

Technische Universität Ilmenau  
Fakultät IA  
Fachgebiet Rechnerarchitektur

Praktikum Rechnerarchitektur 2  
WS 2021/22

# Versuchsprotokoll

**Versuche Befehlsausführung und Mikrocontroller**

21. Januar 2022

## Versuch B: Befehlsausführung

Simulative Untersuchung der Ausführung von Maschinenbefehlen in unterschiedlichen Pipeline-Architekturen

### Aufgabe 1

Untersuchen die vorbereitete Befehlsfolge mit den drei vorgegebenen Grundstrukturen Standard-Pipeline, Superskalar-in-Order und Superskalar-out-of-Order. Beachte den Programmablauf und machen dich mit der Bedienung vertraut! Schauen vor dem Simulationsstart auch die Parametereinstellungen für Sprungvorhersage und Result Forwarding an (hier können auch Änderungen vorgenommen werden) 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

Untersuchen Sie die Befehlsfolgen A4 und B2 mit mindestens je drei unterschiedlichen Simulationsläufen! Die benutzten Pipelinestrukturen und Parametereinstellungen wählen Sie selbst aus. Vergleichen Sie die Ergebnisse mit den Lösungen aus der Übung und suchen Sie Erklärungen für eventuelle Unterschiede!

### Code A4

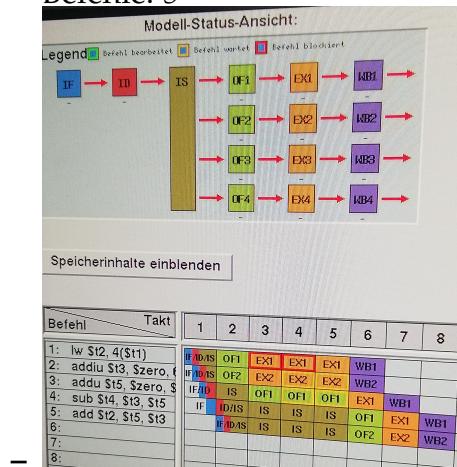
```

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

```

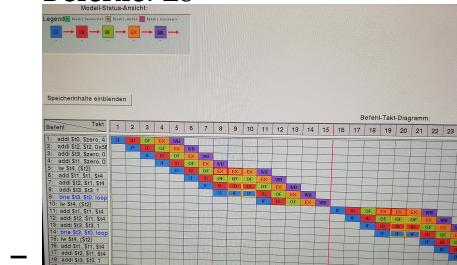
# addition der inhalte von 4 aufeinander folgenden speicherzellen , beginnend mit adresse 0x12345678 ...
# $t2 enthalte bereits den wert 0x12340000
    addi    $t0, $zero, 4          # max. zaehlerwert t0 = 4
    addi    $t2, $t2, 0x5678       # adressregister t2 = startadresse
    addi    $t3, $zero, 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

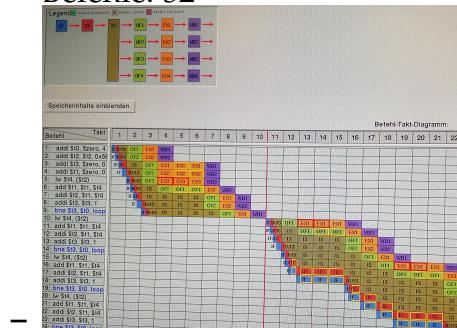
- Befehle: 28



- Superskalar In-Order Pipeline (4 EX Einheiten)

- Takte: 31

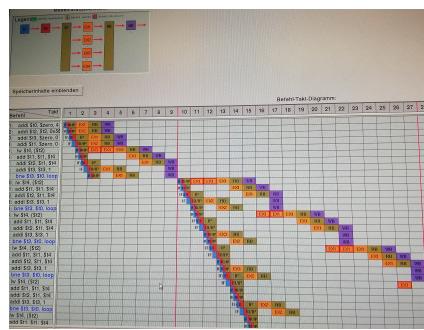
- Befehle: 32



- Superskalar Out-of-Order ( 4 EX Einheiten)

- Takte: 27

- Befehle: 80

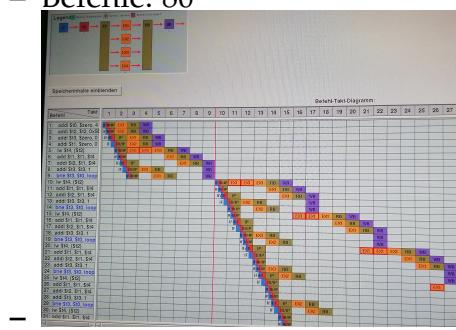


### 1 Bit Vorhersage

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

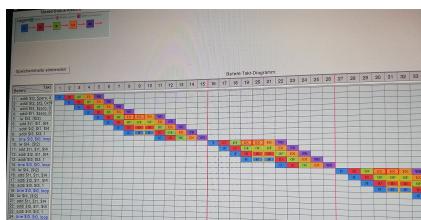


- Superskalar Out-of-Order ( 4 EX Einheiten)
  - Takte: 27
  - Befehle: 80

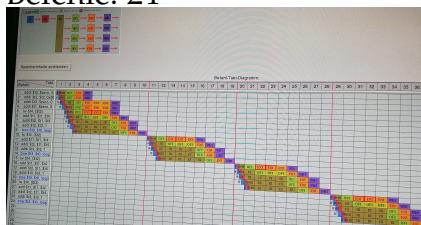


### 0 Bit Vorhersage

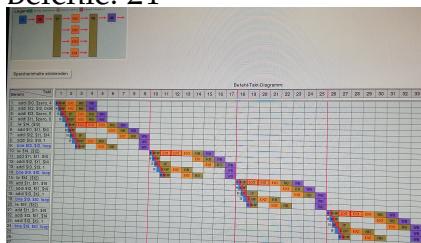
- Standard Pipeline
  - Takte: 48
  - Befehle: 24



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

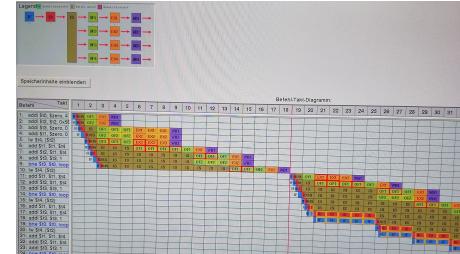


- Superskalar Out-of-Order ( 4 EX Einheiten)
  - Takte: 33
  - Befehle: 24



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

- Takte: 57
- Befehle: 32



## Aufgabe 3

Ändern Sie nun eine der vorgegebenen Pipelinestrukturen ab, indem Sie z.B. die Anzahl der parallelen Pipelines verändern. Orientieren Sie sich zuvor über den In-

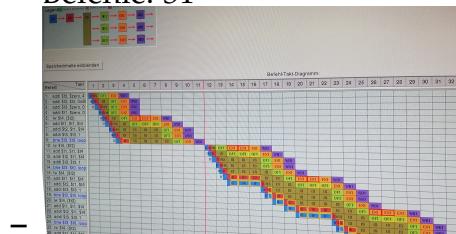
halt des „Baukastens“. Untersuchen Sie mit den oben verwendeten Befehlsfolgen die Auswirkungen auf die Simulationsergebnisse! Variieren Sie dabei die Parameter und interpretieren Sie die Ergebnisse!

**Beobachtung:** jeweils mit 2 Bit Vorhersage und Result Forwarding

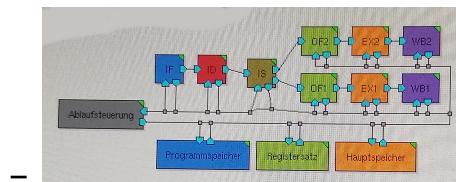
- Superskalar In-Order Pipeline (3 EX Einheiten)

– Takte: 32

– Befehle: 31

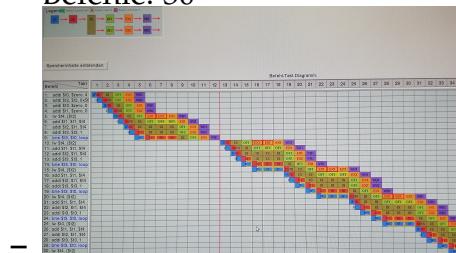


- Superskalar In-Order Pipeline (2 EX Einheiten)

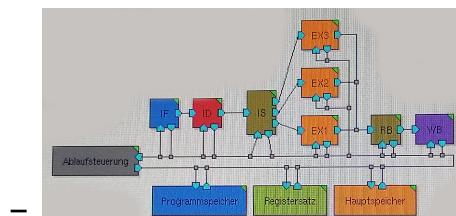


– Takte: 34

– Befehle: 30

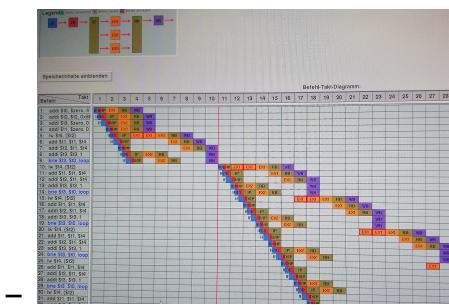


- Superskalar Out-of-Order ( 3 EX Einheiten)



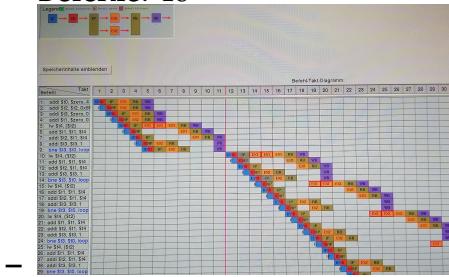
– Takte: 28

– 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

Untersuchen Sie 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

Nehmen Sie weitere Änderungen an Parametern und Pipelinestrukturen vor!

## Z3

Versuchen Sie 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

In dieser Teilaufgabe soll die LED über die beiden Taster ein-, aus- und umgeschaltet werden. Dazu ist eine funktionierende Teillösung vorgegeben, welche Sie anschließend erweitern sollen.

#### Schritt a: Start der Entwicklungsumgebung

Geben Sie 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:
    ldir    16,0x07
    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.

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

### Aufgabe 2: Blinken der LED

In dieser Teilaufgabe soll das Programm 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.

Stützen Sie sich dazu auf das folgende Programmfragment:

```
// Interrupttabelle (muss vor dem ersten ausführbaren Befehl stehen):
tab: rjmp  anf // Programmstart nach Reset ("Interrupt" 1)
      reti
      reti
      reti
      reti
      reti
      reti
      reti
      reti
      rjmp  i_11 // Timer 0 Compare A Interrupt (Interrupt 11)
      reti
      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 TMSK,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. Entwickeln und testen Sie das Programm für diese Aufgabe.

## Schritt b: Erweitertes Blinken

Bauen Sie 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. Testen Sie diese Funktion. Der Vorgang soll sich beliebig wiederholen lassen. Stellen Sie das Programm nun so um, dass die beiden Blinkfrequenzen deutlich langsamer sind: Etwa 1,0 Hz und etwa 0,5 Hz. Beachten Sie, dass der Zählumfang des Timer/Counter

dafür nicht ausreicht, auch nicht mit dem größten Vorteiler. Sie müssen das Programm also in der Struktur verändern. Erweitern Sie das Programm so, dass eine Sequenz aus mindestens vier unterschiedlichen Leuchtzuständen durchlaufen wird. Sie können dabei nach Belieben auch komplexere Abläufe realisieren.

Optional: Stellen Sie die Tasterabfrage auf Interruptbetrieb um. Hierfür bietet sich der „Pin Change Interrupt“ an. Beachten Sie, dass wegen des Prellens der Taster ein einzelner Tastendruck eine ganze Serie von Interrupts auslösen kann.

Optional: Realisieren Sie eine Entprellung der Taster. Dazu ist nach jeder erkannten Änderung des Zustands eines Tasters eine Wartezeit von z.B. 0,2 s nötig, während derer weitere Zustandswechsel dieses Tasters ignoriert werden. Auf der Grundlage entprellter Taster können Sie erweiterte Umschaltfunktionen implementieren, bei denen auch das mehrmalige Drücken desselben Tasters eine Rolle spielt.

### **Aufgabe 3: Einfaches Dimmen der LED mittels PWM**

In dieser Aufgabe soll die Helligkeit der LED mittels PWM (pulse width modulation, Pulsbreitenmodulation) auf wählbare Zwischenwerte eingestellt werden.

#### **Schritt a: Einfache Helligkeitseinstellung**

Zunächst soll die LED (nur eine Farbe) auf eine beliebige, aber konstante Helligkeit eingestellt werden können. Realisieren Sie dazu eine PWM-Ausgabe mit 256 Helligkeitsstufen, wobei Sie die Zeitintervalle wahlweise mittels Zählschleifen oder mittels Timer/Counter-Interrupt generieren. Den Helligkeitswert können Sie über ein Universalregister vorgeben, in welches Sie im Debugger bei gestopptem Programm jeweils unterschiedliche Werte eintragen. Alternativ können Sie auch die PWM-Betriebsarten der Timer/Counter-Baugruppen ausprobieren, soweit es die Hardwaredokumentation zulässt. Empfohlen wird die Betriebsart „Fast PWM“ mit normaler Zählung.

#### **Schritt b: Helligkeitseinstellung mit Tastern**

Nun sollen die beiden Taster als Bedienelemente zum Auf- und Abdimmen verwendet werden. Werten Sie 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.

### **Zusatzaufgabe: Fortlaufendes Auf- und Abdimmen der LEDs**

Diese Aufgabe soll als Anregung für weiterführende Experimente nach eigenen Ideen dienen. In Absprache mit dem Betreuer können Sie andere Aufgabenteile weglassen, falls Sie die Zusatzaufgabe unter Einbeziehung umfangreicher eigener Ideen in Angriff nehmen wollen.

Die Helligkeit der LED soll in einer geeigneten Geschwindigkeit stetig herauf- und heruntergeregt werden, so dass ein „weiches Blinken“ entsteht. Dazu müssen Sie einen Mechanismus implementieren, der den Helligkeitswert nach einem geeigneten Zeitschema verändert. Realisieren Sie weitere Lichteffekte dieser Art, bei denen nun auch mehrere Leuchtfarben beteiligt sind. Realisieren Sie eine Umschaltung zwischen unterschiedlichen Lichteffekten. Realisieren Sie weitergehende Funktionen nach eigenen Ideen.