

Cedric Aebi 17-103-235

Nicolas Müller 17-122-094

RA FS 18 Serie 5

Qiyang Hu, Givi Meishvili, Adrian Wälchli

Die fünfte Serie ist bis Dienstag, den 15. Mai 2018 um 15:00 Uhr zu lösen und auf ILIAS hochzuladen. Für Fragen steht im ILIAS jederzeit ein Forum zur Verfügung. Allfällige unlösbare Probleme sind uns so früh wie möglich mitzuteilen, wir werden gerne helfen.
Viel Spass!

Theorieteil

Gesamtpunktzahl: 10 Punkte

1 Static Branch Structure (3 points)

Welche Probleme treten bei folgendem Assembler-Loop auf, wenn standardmäßig *Predict Not Taken* gewählt wird.

```
1      addi $s1, $zero, 1024 // s1 := 1024
2 loop: addi $s1, $s1, -1 // s1--
3      jal subroutine // call subroutine()
4      bne $s1, $zero, loop // if (s1 != 0) jump loop
```

Optimieren Sie das Codebeispiel auf *Predict not Taken*.

2 Antidependencies (1 point)

Erklären Sie was eine Antidependency ist.

3 Branch Delay Slots (1 point)

Erklären Sie was ein Branch-Delay-Slot ist.

4 Dynamic Branch Structure (2 points)

Gegeben das folgende C Programm.

```
1 int t = 0;
2 for( int i = 0; i < 10; i++ )
3 {
4     for( int j = 0; j < 10; j++ )
5     {
6         t = t + i*j;
7     }
8 }
```

Nehmen Sie an, der Prozessor braucht *Dynamic Branch Prediction* und *Taken* ist genommen per Default.

- Wie viele falsche Predictions geschehen, wenn ein 1-bit Counter benutzt ist? Erläutern Sie ihre Berechnung.
- Wie viele falsche Predictions geschehen, wenn ein 2-bit Counter benutzt ist? Erläutern Sie ihre Berechnung.

5 Multiple-Issue Processors (3 point)

Bestimmen Sie, ob die folgenden Sätze wahr oder falsch sind. Bitte erläutern Sie kurz ihre Antwort.

- Pipelining benutzt Instruction-Level-Parallelism, doch es ermöglicht dem Prozessor **nicht**, mehr als eine Instruktion pro Zeitpunkt auszuführen.
- Multiple-issue Prozessoren können mehr als 1 Instruktion pro Clock-Cycle ausführen.
- SIMD (single-instruction multiple-data) Prozessoren sind statische Multiple-Issue Prozessoren.

Programmierteil

Die Programmieraufgaben mit dem Raspberry Pi sind in Zweier- oder Dreiergruppen zu lösen. Sie und Ihre Gruppenmitglieder arbeiten gemeinsam am Code oder teilen sich Unteraufgaben auf. Um sicherzustellen, dass jeder Programmierer den gesamten Code verstehen und erklären kann, verlangen wir von jedem Mitglied eine leicht unterschiedliche Abgabe. Wie unten beschrieben unterscheiden sich die Versionen nur um eine spezielle Funktionalität für die jeder Teilnehmer selbst verantwortlich ist.

Paddle Ball

In dieser Serie geht es darum eine digitale Version des Spiels *Paddle Ball*¹ zu implementieren. Das Spiel besteht im Wesentlichen aus den folgenden Teilen:

Startzustand Der Spieler startet das Spiel mit Taste **BUTTON2**. Damit wird die Punktezahl auf null zurückgesetzt.

Spielphase Die leuchtende LED (“der Ball”) wandert in eine Richtung. Richtungswechsel ist bei **LED0** bzw. **LED7**, wo der Spieler im richtigen Moment die Taste **BUTTON1** betätigen muss um den Ball in die Gegenrichtung zu schlagen. Dabei können zwei Fälle auftreten:

Ball getroffen Hat der Spieler innerhalb des Zeitfensters **BUTTON1** gedrückt, wird ein Punkt angerechnet. Zusätzlich nimmt die Geschwindigkeit des Lauflichts zu (nächstes “Level”).

Ball verfehlt Verpasst der Spieler das Zeitfenster, dann ertönt ein akustisches Signal durch den Signaltongeber (Buzzer). In diesem Fall wird die Punktezahl nicht erhöht und das Spiel läuft weiter.

Game Over Das Spiel endet sobald die maximale Geschwindigkeit (höchstes Level) erreicht wurde. Die Punktezahl erscheint in binärem Format auf der LED Anzeige.

Details

Das Spiel kann mit der Taste **BUTTON2** gestartet werden. Damit bewegt sich das Licht von links nach rechts, dann wieder nach links und so weiter. Sie dürfen die Startposition und Startrichtung wählen. Wir empfehlen, dass Sie den Code aus Serie 4 zur Animation des Lichts wiederverwenden.

Zum Zeitpunkt an dem entweder **LED0** oder **LED7** aufleuchtet muss der Spieler mit der Taste **BUTTON1** interagieren. Die Punktezahl erhöht sich nicht, wenn die Taste zu früh oder zu spät gedrückt wurde. Stattdessen ertönt ein Warnsignal vom Summer. Im Anhang finden Sie weitere Informationen zur Verwendung des Signaltongebbers.

Mit jedem Richtungswechsel wird die Geschwindigkeit des Lauflichts erhöht. Sie können selbst wählen wie gross die Zunahme der Geschwindigkeit ist. Nachdem eine von Ihnen gewählte Maximalgeschwindigkeit erreicht wurde endet das Spiel und der Punktestand wird als Binärzahl angezeigt. Mit Taste **BUTTON2** kann das Spiel erneut gestartet werden.

Sie können das Spiel mit Ihren eigenen Ideen erweitern. Vergessen Sie aber nicht alle Änderungen im Detail zu dokumentieren.

Aufgaben

- Erweitern Sie das Lauflicht aus Serie 4 zu einem “Paddle Ball” Spiel wie oben beschrieben.

Student 1: Sie geben die beschriebene Standardversion des Spiels ab.

Student 2: Sie modifizieren die Standardversion so, dass die Signalisierung beim Verfehlten des Balles umgekehrt ist. Das heisst, es wird ein Ton ausgegeben, wenn ein Punkt erzielt wird.

Student 3: In Ihrer Version sollen Sie die Anzeige der Punktzahl invertieren, so dass die ausgeschalteten LED Segmente den binären Punktestand anzeigen.

- Stellen Sie sicher, dass Ihr Assemblerprogramm *ausführlich und sinnvoll* kommentiert ist. Als Richtwert gilt, dass jede Zeile kommentiert werden soll. Sie können zwei Zeilen zusammenfassen, falls dies Sinn macht. Dies ist eine notwendige Voraussetzung, damit der Programmierteil als erfüllt gilt.

¹https://en.wikipedia.org/wiki/Paddle_ball

- (c) Stellen Sie ausserdem sicher, dass Ihre Implementation ohne Fehler und Warnungen kompilierbar ist. Dies ist eine notwendige Voraussetzung, damit der Programmienteil als erfüllt gilt.
- (d) Erstellen Sie aus Ihrer Lösung eine Zip-Datei namens <nachname>.zip, wobei <nachname> durch Ihren Nachnamen zu ersetzen ist.
- (e) Geben Sie die Zip-Datei elektronisch durch Hochladen in ILIAS ab.
- (f) Denken Sie daran, dass der Raspberry Pi am Dienstag, den 15. Mai 2018 zurückgegeben werden muss. Bringen Sie alle Geräte bitte in die Übungsstunde, Sie erhalten dann ebenfalls die Kaution in der Höhe von 100 Franken zurück. Sollte es Ihnen nicht möglich sein, an der Rückgabe teilzunehmen, setzen Sie sich bitte frühzeitig mit den Assistenten in Verbindung.

Bonusaufgabe: Timer

Diese Aufgabe ist **optional**. Programmieren Sie einen einfachen Kurzzeitwecker, auch bekannt als "Eieruhr". Die Bedienung der Uhr soll wie folgt implementiert werden.

Startzustand Im Startzustand ist der Timer ausgeschaltet, und die LED auf dem ersten (untersten) Segment leuchtet auf.

Zeitstellung Durch Drücken der Taste BUTTON1 lässt sich das Licht auf der Segmentanzeige aufwärts bewegen. Die Position des Lichts gibt die Anzahl Minuten an, auf die der Timer eingestellt soll. Beispiel: Bei zweimaligem Drücken der Taste ist die dritte LED eingeschaltet, und somit ist der Wecker auf drei Minuten gestellt.

Wartezeit Durch Drücken der Taste BUTTON2 wird der Timer mit der eingestellten Zeit gestartet. Die aktive LED soll blinken um zu signalisieren, dass der Timer am Laufen ist. Zusätzlich soll bei Ablauf jeder Minute die (blinkende) LED ein Segment zurückgehen damit man ablesen kann, wieviel Zeit etwa noch verbleibt.

Alarm Ist die eingestellte Zeit abgelaufen (damit ist die aktive LED zurück in der Startposition), so ertönt ein Alarmsignal über den Tongeber. Durch Drücken der Taste BUTTON1 kann der Alarm ausgeschaltet werden, und der Wecker wird wieder in den Startzustand versetzt.

Sie dürfen die Bedienung des Timers natürlich auch ändern und durch Ihre eigenen Ideen erweitern. Viel Spass!

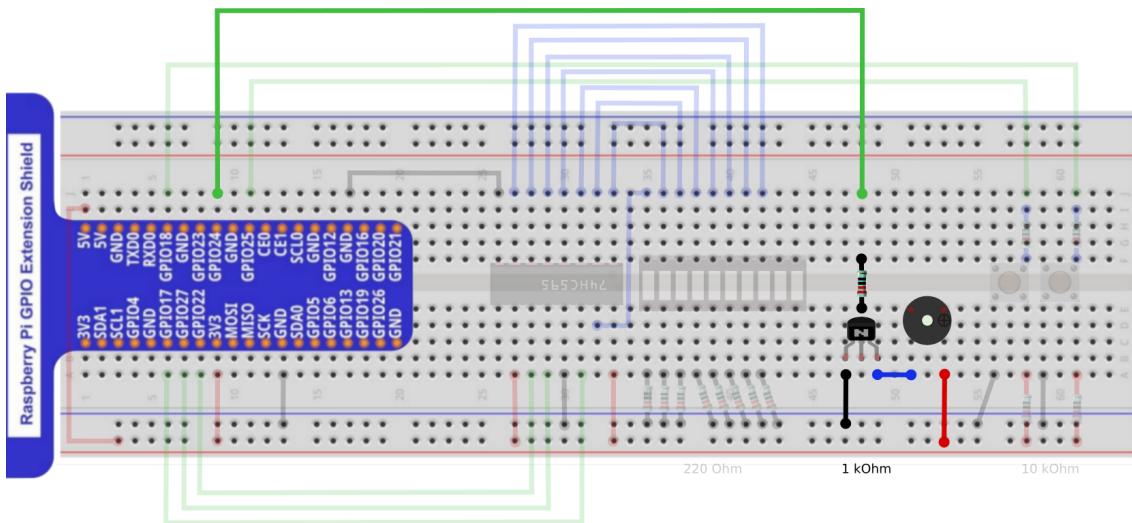


Abbildung 1: Bauplan für den Tongeber/Summer mit NPN-Transistor. Rot: Stromverbindung zu positiver Elektrode (+). Schwarz: Stromverbindung zu negativer Elektrode (-). Grün: Datenleitungen für Eingangs- und Ausgangssignale. Blau: Sonstige Datenleitungen.

A Signaltongeber

Der Signaltongeber, oder auch Summer genannt, wird genau gleich wie die LED bedient. Wie in Abbildung 1 gezeigt ist dieser an den Pin 24 angeschlossen. Mit `digitalWrite` können Sie das Ausgangssignal auf HIGH (Tonausgabe) oder LOW (kein Ton) stellen.

Ist der Tongeber eingeschaltet, gibt er einen konstanten Ton aus. Falls Sie das akustische Signal beim Programmieren stört, dann können Sie provisorisch die LED (siehe Einführungsbeispiel) als Ersatz verwenden. Entfernen Sie dazu das grüne Kabel in Abbildung 1 und verbinden Sie die LED mit Pin 24 (anstelle Pin 21 vorher).

Wenn Sie wollen können Sie für den Signaltongeber auch die 5V Stromzufuhr verwenden (anstelle 3V). Dadurch wird die Lautstärke erhöht. Sie müssen dazu das rote Kabel in Abbildung 1 auf den Pin umstecken, der mit "5V" angeschrieben ist.

1 Static Branch Structure (3 points)

Welche Probleme treten bei folgendem Assembler-Loop auf, wenn standardmäßig *Predict Not Taken* gewählt wird.

```
1      addi $s1, $zero, 1024 // s1 := 1024
2  loop: addi $s1, $s1, -1 // s1--
3      jal subroutine // call subroutine()
4      bne $s1, $zero, loop // if (s1 != 0) jump loop
```

Optimieren Sie das Codebeispiel auf *Predict not Taken*.

Weil die Struktur "Bottom of the loop" ist, und *Predict Not Taken* standardmäßig gebraucht wird, werden unnötigerweise ein paar Instruktionen am Ende der Schlaufe ausgeführt. Wenn nun 1023 mal fälschlicherweise zu viele Instruktionen ausgeführt werden, führt dies zu einem performanceloss (nur beim letzten Loop liegt die Annahme *Predict Not Taken* richtig, und die nachfolgenden Instruktionen können gebraucht werden)

Optimierung: "Top of the loop"

add: \$s1, \$zero, 1024

Loop: beq \$s1, \$zero, Out

add: \$s1, \$s1, -1

jal Subroutine

j Loop

Out: fall out instr



2 Antidependencies (1 point)

Erklären Sie was eine Antidependency ist.

An Out-Of-Order-Issue hat auch mit Data Antidependency zu kämpfen. Antidependency = (write before read). Wenn eine spätere Instruktion (welche zuerst beendet) ein Datenwert generiert, welcher einen anderen Datenwert (gebraucht als Quelle einer früheren Instruktion (that issues later)) zerstört.

$$R3 := \textcircled{RJ} * RS$$

$$R4 := R3 + 1 \quad \text{Antidependency}$$

$$\textcircled{R3} := RS + 1$$

Antidependencies arise because the limited number of registers mean that programmers reuse registers for different computations.

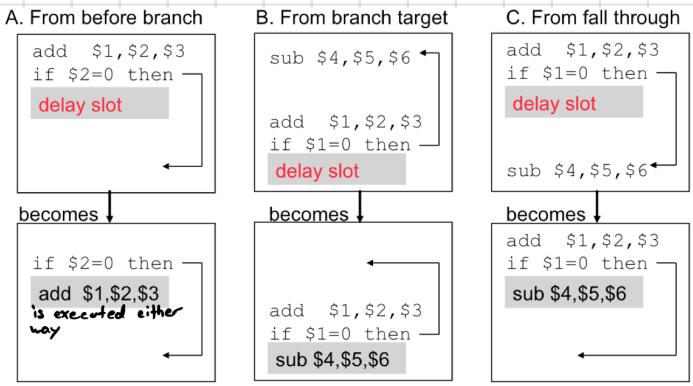
3 Branch Delay Slots (1 point)

Erklären Sie was ein Branch-Delay-Slot ist.



Eine Branch Delay Instruktion ist ein Befehl, welcher direkt einem bedingten Verzweigungsbefehl folgt und unabhängig davon, ob die Verzweigung genommen wurde oder nicht, immer ausgeführt wird.

Die Position eines solchen Befehls in der Pipeline heißt Branch Delay Slot. Diese Slots werden dazu verwendet, um die Pipeline besser auszulasten.



Beispiel für Verwendung dieser Slots.

4 Dynamic Branch Structure (2 points)

Gegeben das folgende C Programm.

```

1 int t = 0;
2 for( int i = 0; i < 10; i++ )
3 {
4     for( int j = 0; j < 10; j++ )
5     {
6         t = t + i*j;
7     }
8 }
  
```

Nehmen Sie an, der Prozessor braucht *Dynamic Branch Prediction* und *Taken* ist genommen per Default.

- Wie viele falsche Predictions geschehen, wenn ein 1-bit Counter benutzt ist? Erläutern Sie ihre Berechnung.
- Wie viele falsche Predictions geschehen, wenn ein 2-bit Counter benutzt ist? Erläutern Sie ihre Berechnung.

(i) Für die innere Schleife liegt er 1mal falsch, weil er die Schleife 10 mal ausführt statt 9 mal, invertiert dann das bit von 1 auf 0, liegt also bei der äusseren Schlaufe zusätzlich falsch \Rightarrow Invertieren von 0 auf 1.
Also 2mal falsch pro äussere Schleife MINUS einmal richtig ganz am Schluss der äusseren Schleife. Also 17mal falsche Prediction.

(ii) Bei 2bit startet er mit 11 in die Schleifen, liegt dann pro innere Schleife 1mal falsch \Rightarrow Wechsel zu 10 äussere Schleife korrekt \Rightarrow Wechsel zu 11 usw. Beim

letzten Durchgang zusätzlich 1 mal falsch. 10 zu 00. Also 10 mal falsch.

5 Multiple-Issue Processors (3 point)

Bestimmen Sie, ob die folgenden Sätze wahr oder falsch sind. Bitte erläutern Sie kurz Ihre Antwort.

- Pipelining benutzt Instruction-Level-Parallelism, doch es ermöglicht dem Prozessor **nicht**, mehr als eine Instruktion pro Zeitpunkt auszuführen.
- Multiple-issue Prozessoren können mehr als 1 Instruktion pro Clock-Cycle ausführen.
- SIMD (single-instruction multiple-data) Prozessoren sind statische Multiple-Issue Prozessoren.

(i) Falsch. ILP ermöglicht einen Superscalar Processor mehrere seiner Komponenten gleichzeitig zu benutzen. Also ist es möglich. ILP ist ein Wert, welchen man Angeben kann. Bei 3 Instruktionen, welche 2 davon gleichzeitig ausgeführt werden können ist er $3/2$.

(ii) Wahr. Multiple-issue Prozessoren können static oder dynamic sein. Beide entscheiden auf ihre Art (static = compile time, dynamic = run time), welche Befehle parallel ausgeführt werden. Dadurch kann im Schnitt mehr als 1 Instruktion pro Clock-Cycle ausgeführt werden.

(iii) Falsch. SIMD unterstützen Vektorrechnungen, d.h. es können mehrere Berechnungen gleichzeitig ausgeführt werden, diese werden von der Hardware beim run time kontrolliert (SIMD bei Grafikkarten).

Die Entscheidungen werden deshalb nicht beim Compilieren gemacht. Deshalb Dynamic.

