Technische Universität Ilmenau
Fakulät IA
Fachgebiet Rechnerarchitektur

Praktikum Rechnerarchitektur 2
WS 2021/22

Versuchsprotokoll

Versuche Befehlsausführung und Mikrocontroller

26. Januar 2022

Versuch B: Befehlsausführung

Simulative Untersuchung der Ausführung von Maschinenbefehlen in unterschiedlichen Pipeline-Architekturen

Aufgabe 1

Untersuche die vorbereitete Befehlsfolge mit den drei vorgegebenen Grundstrukturen Standard-Pipeline, Superskalar-in-Order und Superskalar-out-of-Order. Beobachte den Programmablauf und machen dich mit der Bedienung vertraut! Schauen vor dem Simulationsstart auch die Parametereinstellungen für Sprungvorhersage und Result Forwarding an und interpretiere das Verhalten während der Simulation.

Code A1b

```
        addiu   $t1, $zero, 11
        addiu   $t2, $zero, 0
loop:   addu  $t2, $t2, $t1
        addiu   $t1, $t1, -1
        bnez    $t1, loop
```

Alle Strukturen mit Result-Forwarding und 2-Bit Vorhersage.

Beobachtung:

- Standard Pipeline
  - Takte: 43
  - Befehle: 39
  - Befehle pro Takt: 0,81
  - Sprünge: 11
- Superskalar In-Order Pipeline (4 EX Einheiten)
  - Takte: 29
  - Befehle: 44
  - Befehle pro Takt: 1,21
  - Sprünge: 11
- Superskalar Out-of-Order (4 EX Einheiten)
  - Takte: 20
  - Befehle: 58
  - Sprünge: 12

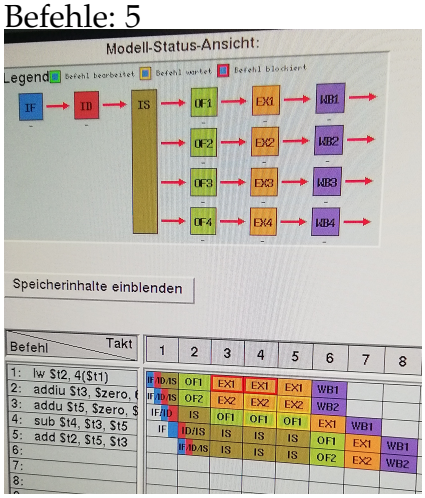
Aufgabe 2

Untersuche die Befehlsfolgen A4 und B2 mit mindestens je drei unterschiedlichen Simulationsläufen! Wähle die benutzten Pipelinestrukturen und Parametereinstellungen selbst aus. Vergleiche die Ergebnisse mit den Lösungen aus der Übung und suche Erklärungen für eventuelle Unterschiede!

Code A4

```
lw      $t2, 4($t1)
addiu   $t3, $zero, 65
addu    $t5, $zero, $t2
sub     $t4, $t3, $t5
add     $t2, $t5, $t3
```

Beobachtung:

- Standard Pipeline
  - Takte: 11
  - Befehle: 5
- Superskalar In-Order Pipeline (4 EX Einheiten)
  - Takte: 8
  - Befehle: 5
  - 

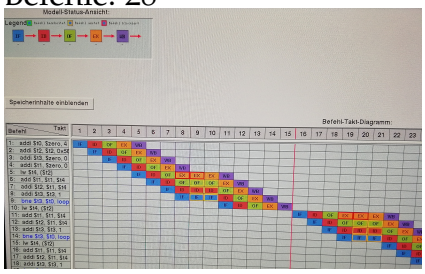
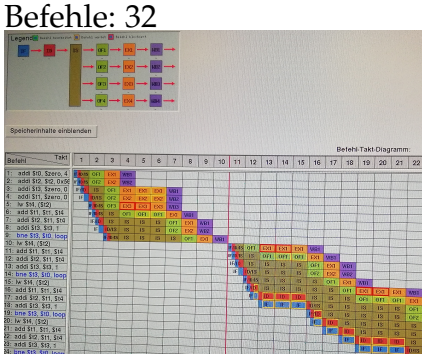
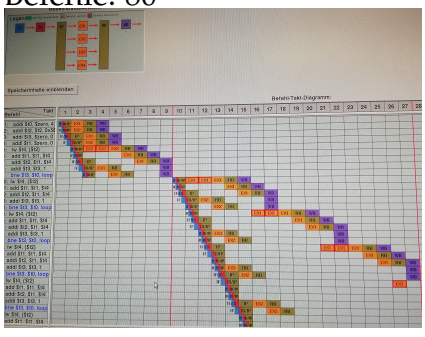
- Superskalar Out-of-Order (4 EX Einheiten)
  - Takte: 8
  - Befehle: 5

Code B2

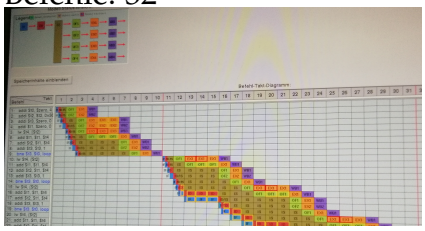
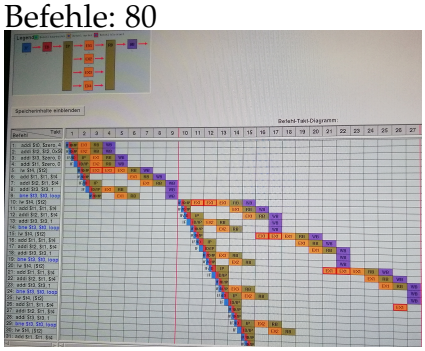
```
# addition der inhalte von 4 aufeinander folgenden speicherzellen, beginnend mit adresse 0x12345678 ...
# $t2 enthalte bereits den wert 0x12340000
    addi    $t0, $zero, 4           # max. zaehlerwert t0 = 4
    addi    $t2, $t2, 0x5678        # adressregister t2 = startadresse
```

```
    addi    $t3, $zero, 0          # zaehlerregister t3 = 0
    addi    $t1, $zero, 0          # ergebnisregister t1 = 0
loop:   lw      $t4, ($t2)         # tempregister t4 <- wert laden
    add     $t1, $t1, $t4          # summieren
    addi    $t2, $t2, 4            # adresse um 4 erhöhen
    addi    $t3, $t3, 1            # zaehler +1
    bne     $t3, $t0, loop         # loop für zaehler != 4
```

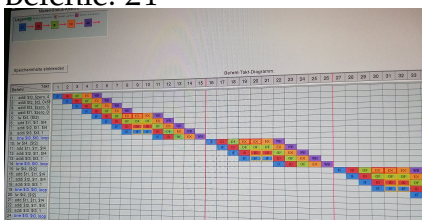
Beobachtung: 2 Bit Vorhersage

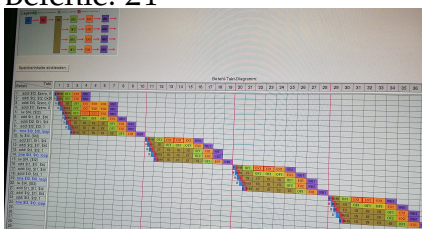
- Standard Pipeline
  - Takte: 40
  - Befehle: 28
  - 
- Superskalar In-Order Pipeline (4 EX Einheiten)
  - Takte: 31
  - Befehle: 32
  - 
- Superskalar Out-of-Order (4 EX Einheiten)
  - Takte: 27
  - Befehle: 80
  - 

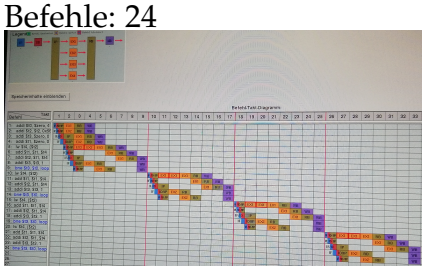
1 Bit Vorhersage

- Standard Pipeline
  - Takte: 40
  - Befehle: 28
- Superskalar In-Order Pipeline (4 EX Einheiten)
  - Takte: 31
  - Befehle: 32
  - 
- Superskalar Out-of-Order (4 EX Einheiten)
  - Takte: 27
  - Befehle: 80
  - 

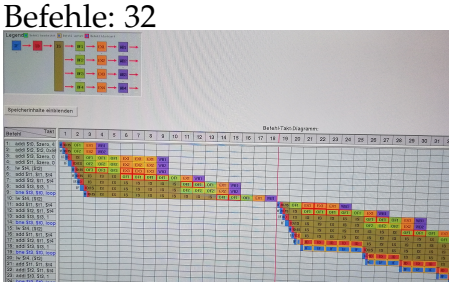
0 Bit Vorhersage

- Standard Pipeline
  - Takte: 48
  - Befehle: 24
  - 
- Superskalar In-Order Pipeline (4 EX Einheiten)
  - Takte: 37

  - Befehle: 24
  - 

- Superskalar Out-of-Order (4 EX Einheiten)
  - Takte: 33
  - Befehle: 24
  - 

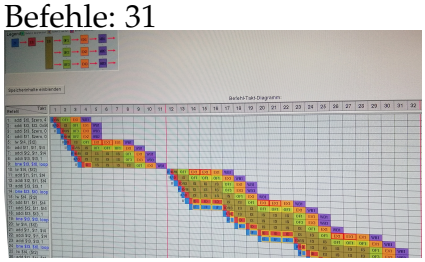
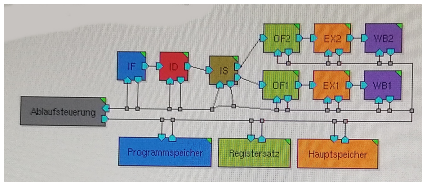
Superskalar In-Order Pipeline ohne Result Forwarding (4 EX Einheiten)

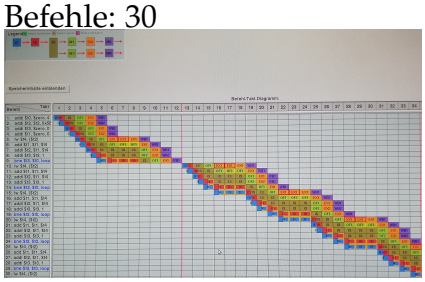
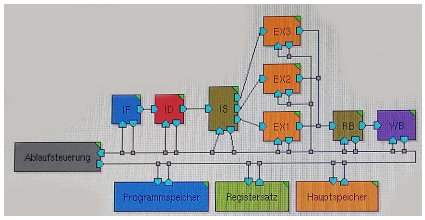
- Takte: 57
- Befehle: 32
- 

Aufgabe 3

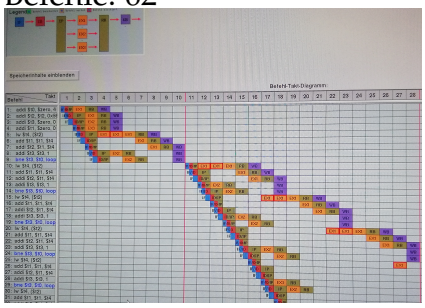
Änderne nun eine der vorgegebenen Pipelinestrukturen ab, z.B. die Anzahl der parallelen Pipelines verändern. Orientiere dich zuvor über den Inhalt des „Baukastens". Untersuche mit den oben verwendeten Befehlsfolgen die Auswirkungen auf die Simulationsergebnisse! Variiere dabei die Parameter und interpretiere die Ergebnisse!

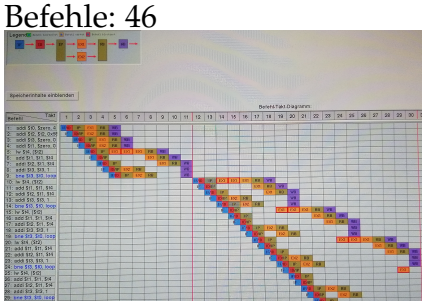
Beobachtung: jeweils mit 2 Bit Vorhersage und Result Forwarding

- Superskalar In-Order Pipeline (3 EX Einheiten)
  - Takte: 32
  - Befehle: 31
  - 
- Superskalar In-Order Pipeline (2 EX Einheiten)
  - 

  - Takte: 34
  - Befehle: 30
  - 
- Superskalar Out-of-Order (3 EX Einheiten)
  - 

  - Takte: 28

  - Befehle: 62
  - 

- Superskalar Out-of-Order (2 EX Einheiten)
  - Takte: 30
  - Befehle: 46
  - 

- Superskalar Out-of-Order (9 EX Einheiten)
  - Takte: 27
  - Befehle: 165

Zusatzaufgaben

Z1

Untersuche weitere Befehlsfolgen, z.B. aus A5, A6, A7, B1 oder nach eigenen Entwürfen!

Code A5

```
    addiu   $t1, $zero, 3       #$t1:=3
    addiu   $t2, $zero, 0       #$t2:=0
loop: addu  $t2, $t2, $t1       #$t2:=$t2+$t1
    addiu   $t1, $t1, -1        #$t1:=$t1-1
    bnez    $t1, loop           #branch loop (if $t1<>0)
    or      $t3, $zero, $t1     #$t3:=$t1
    sll     $t4, $t1, 2         #$t4:=$t1 << 2
    and     $t5, $t1, $t5       #$t5:=$t5 AND $t1
    or      $t6, $t1, $t6       #$t6:=$t6 OR $t1
```

Code A6

```
        addiu   $t1, $zero, 100
loop1:  addiu   $t2, $zero, 100
loop2:  addiu   $t2, $t2, -1
        ...
        ...
        bnez    $t2, loop2
        addiu   $t1, $t1, -1
        bne     $t1, 1, loop1
```

Code A7

```
        addiu   $t1, $zero, 991
loop:   ...
        addu    $t2, $zero, $t1
        and     $t2, $t2, 0x08
        bnez $t2, next
        ...
next:   ...
        addiu $t1, $t1, -1
        bne
        $t1, -1, loop
```

Code B1

```
add     $t5, $zero, $t2
add     $t4, $t6, $t5
add     $t3, $t7, $t3
lw      $t0, ($t3)
add     $t7, $zero, $t2
add     $t1, $t6, $t0
sw      $t5, ($t1)
sub     $t2, $t5, $t6
addi    $t4, $zero, 0
addi    $t3, $t3, 1
```

Z2

Nehme weitere Änderungen an Parametern und Pipelinestrukturen vor!

Z3

Versuche Befehlsfolgen zu finden, die die strukturellen Ressourcen besonders gut ausnutzen oder die Wirksamkeit bestimmter Methoden (wie z.B. Sprungvorhersagen) besonders gut sichtbar werden lassen!

Versuch M: Mikrocontroller

Assemblerprogrammierung mit dem 8-Bit-Mikrocontroller ATtiny25

Aufgabe 1: Ein- und Ausschalten der LED

Die LED soll über die beiden Taster ein-, aus- und umgeschaltet werden. Dazu ist eine funktionierende Teillösung vorgegeben, welche erweitert werden soll.

Schritt a: Start der Entwicklungsumgebung

Gebe das folgende Programm ein. Es soll die vorhandenen Befehle ersetzen.

```
.INCLUDE "tn25def.inc"      // Einfügen von Symbolen, u.a. für I/O-Register
.DEVICE ATtiny25            // Festlegen des Controllertyps
anf:
    ldi     r16,0x07
    out     DDRB,r16        // Port B: Richtungseinstellung
    ldi     r16,0x18
    out     PORTB,r16       // Port B: Pull-up für Taster-Eingänge aktivieren
lo1:
    sbis    PINB,PB4        // Abfrage TASTER1, Skip Folgebefehl wenn nicht gedrückt
    sbi     PORTB,0         // Einschalten der LED (blau)
    sbis    PINB,PB3        // Abfrage TASTER2, Skip Folgebefehl wenn nicht gedrückt
    cbi     PORTB,0         // Ausschalten der LED (blau)
    rjmp    lo1             // Sprung zum Schleifenbeginn
```

Schritt b: Manuelle Farbwechsel der LED

Das Programm soll jetzt so erweitert werden, dass die LED mit den beiden Tastern zwischen zwei Leuchtfarben umgeschaltet werden kann.

```
.INCLUDE "tn25def.inc"      // Einfügen von Symbolen, u.a. für I/O-Register
.DEVICE ATtiny25            // Festlegen des Controllertyps
anf:
    ldi     r16,0x07
    out     DDRB,r16        // Port B: Richtungseinstellung
    ldi     r16,0x18
    out     PORTB,r16       // Port B: Pull-up für Taster-Eingänge aktivieren
lo1:
    sbis    PINB,PB4        // Abfrage TASTER1, Skip Folgebefehl wenn nicht gedrückt
    jmp     blue
    sbis    PINB,PB3        // Abfrage TASTER2, Skip Folgebefehl wenn nicht gedrückt
    jmp     green
    rjmp    lo1             // Sprung zum Schleifenbeginn
blue:
    sbi     PORTB,0         // Einschalten der LED (blau)
    cbi     PORTB,1         // Ausschalten der LED (grün)
    rjmp    lo1
green:
    cbi     PORTB,0         // Ausschalten der LED (blau)
    sbi     PORTB,1         // Einschalten der LED (grün)
    rjmp    lo1
```

Verändere das Programm nun so, dass durch abwechselndes Drücken der beiden Taster eine Sequenz von mindestens sechs unterschiedlichen Leuchtvarianten der LED durchgeschaltet werden kann.

```
.INCLUDE "tn25def.inc"      // Einfügen von Symbolen, u.a. für I/O-Register
.DEVICE ATtiny25            // Festlegen des Controllertyps
anf:
    ldi     r16,0x07
    out     DDRB,r16        // Port B: Richtungseinstellung
    ldi     r16,0x18
    out     PORTB,r16       // Port B: Pull-up für Taster-Eingänge aktivieren
    ldi     r17,0x01        // Zähler
lo1:
    sbis    PINB,PB4        // Abfrage TASTER1, Skip Folgebefehl wenn nicht gedrückt
    rjmp    up
    sbis    PINB,PB3        // Abfrage TASTER2, Skip Folgebefehl wenn nicht gedrückt
    rjmp    down
    rjmp    lo1             // Sprung zum Schleifenbeginn
up:
    inc     r17
    cmp     r17, 0x07
    jnz     blue
    ldi     r17, 0x00
    rjmp    blue
down:
    dec     r17
    cmp     r17, 0x00
    jnz     blue
    ldi     r17, 0x06
blue:
    cmp     r17, 0x01
    jnz     cyan
    sbi     PORTB,0         // Einschalten der LED (blau)
    cbi     PORTB,1         // Ausschalten der LED (grün)
    cbi     PORTB,2         // Ausschalten der LED (rot)
    rjmp    lo1
cyan:
    cmp     r17, 0x02
    jnz     green
    sbi     PORTB,0         // Einschalten der LED (blau)
    sbi     PORTB,1         // Einschalten der LED (grün)
    cbi     PORTB,2         // Ausschalten der LED (rot)
    rjmp    lo1
green:
    cmp     r17, 0x03
    jnz     yellow
    cbi     PORTB,0         // Ausschalten der LED (blau)
    sbi     PORTB,1         // Einschalten der LED (grün)
    cbi     PORTB,2         // Ausschalten der LED (rot)
    rjmp    lo1
yellow:
    cmp     r17, 0x04
    jnz     red
    cbi     PORTB,0         // Ausschalten der LED (blau)
    sbi     PORTB,1         // Einschalten der LED (grün)
    sbi     PORTB,2         // Einschalten der LED (rot)
    rjmp    lo1
red:
    cmp     r17, 0x05
    jnz     violett
    cbi     PORTB,0         // Ausschalten der LED (blau)
    cbi     PORTB,1         // Ausschalten der LED (grün)
    sbi     PORTB,2         // Einschalten der LED (rot)
    rjmp    lo1
violett:
    sbi     PORTB,0         // Einschalten der LED (blau)
    cbi     PORTB,1         // Ausschalten der LED (grün)
    sbi     PORTB,2         // Einschalten der LED (rot)
    rjmp    lo1
```

Aufgabe 2: Blinken der LED

Das Programm soll die LED fortlaufend blinken lassen. Diese Funktion wird mit einem Zähler/Zeitgeber-Interrupt realisiert.

Schritt a: Einfaches Blinken

Die Aufgabe besteht nun darin, die LED periodisch ein- und auszuschalten, so dass sich eine Frequenz von etwa 2 Hz ergibt. Das Umschalten der LED soll in der Interruptserviceroutine eines Zähler/Zeitgeber-Interrupts erfolgen. Dafür soll Timer/-Counter 0 so initialisiert werden, dass er Interrupts mit einer Folgefrequenz von etwa 4 Hz auslöst.

```
// Interrupttabelle (muss vor dem ersten ausführbaren Befehl stehen):
tab: rjmp   anf // Programmstart nach Reset ("Interrupt" 1)
    reti
    reti
    reti
    reti
    reti
    reti
    reti
    reti
    reti
    rjmp    i_11 // Timer 0 Compare A Interrupt (Interrupt 11)
    reti
    reti
    reti
    reti    // Tabellenende (Interrupt 15)

// Initialisierungsteil und Hintergrundprogramm:
anf: [...]  // Weitere Initialisierungen
     [...]  // Initialisierung von Timer/Counter 0 (Empfehlung:
            // Betriebsart CTC, Vergleichsregister A nutzen)
    sei     // Globale Interruptfreigabe
    ldi r16,0x10
    out TIMSK,r16   // Freigabe von Interrupt 11 (Timer 0 Compare A)
lo2: rjmp   lo2     // Leere Hintergrundschleife

// Interruptserviceroutine:
i_11: in    r25,SREG    // Flags retten (weitere Rettungen nach Bedarf)
    [...]               // Inhalt der Routine
    out     SREG,r25    // Flags restaurieren
    reti                // Routine beenden
```

Die Hintergrundschleife bleibt zunächst leer. Entwickle und teste das Programm für diese Aufgabe.

```
.INCLUDE "tn25def.inc"      // Einfügen von Symbolen, u.a. für I/O-Register
.DEVICE ATtiny25            // Festlegen des Controllertyps
// Interrupttabelle (muss vor dem ersten ausführbaren Befehl stehen):
tab: rjmp   anf // Programmstart nach Reset ("Interrupt" 1)
    reti
    reti
    reti
    reti
    reti
    reti
    reti
```

```
    reti
    reti
    rjmp    timer_compare // Timer 0 Compare A Interrupt (Interrupt 11)
    reti
    reti
    reti
    reti    // Tabellenende (Interrupt 15)

    // Initialisierungsteil und Hintergrundprogramm:
anf: [...]  // Weitere Initialisierungen
    // Initialisierung von Timer/Counter 0
    ldi     r16, high( 40000 - 1 )
    out     OCR1AH, r16
    ldi     r16, low( 40000 - 1 )
    out     OCR1AL, r16
    // CTC Modus einschalten, Vorteiler auf 1
    ldi     r16, ( 1 << WGM12 ) | ( 1 << CS10 )
    out     TCCR1B, r16
    // OCIE1A: Interrupt bei Timer Compare
    ldi     r16, 1 << OCIE1A
    out     TIMSK, r16

    sei     // Globale Interruptfreigabe
    //ldi r16,0x10
    //out TIMSK,r16   // Freigabe von Interrupt 11 (Timer 0 Compare A)

    ldi     r16,0x07
    out     DDRB,r16    // Port B: Richtungseinstellung
    ldi     r16,0x18
    out     PORTB,r16       // Port B: Pull-up für Taster-Eingänge aktivieren

    ldi     r17, 0x00 // Zähler
lo2:
    rjmp    lo2     // Leere Hintergrundschleife

    // Interruptserviceroutine:
timer_compare:
    in      r25,SREG      // Flags retten (weitere Rettungen nach Bedarf)
    inc     r17
    cmp     r17, 0x02
    jnz     on
off:
    ldi     r17, 0x00
    cbi     PORTB,0     // Ausschalten der LED (blau)
    rjmp    close
on:
    sbi     PORTB,0     // Einschalten der LED (blau)
close:
    out     SREG,r25    // Flags restaurieren
    reti                // Routine beenden
```

Alternativ

```
init:
    ; Modus 14:
    ldi     r17, 1<<COM1A1 | 1<<WGM11
    out     TCCR1A, r17
    ldi     r17, 1<<WGM13 | 1<<WGM12 | 1<<CS12
    out     TCCR1B, r17

    ldi     r17, 0x6F
    out     ICR1H, r17
    ldi     r17, 0xFF
    out     ICR1L, r17

    ; der Compare Wert
    ldi     r17, 0x3F
    out     OCR1AH, r17
    ldi     r17, 0xFF
    out     OCR1AL, r17

    ; Den Pin OC1A auf Ausgang schalten
```

```
    ldi     r17, 0x02
    out     DDRB, r17
main:
    rjmp    main
```

Alternativ

```
init:
    LDI r16, 0x07
    STS DDRB, r16
    LDI r17, 0x18
    OUT PORTB, r17
    LDI r16, 0x00
    STS TCCR1A, r16
    RET
main:
    LDI r16, 0xF0
    STS TCNT1H, r16
    LDI r16, 0xBC
    STS TCNT1L, r16
    LDI r16, 0x05
    STS TCCR1B, r16
loop:  LDS R0, TIFR1
    SBRS R0, 0
    RJMP loop
    LDI r16, 0x00
    STS TCCR1B, r16
    LDI r16, 0x01
    STS TIFR1, r16
    COM r17
    STS PORTB, r17
    RET
```

Schritt b: Erweitertes Blinken

Baue in die Hintergrundschleife eine Abfrage von TASTER1 und TASTER2 ein. Durch Drücken von TASTER1 soll die Blinkfrequenz verdreifacht werden, durch TASTER2 wird sie auf den ursprünglichen Wert zurückgestellt. Teste diese Funktion. Der Vorgang soll sich beliebig wiederholen lassen.

Stelle das Programm nun so um, dass die beiden Blinkfrequenzen deutlich langsamer sind: Etwa 1,0 Hz und etwa 0,5 Hz. Beachte, dass der Zählumfang des Timer/-Counter dafür nicht ausreicht, auch nicht mit dem größten Vorteiler. Das Programm muss also in der Struktur verändert werden. Erweitere das Programm so, dass eine Sequenz aus mindestens vier unterschiedlichen Leuchtzuständen durchlaufen wird.

$$Verzoegerungswert = 2^{16} - \frac{frequenz \times delaytime}{prescaler}$$

16-Bit Wert in zwei 8-Bit teilen und in TCNT1H und TCNT1L laden

```
init:
    LDI r16, 0x07
    OUT DDRB, r16
    LDI r17, 0x18
```

```
    OUT PORTB, r17
    LDI r16, 0x00
    STS TCCR1A, r16    ; alle bits von TCCR1A auf 0
    RET
main:
    LDI r16, 0xF0
    STS TCNT1H, r16      ; timer high register
    LDI r16, 0xBC
    STS TCNT1L, r16      ; timer low register
    LDI r16, 0x05
    STS TCCR1B, r16      ; use 1024 prescalar
loop:  LDS R0, TIFR1      ; TIFR1 in R0 laden
    SBRS R0, 0          ; skippen falls overflow
    RJMP loop           ; schleife bis overflow
    LDI r16, 0x00
    STS TCCR1B, r16    ; stoppe Timer/Counter1
    LDI r16, 0x01
    STS TIFR1, r16     ; overflow zurücksetzten
    COM r17            ; complement r17
    STS PORTB, r17     ; toggle LED
    sbis PINB, PB4     ; skip folgebefehl wenn nicht gedrückt
    rjmp t1
    sbis PINB, PB3     ; skip folgebefehl wenn nicht gedrückt
    rjmp t2
    RJMP loop
t1:
    LDI r16, 0xFB
    STS TCNT1H, r16      ; timer high register
    LDI r16, 0x6C
    STS TCNT1L, r16      ; timer low register
    RJMP loop
t2:
    LDI r16, 0xF0
    STS TCNT1H, r16      ; timer high register
    LDI r16, 0xBC
    STS TCNT1L, r16      ; timer low register
    RJMP loop
```

Aufgabe 3: Einfaches Dimmen der LED mittels PWM

Stelle die Helligkeit der LED mittels PWM (pulse width modulation, Pulsbreitenmodulation) auf wählbare Zwischenwerte ein.

Schritt a: Einfache Helligkeitseinstellung

Zunächst soll die LED (nur eine Farbe) auf eine beliebige, aber konstante Helligkeit eingestellt werden können. Realisiere dazu eine PWM-Ausgabe mit 256 Helligkeitsstufen, wobei die Zeitintervalle wahlweise mittels Zählschleifen oder mittels Timer/Counter-Interrupt generiert werden. Der Helligkeitswert kann über ein Universalregister vorgegeben werden, in welches im Debugger bei gestopptem Programm jeweils unterschiedliche Werte eintragen werden. Alternativ können auch die PWM-Betriebsarten der Timer/Counter-Baugruppen ausprobiert werden, soweit es die Hardwarekonfiguration zulässt. Empfohlen wird die Betriebsart „Fast PWM"

mit normaler Zählung.

```
init:
    ldi r16,0xff
    out DDRB,r16
    cbi PORTB,0
    ldi r17, 25             ; r17 ist helligkeitswert
l1:
    sbi PORTB, 0            ; LED an
    mov r16, r17            ; R16 kontrolliert länge des delay (= r17)
    rcall delay
    cbi PORTB, 0            ; LED aus
    ldi r16, 255
    sub r16, r17            ; R16 kontrolliert länge des delay  (= 255 - r17)
    rcall delay
    rjmp l1
; Delay for (R16 * 4) microseconds
delay:
    tst r16                 ; R16 = 0? (no delay)
    breq dly4
dly2:
    ldi r24,low(16)
    ldi r25,high(16)
dly3:
    sbiw r24,1              ; 2 cycles
    brne dly3               ; 2 cycles
    dec r16
    brne dly2
dly4:
    ret
```

Schritt b: Helligkeitseinstellung mit Tastern

Nun sollen die beiden Taster als Bedienelemente zum Auf- und Abdimmen verwendet werden. Werte dabei die Dauer der Tastendrücke aus, nicht deren Anzahl. Die Helligkeit soll bei gedrückt gehaltenem Taster stetig zu- oder abnehmen. Bei losgelassenen Tastern soll die Helligkeit konstant bleiben.

```
init:
    ldi r16,0x07            ; Port B Richtungseinstellung
    out DDRB,r16
    ldi r16, 0x18
    out PORTB, r16          ; Pull Up für Taster
    cbi PORTB, 0            ; LED aus
    ldi r17, 25             ; r17 ist helligkeits wert
l1:
    sbi PORTB, 0            ; LED an
    mov r16, r17            ; R16 kontrolliert länge des delay (= r17)
    rcall delay
    cbi PORTB, 0            ; LED aus
    ldi r16, 255
    sub r16, r17            ; R16 kontrolliert länge des delay  (= 255 - r17)
    rcall delay
    sbis PINB, PB4
    inc r17
    sbis PINB, PB3
    dec r17
    rjmp l1
; Delay for (R16 * 4) microseconds
delay:
    tst r16                 ; R16 = 0? (no delay)
    breq dly4
dly2:
    ldi r24,low(16)
```

```
    ldi r25,high(16)
dly3:
    sbiw r24,1              ; 2 cycles
    brne dly3               ; 2 cycles
    dec r16
    brne dly2
dly4:
    ret
```

Zusatzaufgabe: Fortlaufendes Auf- und Abdimmen der LEDs

Diese Aufgabe soll als Anregung für weiterführende Experimente nach eigenen Ideen dienen. Die Helligkeit der LED soll in einer geeigneten Geschwindigkeit stetig herauf- und heruntergeregelt werden, so dass ein „weiches Blinken" entsteht. Dazu muss einen Mechanismus implementiert werden, der den Helligkeitswert nach einem geeigneten Zeitschema verändert. Realisiere weitere Lichteffekte dieser Art, bei denen nun auch mehrere Leuchtfarben beteiligt sind. Realisiere eine Umschaltung zwischen unterschiedlichen Lichteffekten. Realisiere weitergehende Funktionen nach eigenen Ideen.

```
init:
    ldi r16,0xff
    out DDRB,r16
    cbi PORTB,0

; LED von low zu high
dopwm:
    ldi r17,25
l1:
    ldi r18, 0x01           ; R18 zählt PWM cycles
l2:
    cbi PORTB,0
    mov r16,r17
    rcall delay
    sbi PORTB,0
    ldi r16,255
    sub r16,r17
    rcall delay
    dec r18
    brne l2
    inc r17                 ; helligkeit erhöhen
    brne l1

; LED von high zu low
    ldi r17,255
l3:
    ldi r18, 0x01
l4:
    cbi PORTB,0
    mov r16,r17
    rcall delay
    sbi PORTB,0
    ldi r16,255
    sub r16,r17
    rcall delay
    dec r18
    brne l4
    dec r17                 ; helligkeit runter
    cpi r17,25
    brne l3
    rjmp dopwm

; Delay for (R16 * 4) microseconds
delay:
    tst r16
    breq dly4
dly2:
    ldi r24,low(16)
    ldi r25,high(16)
dly3:
    sbiw r24,1              ; 2 cycles
    brne dly3               ; 2 cycles
```

```
    dec r16
    brne dly2
dly4:
    ret
```