Rechnerarchitektur
Sommersemester 25

ertis embedded real-time systems
ulm university institute

Laborprojekt – Versuch 9

Forwarding und Stall in der RISC-V

Dieser Versuch führt Sie in die Mechanismen des **Forwarding** (Bypassing) und des **Stallings** in einer 5-Stufen-Pipeline für RISC-V ein. Ziel ist es, Datenhazards bei Register- und Speicherzugriffen zu vermeiden und somit die Pipeline effizienter zu machen.

Ziele des Versuchs

- Umsetzung von Forwarding (Bypassing) zur Vermeidung von unnötigen Pipeline-Stalls.
- Behandlung von NOPs (No-Operation-Befehle) im Kontext von Forwarding.
- Einführung eines Stall-Mechanismus und Bypassing für Load-Use Hazards.
- Testen der Implementierung mittels bereitgestellter Testbenches.

Die im aktuellen Labor verwendete Umsetzung der RISC-V-Architektur orientiert sich an der klassischen 5-Stufen-Pipeline nach Hennessy und Patterson. Demnach werden Daten erst **nach Abschluss der Write-Back-Phase (WB)** im Registerfile verfügbar (read before write). Aus diesem Grund ist für die korrekte Verarbeitung nachfolgender Befehle ein **Forwarding** (auch *Bypassing* genannt) notwendig. Dabei erfolgt die Weiterleitung der Daten aus den späteren Pipeline-Stufen zurück in die EX-Phase, um Datenabhängigkeiten aufzulösen, ohne die Pipeline anzuhalten.

Entscheidung über Forwarding in der ID-Phase: In der klassischen 5-Stufen-Pipeline der RISC-V-Architektur wird die Entscheidung, ob und welche Daten aus späteren Pipeline-Stufen an die aktuelle Instruktion weitergeleitet werden müssen, bereits in der **Instruction Decode-Phase (ID)** getroffen.

Dabei werden die Quellregisteradressen (`rs1`, `rs2`) der aktuell dekodierten Instruktion mit den Zielregisteradressen (`rd`) der Instruktionen verglichen, die sich bereits in den Pipeline-Stufen **Execute (EX)**, **Memory (MEM)** und **Write-Back (WB)** befinden.

Je nachdem, ob eine Übereinstimmung vorliegt und ob die vorausgehende Instruktion das Register schreibt, werden Forwarding-Steuersignale erzeugt, die Multiplexer in der EX-Phase ansteuern, um die korrekten (aktuellsten) Operanden bereitzustellen.

Diese frühe Entscheidung in der ID-Phase ermöglicht eine effiziente Datenweitergabe und vermeidet unnötige Pipeline-Stalls.

Forwarding-Pfade:

- **EX → EX:** Das Ergebnis eines Befehls in der Execute-Phase kann direkt an den nächsten Befehl in der EX-Phase uber das Pipeline-Register ID/EX weitergereicht werden, z. B. wenn unmittelbar auf das Ergebnis der ALU zugegriffen werden muss.
- **MEM → EX:** Daten, die sich gerade in der Memory-Phase befinden, können an die EX-Phase weitergeleitet werden, etwa wenn ein vorheriger Befehl eine Speicheroperation ausführt oder die Daten bereits vor der Aktualisierung des Registerfiles benötigt werden.
- **WB → EX:** Bereits im Write-Back befindliche Daten können ebenfalls an die EX-Phase weitergegeben werden, falls die Daten bereits vor der Aktualisierung des Registerfiles benötigt werden.

Priorisierung: Im Fall, dass mehrere Forwarding-Quellen für denselben Operand zur Verfügung stehen (z. B. sowohl aus EX als auch MEM und WB), muss der **aktuellste Wert priorisiert** geladen werden, um Korrektheit zu gewährleisten.

Besonderheit bei Load-Befehlen: Bei Load-Befehlen ist zu beachten, dass der geladene Wert erst in der **MEM-Phase aus dem Speicher** verfügbar ist und somit nicht aus dem ALU-Ergebnis (EX-Phase) übernommen werden darf. Daher muss beim Forwarding aus der MEM-Phase explizit der `readdata`-Ausgang des Speichers und nicht das ALU-Ergebnis verwendet werden.

Load-Use Hazard und Pipeline-Stall: Falls ein Befehl unmittelbar nach einem Load-Befehl einen Wert aus dem geladenen Register benötigt (sog. *Load-Use Hazard*), kann dieser Wert zum Zeitpunkt der EX-Phase noch nicht vorliegen, da der Speicherzugriff erst in der MEM-Phase abgeschlossen wird. In einem solchen Fall ist ein **Pipeline-Stall** einzufügen, um die korrekte Ausführung sicherzustellen. Der Stall verzögert den Folge-Befehl so lange, bis die Daten verfügbar sind.

Aufgaben

Aufgabe 1: Forwarding

- Erweitern Sie System in der ID-Phase um eine Umsetzung des Forwarding, die für R- und I-Befehle korrekt Daten von EX-, MEM- und WB-Phase zurück zur ALU in die EX-Phase leitet. Eine mögliche Umsetzung wäre das nach Prioritäten sortierte Vergleichen der Quelladresse in der ID-Phase mit den Zieladressen in den folgenden Phasen. Dazu könnte ein Selektor für den ersten Operanden wie folgend bestimmt werden:

```
1        s_byp_rs1_sel <= "01" when (s_rs1AdrID = s_dAdrEX) else
2                         "10" when (s_rs1AdrID = s_dAdrMEM) else
3                         "11" when (s_rs1AdrID = s_dAdrWB) else
4                         "00";
```

- Da in der MEM-Phase je nach Instruktion entweder ein ALU-Ergebnis oder geladene Daten aus dem Speicher vorliegen können, müssen zwischen diesen unterschieden werden:

```
1  s_byp_rs1_MEM <= s_readdataMEM when (s_rs1AdrID = s_dAdrsMEM) and (
       s_controlWordMEM.MEM_READ = '1') else s_aluOutMEM;
```

- Erstellen Sie für die Weiterleitung jeweils eine Multiplexer-Instanz (ein 4:1-Multiplexer ist dazu ausreichend), die basierend auf `s_byp_rsX_sel` jeweils den geforderten Wert als Operand an die ALU in der EX-Phase liefert.
- Setzten Sie die Schritte und Empfehlungen jeweils für beide Operanden in Ihrem System um. Sollten SIe eine andere Lösung priorisieren können Sie dies durch entsprechend Dokumentation im Code verdeutlichen.
- Testen Sie Ihre Implementierung mit der bereitgestellten Testbench `*_tb2.vhdl`. Passen Sie gegebenenfalls den Namen Ihres Systems oder die Instanziierte Entität in der Testbench an.

Aufgabe 2: Stall und Load-Use Hazards

- Implementieren Sie einen Stall-Mechanismus, der den Pipeline-Takt anhält, wenn ein Load-Use Hazard erkannt wird. Eine mögliche Umsetzung wäre wie folgt gegeben:

```
1          s_stall <= '1' when
2              (s_controlWordEX.MEM_READ = '1') and
3              ((s_dAdressEX /= "00000") and
4              ((s_dAdressEX = s_rs2AdressID) or
5              (s_dAdressEX = s_rs1AdressID)))
6              else '0';
7
```

- Erweitern Sie das Forwarding aus der ersten Aufgabe nun so, dass die für alle Fälle korrekt funktioniert. Dazu muss nun Forwarding zusätzlich beachten das es bei einem Load-Use-Hazard nicht zu einer Weiterleitung aus der EX-Phase in die EX-Phase gibt sondern einen Stall.

- Erweitern Sie Ihr System nun so das bei einem Stall der PC- nicht inkrementiert und beim Übergang der ID/EX-Pipeline-Register nur NOPs weitergeleitet werden. Eine Möglich Umsetzung wäre die Nutzung des Flush der jeweiligen Register. Der PC (Program-Counter) hingegen kann über einen zusätzliches Enable-Port angesteuert werden und der Addierer wechselt in Abhängigkeit on es sich um einen Stall handelt oder nicht zwischen $+4$ und $+0$.
- Validieren Sie Ihr System erneut mit der Testbench `tb3`, die jetzt auch Load-Use-Hazards enthält.

Aufgabe 3: Abgabe

Laden Sie die vollständige Ordnerstruktur inklusive aller Dateien als ZIP-Archiv unter folgendem Namen auf Moodle hoch:

Name_Vorname_Versuch9.zip

Hinweise zur Fehlersuche

- Prüfen Sie, ob die Forwarding-Signale korrekt zu den ALU-Inputs geführt werden.
- Kontrollieren Sie, dass NOPs keine Forwarding-Signale auslösen.
- Überprüfen Sie die Stall-Bedingungen anhand der Pipeline-Register.
- Stellen Sie sicher, dass nach einem Stall die Pipeline korrekt fortgesetzt wird.