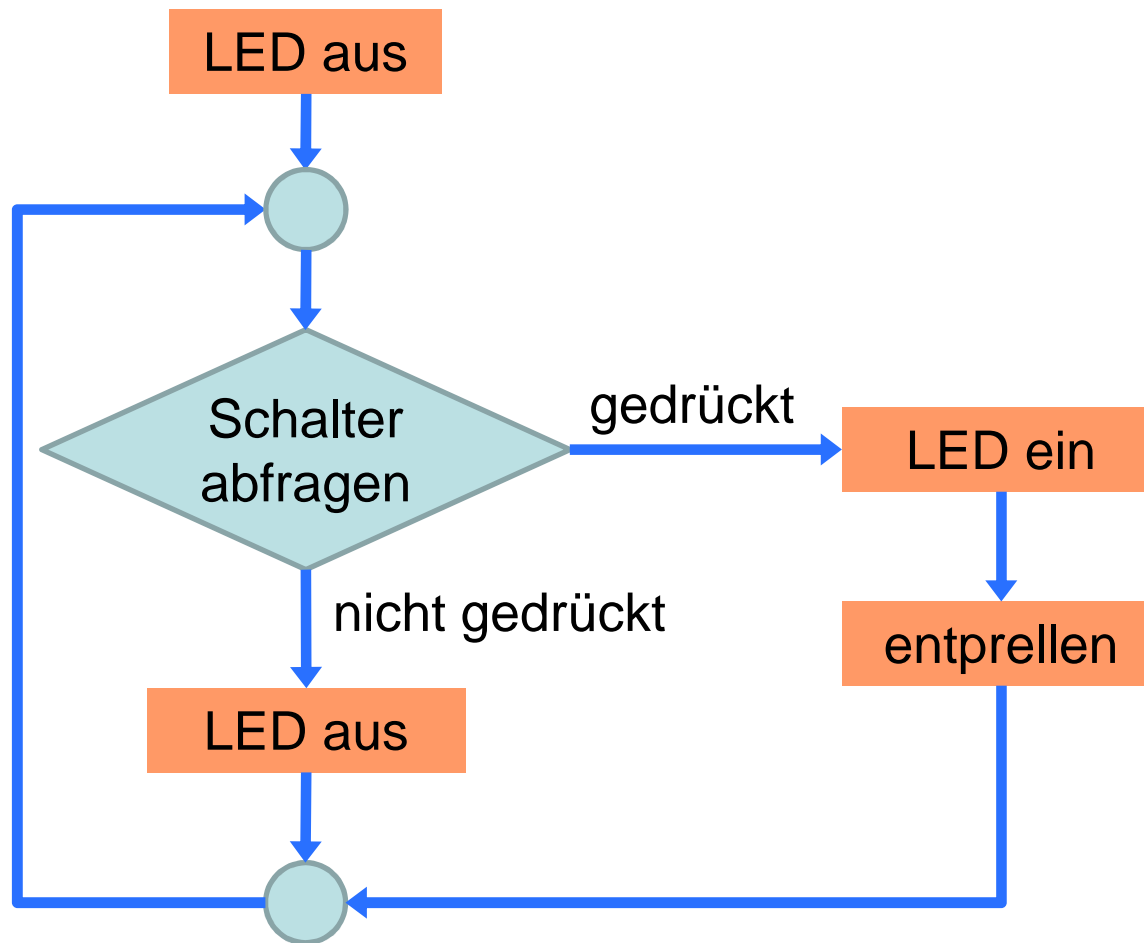
Hinweise:

Nutzen Sie die Einstellungen: CLK = 11.026 MHz Update = 50000 Hz

Sie können das Programm auch schrittweise ausführen: Nutzen Sie dazu „Assm“ und „Step“. Zudem können Sie durch Doppelklick auf eine Zeile einen Breakpoint setzen und das Programm bis dorthin mit „Run“ ausführen.

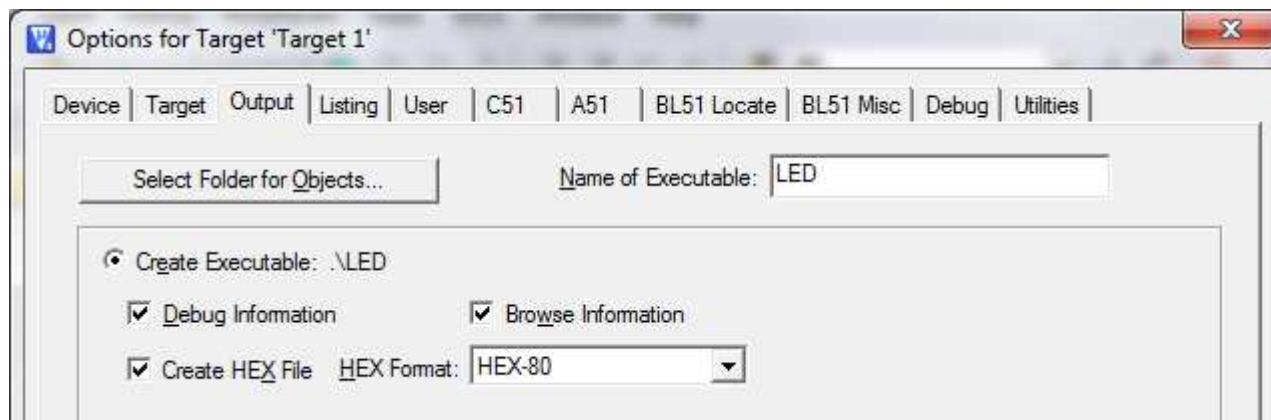
```
main:
    SETB    P1.0          ; LED off (high)
    SETB    P2.0          ; switch off (high)
loop:
    JNB     P2.0, led      ; check switch (low) and jump
    SETB    P1.0          ; if not : LED off (high)
    JMP     loop
led:
    CLR     P1.0          ; LED on (low)
    CALL    delay         ; switch de-bounce
    JMP     loop
delay:
    MOV     R0, #100
loop1:
    DJNZ    R0, loop1      ; loop until 0
    RET
```

e) Tragen Sie neben dem Ablaufdiagramm die entsprechenden
Assembler Befehle ein:



-
- f) Erstellen Sie nun das Assembler Programm aus der Vorlesung mit dem Keil Entwicklungssystem und führen Sie das Programm als HEX Datei in EdSim51 aus. Musterlösung „1b_LED_ASM_HEX\LED.uvproj“

Hinweis: EdSim51 kann als Simulator ein Assembler Programm als Text ausführen. Reale Mikrocontroller müssen ein Programm als Code in den Programmspeicher (ROM/Flash) laden. Hierzu muss das Assembler Programm assembliert werden. Das Ergebnis ist eine HEX Datei (oder eine Binärdatei).



-
- g) Erstellen Sie nun ein funktionsgleiches C Programm mit dem Keil Entwicklungssystem und führen Sie das Programm als HEX Datei in EdSim51 aus. Musterlösung „1c_LED_C\LED.uvproj“**
 - h) Erstellen Sie das C Programm zur Leuchtkette aus der Vorlesung mit dem Keil Entwicklungssystem und führen Sie das Programm als HEX Datei in EdSim51 aus. Musterlösung „1d_LED_C\LED.uvproj“**

Problem:

Leuchtkette über alle LEDs

Lösung:

An Port P1 wird 1 Bit invers von P1.0 bis P1.7 durchgeschoben.

- i) Welche Werte nimmt der Port P1 im Programmverlauf an?**

Programm zu g) : Schalter

```
#include <REG51.h>
void delay(void) {
    volatile unsigned char n; // prevent compiler omission
    for (n=0; n<250; n++) {}
}

sbit LED = P1^0;
sbit SWITCH = P2^0;
void main(void) {
    LED = 1;           // LED off (high)
    SWITCH = 1;        // switch off (high)
    while (1) {
        if (SWITCH)    // check switch (low)
            LED = 1;    // if not : LED off (high)
        else {
            LED = 0;    // LED on (low)
            delay();    // switch de-bounce
        }
    }
}
```

Programm zu h) : Leuchtkette

```
#include <REG51.h>
void delay(void) {
    volatile unsigned char n;    // prevent compiler omission
    for (n=0; n<250; n++) {}
}

void main(void) {
    unsigned char idx;

    while (1)
    {
        for (idx=0; idx<8; idx++)    // bit index
        {
            P1 = 0xFF ^ (1 << (idx)); // running clear one LED

            delay();
        }
    }
}
```

2. Subroutinen und Arithmetik

a) Prüfen Sie mit EdSim51 die Funktion dieses Programms:

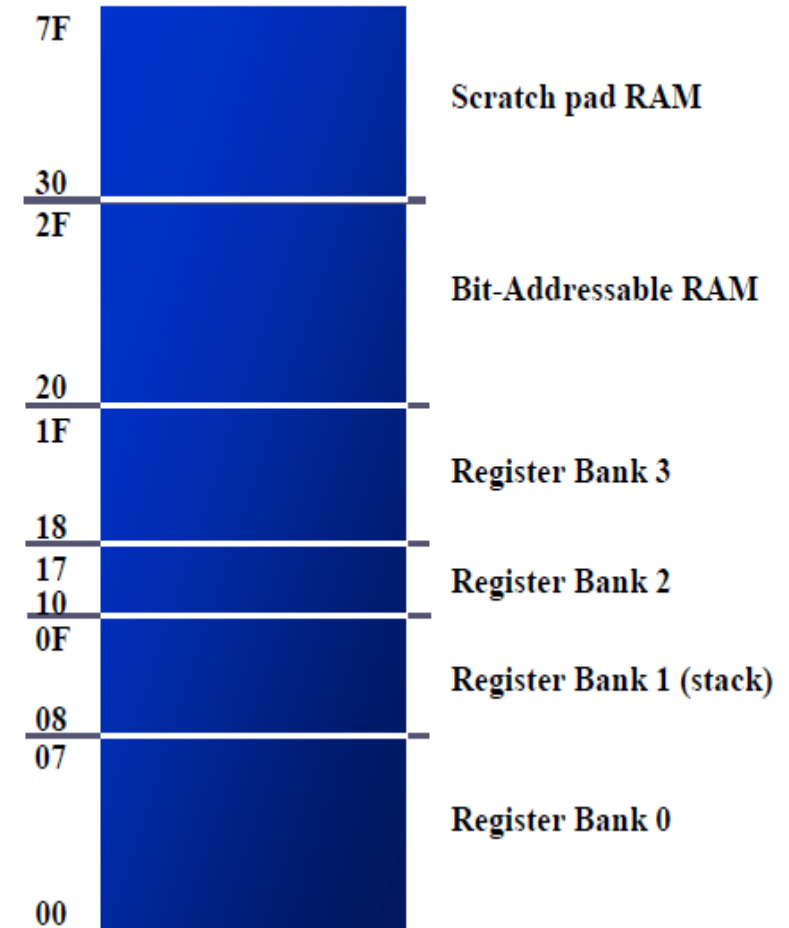
```
main:
    MOV     R0, #30H
    MOV     @R0, #6H
    MOV     R0, #31H
    MOV     @R0, #4H
    MOV     R0, #32H
    MOV     @R0, #1H
function:
    MOV     A, 30H
    ADD     A, 31H
    ADD     A, 32H
    MOV     B, #3H
    DIV     AB
    MOV     40H, A
    MOV     41H, B
```

Hinweis:

„Data Memory“ Adressen 0x30 und 0x40

Prof. Dr. Berger © 2018

RAM Allocation in 8051



b) Nun wird für dieselbe Funktion ein Unterprogramm und der Stack genutzt: SP = 0x6F (0x70 – 1)

```
main:
    MOV     R0, #30H
    MOV     @R0, #6H
    MOV     R0, #31H
    MOV     @R0, #4H
    MOV     R0, #32H
    MOV     @R0, #1H
parameter:
    MOV     R0, #30H
    MOV     R1, #3H
    MOV     SP, #6FH
    CALL    sub
    MOV     40H, A
    MOV     41H, B
    JMP     main
```

```
sub:
    PUSH    0
    PUSH    1
function:
    MOV     B, R1
    CLR     A
loop:
    ADD     A, @R0
    INC     R0
    DJNZ    R1, loop
    DIV     AB
    POP     1
    POP     0
    RET
```

c) Erklären Sie wie die Werte auf dem Stack zustande kommen:



70	15	00	30	03
----	----	----	----	----

3. Speicherzugriff

a) Bestimmen Sie die Funktion dieses Programms in EdSim51:

```
ORG 000
main:
    MOV        DPTR, #180H        ; table address
    MOV        A, #0FFH
    MOV        P2, A              ; P2 input
loop:
    MOV        A, P2              ; read n
    CPL        A
    MOVC        A, @A+DPTR        ; n*n from table
    CPL        A
    MOV        P1, A              ; output n*n
    JMP        loop

; ROM from byte 384
ORG 180H
table:
    DB          0,1,4,9,16,25,36,49,64,81

    END
```

-
- b) Erstellen Sie eine Subroutine, die eine Tabelle nutzt, um den ganzzahligen Wert $100 \cdot \sin(\phi)$ für $\phi = 0^\circ, 10^\circ, \dots, 90^\circ$ zurückzugeben.
- c) Testen Sie die Subroutine in EdSim51 für $\phi = 30^\circ$

Hinweis: Musterlösung „3_ROM_ASM\3b_ROM.asm“

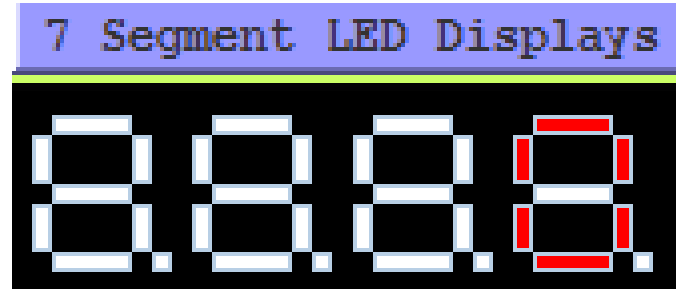
4. Peripherie – Segment-Anzeige

a) Bestimmen Sie die Funktion dieses Programms in EdSim51:

```
ORG 000
main:
    CLR        P0.7           ; MUX CS -> SEG off
    MOV        P1, #0FFH      ; SEG clear
    CLR        P3.3           ; MUX A0 = 0 -> SEG #1 select
    CLR        P3.4           ; MUX A1 = 0
    SETB       P0.7           ; MUX CS -> SEG on
    MOV        DPTR, #180H     ; SEG table

loop1:
    MOV        R0, #0          ; SEG = 0

loop2:
    MOV        A, R0
    MOVC       A, @A+DPTR      ; SEG number from table
    MOV        P1, A           ; SEG show number
    CALL       delay
    INC        R0              ; SEG + 1
    CJNE       R0, #3, loop2   ; MAX ?
    JMP        loop1
```



```

delay:
    MOV            R3, #250            ; long delay : 250*250
d_loop1:
    MOV            R4, #250
d_loop2:
    DJNZ           R4, d_loop2          ; loop while <> 0
    DJNZ           R3, d_loop1          ; loop while <> 0
    RET

; ROM from byte 384
ORG 180H
table:
    DB             11000000B           ; 0
    DB             11111001B           ; 1
    DB             10100100B           ; 2

    END

```

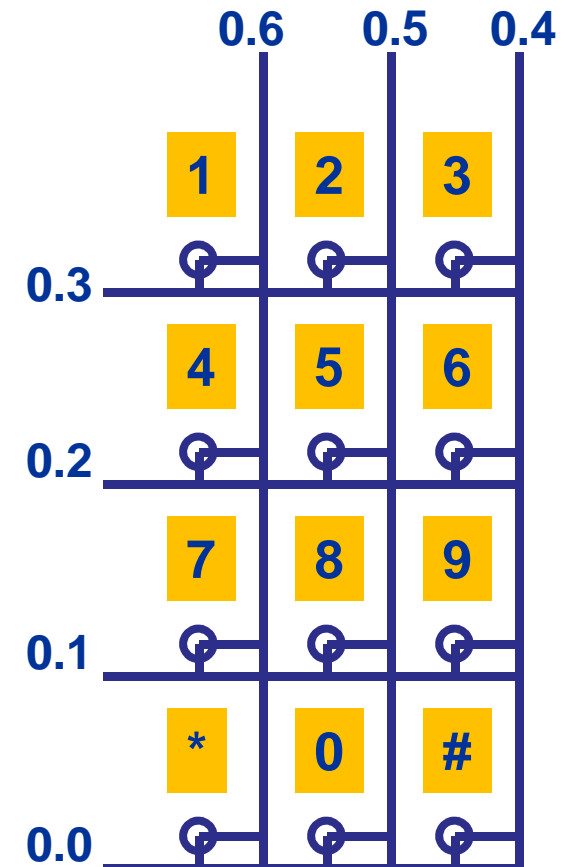
b) Erstellen Sie nun ein funktionsgleiches C Programm. Erweitern Sie dieses derart dass die Segment-Anzeige von 0 bis 9 zählt.

Hinweis: Musterlösung „4b_SEG_C“

5. Peripherie – Keypad

a) Führen Sie das Programm in EdSim51 aus.
Kommentieren Sie die Programmschritte.

```
main:
    MOV        R0, #-1
    ORL        P0, #01111111B
row3:
    CLR        P0.3
    JNB        P0.6, key1
    JNB        P0.5, key2
    JNB        P0.4, key3
    JMP        main
key1:
    MOV        R0, #1
    RET
key2:
    MOV        R0, #2
    RET
key3:
    MOV        R0, #3
    RET
```



-
- b) Erweitern Sie das Programm, um das gesamte Keypad abzudecken.
- c) Nutzen Sie nun das C Programm für die Segment-Anzeige von 0 bis 9 zur Anzeige des Keypads. Rufen Sie hierzu das Assembler Programm für das Keypad aus dem C Programm auf:

```
extern unsigned char keypad(void);
```

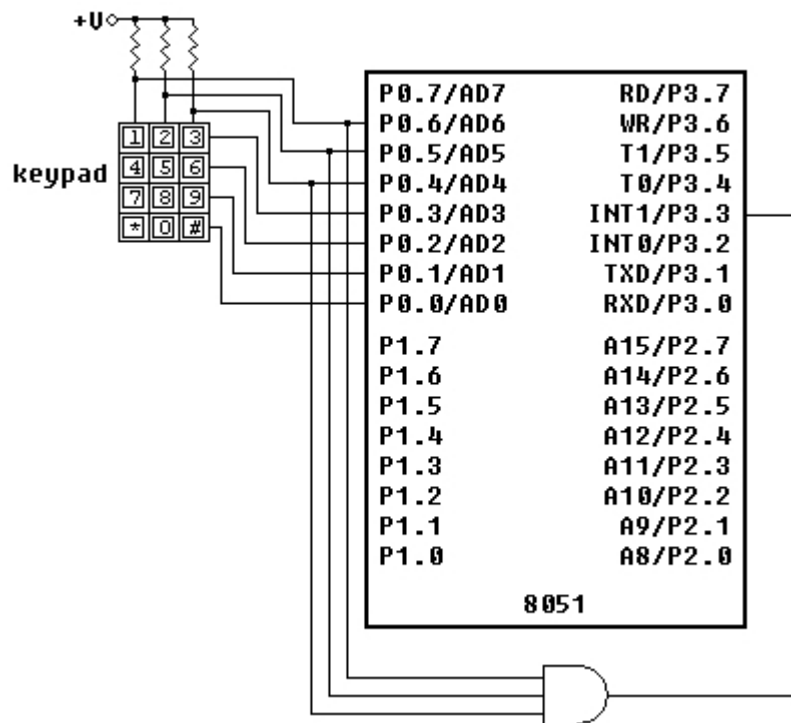
Hinweis:

Beim 8051 erfolgt die Übergabe von Assembler nach C in umgekehrter Register-Reihenfolge, somit ist das hier zurückzugebende Byte im höchsten Register R7 zu platzieren.

Musterlösung „5b_KEY_SEG“

6. Peripherie – Keypad – Interrupt

Das Keypad ist zusätzlich zum Port P0 mit dem externen Interrupt Pin INT1 verbunden. Eine negative Flanke an INT1 bewirkt eine Unterbrechung des Hauptprogramms und den Aufruf einer programmierbaren Interrupt-Funktion an der Adresse 0x13 bzw. mit der Interrupt-Nummer 2 (→ Vorlesungs-Skript)



a) Führen Sie das Programm in EdSim51 aus. Erklären Sie die Programmschritte anhand des Schaltplans.

```
ORG 000
    JMP                main

ORG 013H
    JNB                P0.6, key1
    JNB                P0.5, key2
    JNB                P0.4, key3
key1:
    CPL                P1.0
    RETI
...
```

```

...
key2:
    CPL                P1.1
    RETI
key3:
    CPL                P1.2
    RETI

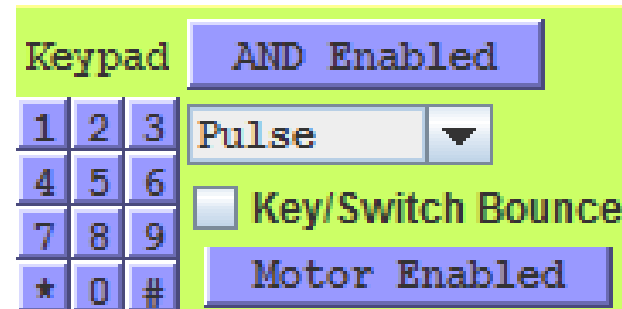
ORG 180H
main:
    MOV                TCON, #00000100B
    MOV                IE, #10000100B

    ORL                P0, #01111111B
    CLR                P0.3

loop:
    JMP                loop

```

Hinweis: für das korrekte Funktionieren muss das Keypad auf Pulse Mode gesetzt und das verbindende AND Gatter enabled sein



b) Erstellen Sie ein äquivalentes Programm in C

Hinweis: Assembler
C

```

ORG 013H ...
void extint1(void) interrupt 2 using 3 { ...

```

Nummer **Register-Bank**



Hinweis: Musterlösung „6b_KEY_INT_C“

7. Timer

a) Bestimmen Sie die Funktion dieses C Programms in EdSim51 und kommentieren Sie die Programmschritte:

```
#include <REG51.h>
```

```
void T1Start() {
```

```
    TMOD=0x20;
```

```
    TH1=0x5;
```

```
    TF1=0;
```

```
    TR1=1; }
```

```
void T1Wait() {
```

```
    while (TF1==0);
```

```
    TF1=0; }
```

← Hinweis: Timer – Polling

```
void main(void) {
```

```
    T1Start();
```

```
    while (1) {
```

```
        P1=0x00;  T1Wait();
```

```
        P1=0xff;  T1Wait();
```

```
    }
```

```
}
```

Prof. Dr. Berger © 2018

b) Erstellen Sie nun ein funktionsgleiches Assembler Programm.

Hinweis: Musterlösung „7a_TIMER_POLL\LED.asm“

c) Das C Programm nutzt Timer 1. Ändern Sie das Programm so ab, dass Timer 0 genutzt wird.

d) Bestimmen Sie die Funktion dieses C Programms in EdSim51 und kommentieren Sie die Programmschritte:

```
#include <REG51.h>
```

```
void T0Start() {  
    TMOD=0x02;  
    TH0=0x05;  
    IE=0x82;  
    TR0=1;  
}
```

Hinweis: Timer – IRQ



```
void T0Event(void) interrupt 1 using 3 {  
    P1 = ~P1;  
}
```

```
void main(void) {  
    T0Start();  
    P1=0xff;  
    while (1)  
    {  
    }  
}
```

e) Warum muss das Timer-Flag nicht explizit gelöscht werden?

f) Erstellen Sie nun ein funktionsgleiches Assembler Programm.

Hinweis: Musterlösung „7b_TIMER_IRQ\LED.asm“

g) Das C Programm nutzt Timer 0. Ändern Sie das Programm so ab, dass Timer 1 genutzt wird.

-
- h) Ein Werkstück soll für 1s in ein Bad getaucht werden. Key 1 soll den Vorgang auslösen, und LED 1 die Dauer anzeigen. Erstellen Sie ein dafür geeignetes Programm in C und testen Sie mit EdSim51

Hinweis: Zum Abzählen von 1s:

1 Timer/1 Interrupt/250us, 1 Globaler Zähler/unsigned short/1s

Musterlösung „7c_TIMER_SWI“

- i) Nun stehen 2 Bäder zur Verfügung. Ist ein Bad noch belegt, soll automatisch das zweite Bad genutzt werden. Key 1 soll den Vorgang auslösen, und LED 1 und 2 sollen die Dauer anzeigen.

Hinweis: Zum Abzählen von 2x 1s:

1 Timer/1 Interrupt/250us, 2 Globale Zähler/unsigned short/1s

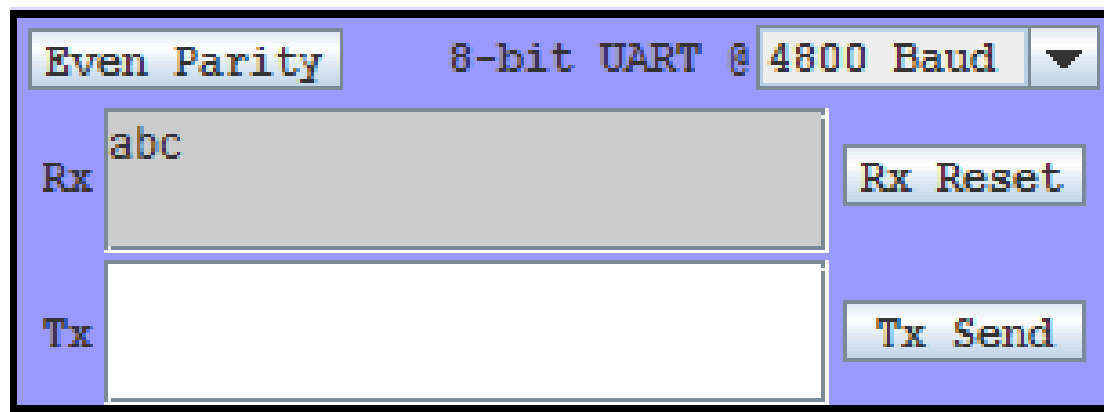
Musterlösung „7d_TIMER_SWI_MULT“ (kein Prüfungsstoff)

8. Serielle Kommunikation

Senden

Hinweis: für das korrekte Funktionieren der seriellen Simulation sollte Update = 100 Hz gesetzt werden

- a) Führen Sie das Assembler Programm in EdSim51 aus und prüfen Sie die Funktion mit folgenden Einstellungen:



The screenshot shows the UART configuration window in EdSim51. At the top, it displays 'Even Parity', '8-bit UART @', and '4800 Baud' with a dropdown arrow. Below this, there are two main sections: 'Rx' and 'Tx'. The 'Rx' section has a text box containing 'abc' and a button labeled 'Rx Reset'. The 'Tx' section has an empty text box and a button labeled 'Tx Send'.

- b) Erstellen Sie ein äquivalentes Programm in C

Hinweis: Musterlösung „8b_SERIAL_SEND_C“

```

main:
    MOV     SCON, #01000000B ; USART mode %01 = 8-bit data
    ORL     PCON, #10000000B ; set double baud rate
    MOV     TMOD, #20H       ; timer 1 mode 2
    MOV     TH1, #-12        ; timer 1 = -12: baud = 2400: double = 4800
    MOV     TL1, #-12        ; timer 1 sync
    SETB    TR1              ; timer 1 start
    MOV     30H, #'a'         ; start string
    MOV     31H, #'b'
    MOV     32H, #'c'
    MOV     33H, #0           ; terminate string
    MOV     R0, #30H

loop:
    MOV     A, @R0
    JZ      done              ; string terminated
    MOV     C, P              ; add parity to char through carry
    MOV     ACC.7, C          ; in accu
    MOV     SBUF, A           ; send data
    INC     R0

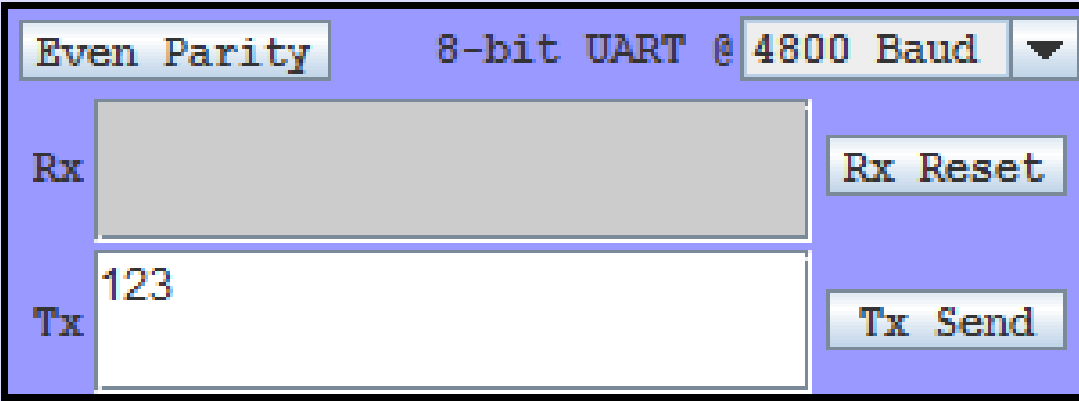
wait:
    JNB     TI, wait          ; wait for data sent
    CLR     TI                ; clear sent flag
    JMP     loop

done:
    END

```

Empfangen

- c) Führen Sie das Assembler Programm in EdSim51 aus und prüfen Sie die Funktion mit folgenden Einstellungen:



The screenshot shows the UART configuration window in EdSim51. At the top, it displays 'Even Parity', '8-bit UART @', and '4800 Baud' with a dropdown arrow. Below this, there are two main sections: 'Rx' (Receiver) and 'Tx' (Transmitter). The 'Rx' section has a large grey rectangular area for displaying received data. The 'Tx' section has a text input field containing the number '123'. To the right of the 'Rx' area is a button labeled 'Rx Reset'. To the right of the 'Tx' input field is a button labeled 'Tx Send'.

- d) Erstellen Sie ein äquivalentes Programm in C

Hinweis: Musterlösung „8d_SERIAL_RECV_C“

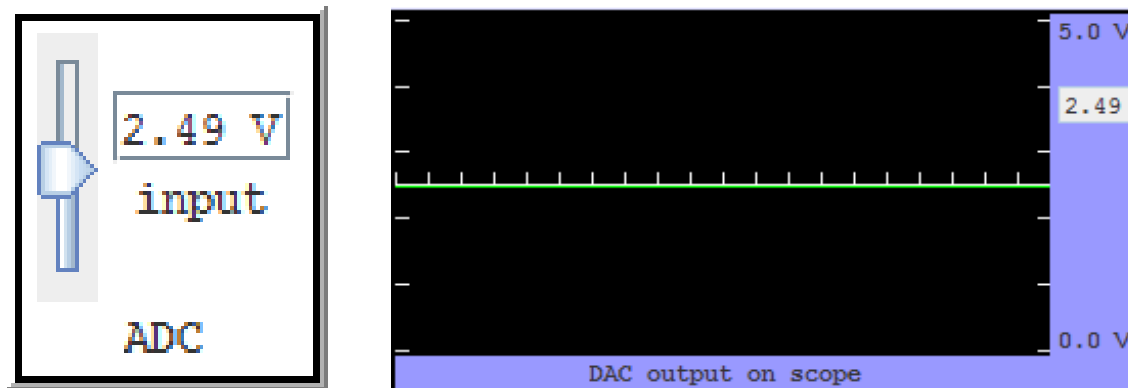
```

main:
    MOV     SCON, #01010000B ; USART mode %01 = 8-bit data + enable receiver
    ORL     PCON, #10000000B ; set double baud rate
    MOV     TMOD, #20H       ; timer 1 mode 2
    MOV     TH1, #-12        ; timer 1 = -12: baud = 2400: double = 4800
    MOV     TL1, #-12        ; timer 1 sync
    SETB    TR1              ; timer 1 start
    MOV     R0, #30H         ; empty string start
loop:
    JNB     RI, loop         ; wait for data received
    CLR     RI               ; clear received flag
    MOV     A, SBUF          ; fetch received byte
    CJNE    A, #8DH, skip    ; receiver termination is 8DH
    JMP     done
skip:
    CLR     ACC.7            ; clear parity bit
    MOV     @R0, A           ; put into empty string
    INC     R0
    JMP     loop
done:
    END

```

9. Peripherie – ADC / DAC

In EdSim51 steht ein ADC (Analog Digital Converter) zur Verfügung, an den ein Schieberegler angeschlossen ist, sowie ein Oszilloskop, das als DAC (Digital Analog Converter) anzusehen ist.



- Testen Sie das Assembler Programm mit EdSim51 und erklären Sie die Funktion.
- Erstellen Sie ein äquivalentes Programm in C
Hinweis: Musterlösung „9b_ADC_C“

```

ORG 000H
    JMP    main
ORG 003H
    JMP    ext0                ; ADC IRQ on INT0
ORG 00BH
    JMP    timer0             ; ADC sampling rate by T0
ORG 030H
main:
    CLR    P0.7                ; enable DAC write
    MOV    TCON, #00000001B    ; external interrupt 0 edge mode
    MOV    IE, #10000011B
    MOV    TMOD, #2
    MOV    TH0, #-20
    SETB   TR0
loop:
    JMP    loop
timer0:
    CLR    P3.6                ; send sampling pulse to ADC
                                ; enable ADC write
    NOP
    SETB   P3.6                ; disable ADC write
    RETI
ext0:
    CLR    P3.7                ; ADC has aquired
                                ; enable ADC read
    MOV    P1, P2              ; read data from ADC and write into DAC
    SETB   P3.7                ; disable ADC read
    RETI

```

Anhang • LCD-Display

Peripherie – LCD-Display (**kein Prüfungsstoff**)

Ein LCD-Display ist ein externes Peripheriegerät, dass nicht über 8051 Steuerregister betrieben wird, sondern über ein Steuerprotokoll, üblicherweise 8-bit oder 4-bit. Das Steuerprotokoll wird über einen Datenbus verschickt, als Datenbus kann ein Port dienen. Für das Erstellen eines LCD Programms muss die Bedienungsanleitung des Moduls herangezogen werden.

Hinweis: <http://www.mikrocontroller.net/articles/HD44780>

Führen Sie das C/Assembler Programm „9_LCD“ in EdSim51 aus (hierzu muss von Segment auf LCD umgeschaltet werden):

