SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

Systemarchitektur SS 2021

Aufgabenblatt 9

Sie können Ihre Lösungen bis **Mittwoch, dem 23.06.2021, um 10:00 Uhr** im CMS abgeben.
Geben Sie auf Ihrer Lösung Ihr Tutorium sowie die Namen und Matrikelnummern aller Gruppenmitglieder an.

Aufgabe 9.1: Pipelining (1)

Der Speedup-Faktor einer k-stufigen Pipeline konvergiert gegen k, wenn die Anzahl der betrachteten Instruktionen gegen ∞ geht. Warum gibt es trotzdem keine Pipelines mit 100 und mehr Stufen? Wodurch werden die idealisierenden Annahmen verletzt, welche der Fließbandverarbeitung zugrunde liegen?

Aufgabe 9.2: Pipelining (2)

Betrachten Sie folgende Instruktionssequenz.

```
LW r1, 5(r0)       // Lade Wert an Adresse (r0 + 5) in Register r1
ADDU r3, r1, r2    // Addiere die Inhalte von r1 und r2 nach r3
SUBU r6, r4, r5    // Subtrahiere Inhalt von r5 von r4 nach r6
LW r7, 10(r0)      // Lade Wert an Adresse (r0 + 10) in Register r7
ADDU r8, r3, r6    // Addiere die Inhalte von r3 und r6 nach r8
BEQ r7 r0 <Ziel>   // Springe nach Ziel, wenn der Inhalt von r7 0 ist
ADDU r9, r2, r4    // Addiere die Inhalte von r2 und r4 nach r9
...
Ziel:
ADDU r9, r2, r5    // Addiere die Inhalte von r2 und r5 nach r9
SUBU r5, r4, r3    // Subtrahiere Inhalt von r4 von r3 nach r5
LW r10, 5(r5)      // Lade Wert an Adresse (r5 + 5) in Register r10
```

Identifizieren Sie Daten-/Kontrollabhängigkeiten und klassifizieren Sie die Datenabhängigkeiten nach RAW, WAR, WAW. Welche dieser Abhängigkeiten verursachen auch einen Hazard auf einem Prozessor mit einer 5-stufigen Pipeline ohne *frühzeitige (Sprung-)Entscheidung*, ohne *delay slots* und ohne *forwarding* und *stalling*? Wie müsste man das Programm modifizieren, sodass es auf einem solchen Prozessor korrekt ausgeführt werden kann? Wie viel Speicherplatz benötigt das (modifizierte) Programm? Wie viele Prozessorzyklen benötigt die Ausführung?

Beantworten Sie obige Fragen nochmals, wenn nacheinander *frühzeitige (Sprung-)Entscheidung*, *delay slots*, *forwarding* und *stalling* hinzugefügt werden.

Aufgabe 9.3: Hazard Unit

In dieser Aufgabe möchten wir uns näher mit der Implementierung der Hazard Unit für die Fließbandverarbeitung beschäftigen. Um die Aufgabe einfacher zu halten, dürfen Sie annehmen, dass keine Ablaufhazards auftreten, d.h. wir interessieren uns nur für Daten-Hazards.

Im ersten Teil implementieren wir den Forwarding Mechanismus (Folie 26, Foliensatz 11). Dazu bekommen wir als Eingaben:

- Aus der Execute Phase: Die Registernummern unserer Operanden A (Rs) und B (Rt).
- Aus der Memory Phase: Die Registernummer des Zielregisters (`WriteRegM`) sowie das Registerschreibsignal (`RegWriteM`).
- Aus der Write-back Phase: Die Registernummer des Zielregisters (`WriteRegW`) sowie das Registerschreibsignal (`RegWriteW`).

Die Ausgaben `ForwardAE` und `ForwardBE` geben an ob – und wenn ja von wo – wir Daten forwarden. Überlegen Sie sich den zugehörigen Schaltkreis.

Im zweiten Teil kommt der Stalling-Mechanismus hinzu (Folie 30, Foliensatz 11). Dazu bekommen wir Zusatzeingaben:

- Aus der Decode Phase: Die Registernummern unserer Operanden A und B.
- Aus der Execute Phase: Das Speicherladesignal (`MemtoRegE`).

Das Ziel ist es, herauszufinden ob die Instruktion in der Decode-Phase einen Operanden liest, der von der Instruktion in der Execute-Phase später aus dem Speicher gelesen wird. Wenn ja, möchten wir die Stufen Fetch und Decode für einen Zyklus (`StallF`, `StallD`) anhalten und in die Stufe Execute eine Bubble einfügen (`FlushE`). Überlegen Sie sich den zugehörigen Schaltkreis.

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

System Architecture SS 2021

Assignment 9

You may submit your solutions via the CMS until **10:00 a.m. on Wednesday, June 23, 2021**.
Please state on your solutions your tutorial, and the names and matriculation numbers of all team members.

Problem 9.1: Pipelining (1)

The speedup factor of a k-stage pipeline converges to k when the number of instructions considered approaches ∞. Why are there nevertheless no pipelines with 100 and more stages? What violates the idealized assumptions that pipelining is based on?

Problem 9.2: Pipelining (2)

Consider the following instruction sequence.

```
LW r1, 5(r0)        // Load value at address (r0 + 5) into register r1
ADDU r3, r1, r2     // Add contents of r1 and r2 to r3
SUBU r6, r4, r5     // Subtract content of r5 from r4 to r6
LW r7, 10(r0)       // Load value at address (r0 + 10) into register r7
ADDU r8, r3, r6     // Add contents of r3 and r6 to r8
BEQ r7 r0 <target> // Jump to target if content of r7 is 0
ADDU r9, r2, r4     // Add the contents of r2 and r4 to r9
...
target:
ADDU r9, r2, r5     // Add the contents of r2 and r5 to r9
SUBU r5, r4, r3     // Subtract content of r4 from r3 to r5
LW r10, 5(r5)       // Load value at address (r5 + 5) into register r10
```

Identify data/control dependencies and classify the data dependencies as RAW, WAR, WAW. Which of these dependencies also cause a hazard on a processor with a 5-stage pipeline without *early branch resolution*, without *delay slots*, and without *forwarding* and *stalling*? How would one have to modify the program so that it can be executed correctly on such a processor? How much memory does the (modified) program require? How many processor cycles does the execution require?

Answer the above questions again as *early branch resolution*, *delay slots*, *forwarding* and *stalling* are added successively.

Problem 9.3: Hazard Unit

In this problem, we would like to take a closer look at the implementation of the hazard unit for pipelining. To keep the problem simpler, you may assume that no control hazards occur, i.e., we are only interested in data hazards.

In the first part, we implement the forwarding mechanism (slide 26, slide deck 14). For this we get as inputs:

- From the execute phase: The register numbers of our operands A (Rs) and B (Rt).
- From the memory phase: The register number of the destination register (`WriteRegM`) and the register write signal (`RegWriteM`).
- From write-back phase: The register number of the destination register (`WriteRegW`) and the register write signal (`RegWriteW`).

The outputs `ForwardAE` and `ForwardBE` indicate whether – and if so from where – we forward data. Develop the corresponding circuit.

In the second part, we add the stalling mechanism (slide 30, slide deck 14). For this we get additional inputs:

- From the decode phase: The register numbers of our operands A and B.
- From the execute phase: The memory load signal (`MemtoRegE`).

The goal is to find out if the instruction in the decode stage reads an operand that is later read from memory by the instruction in the execute stage. If so, we want to stop the fetch and decode stages for one cycle (`StallF`, `StallD`) and insert a bubble into the execute stage (`FlushE`). Develop the corresponding circuit.