

Prof. L. Thiele

Technische Informatik 1 - HS 2011

Lösungsvorschläge für Übung 8

Datum: 01.12.2011

Pipelining

1 Taktrate / Latenz

In dieser Aufgabe soll der Einfluss von Pipelining auf die Taktrate des Prozessors untersucht werden. Gegeben sei der in der Vorlesung vorgestellte Datenpfad der MIPS Pipeline-Architektur, siehe Abbildung 1. Gegeben seien weiterhin die in Tabelle 1 dargestellten Rechenzeiten für die einzelnen Stufen der Pipeline.

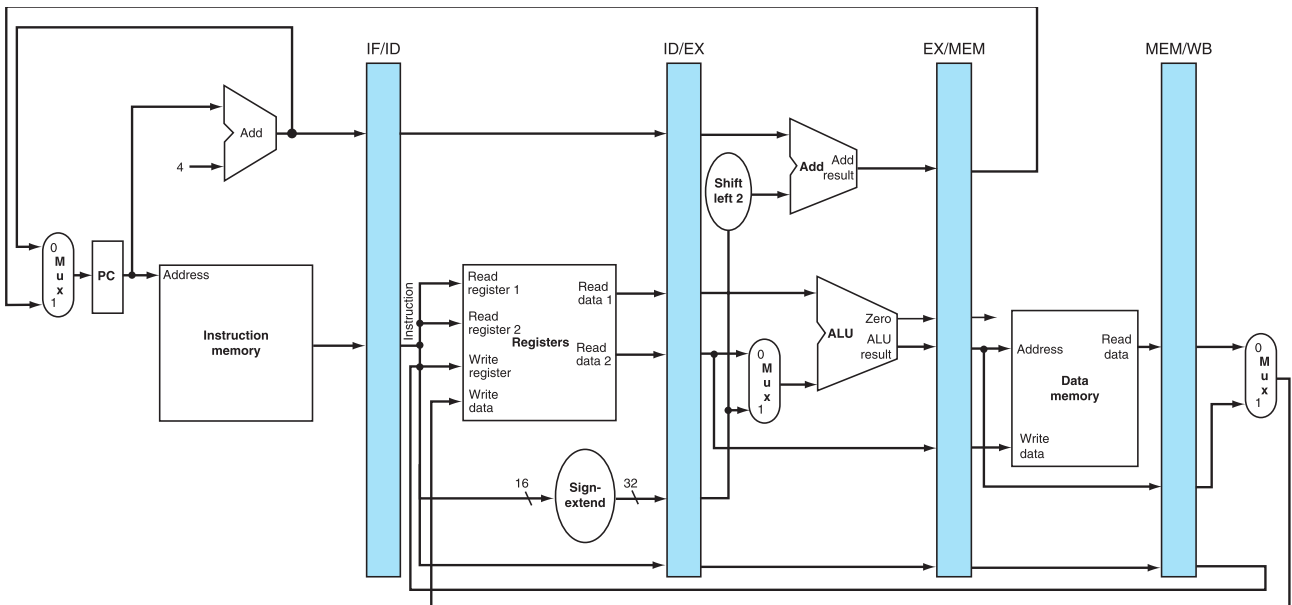


Abbildung 1: Datenpfad der MIPS Pipeline-Architektur

IF	ID	EX	MEM	WB
100 ps	120 ps	90 ps	130 ps	60 ps

Tabelle 1: Rechenzeiten der einzelnen Pipeline-Stufen

Nehmen Sie an, dass sowohl das Lesen als auch das Schreiben der Register zwischen den einzelnen Pipeline-Stufen (d.h. der Puffer IF/ID, ID/EX usw.) eine zusätzliche Verzögerung von jeweils 15 ps erzeugt.

- (a) Wie gross ist die minimal zulässige Taktperiode der Pipeline-Architektur? Wie gross ist die minimal zulässige Taktperiode einer entsprechenden Einzeltakt-Architektur ohne Pipelining?

Lösung:

- Pipeline-Architektur: $160ps$
Die Taktperiode richtet sich nach der langsamsten Stufe, in diesem Fall MEM mit $130ps$. Hinzu kommen $2 * 15ps$ für das Lesen und Schreiben der Register.
- Einzeltakt-Architektur: $500ps$

- (b) Wie gross ist die Ausführungszeit einer $1w$ Instruktion in der Pipeline-Architektur? Ignorieren Sie mögliche Hazards/Stalls. Wie gross ist die Ausführungszeit in der Einzeltakt-Architektur?

Lösung:

Pipeline-Architektur: $5 * 160ps = 800ps$

Einzeltakt-Architektur: $500ps$

Die Ausführung einer einzelnen Instruktion dauert in der Pipeline-Architektur also länger. Dafür ist der Instruktions-Durchsatz (Anzahl abgeschlossener Instruktionen pro Zeiteinheit) aber bedeutend höher im Vergleich zur Einzeltakt-Architektur.

- (c) Wie lange braucht die Pipeline-Architektur, um ein Programm mit 100 Instruktionen ohne Abhängigkeiten und Sprünge auszuführen? Wie gross ist der Speed-up im Vergleich zur Ausführung auf einer Einzeltakt-Architektur?

Lösung:

- Laufzeit Pipeline-Architektur: $(4 + 1 * 100) * 160ps = 16.64ns$
(4 Taktzyklen zu Beginn um Pipeline zu füllen, dann 1 Taktzyklus pro Instruktion)
- Laufzeit Einzeltakt-Architektur: $100 * 500ps = 50.00ns$
- Speed-up: $50/16.64 = 3.0048$

- (d) Nehmen Sie an, Sie können eine der fünf Stufen der beschriebenen Pipeline in zwei Unterstufen mit halber Rechenzeit aufteilen. Welche Stufe würden Sie wählen? Wie gross ist die neue minimale Taktperiode?

Lösung:

Die MEM Stufe ist die beste Wahl, da sie die längste Rechenzeit hat. Die neue Taktperiode beträgt $150 ps$.

- (e) Nehmen Sie an, dass alle Stufen der beschriebenen fünfstufigen Pipeline in n Unterstufen aufgeteilt werden, wobei sich die Rechenzeiten der ursprünglichen Stufen gleichmässig auf die entsprechenden Unterstufen aufteilen. Die Latenz von Puffer-Registern zwischen Pipeline-Stufen bleibt unverändert. Geben Sie die neue minimale Taktperiode für $n = 5$ und $n = 10$ an und berechnen Sie für beide Fälle die Ausführungszeit eines Programmes mit 100 Instruktionen ohne Abhängigkeiten und Sprünge. Geben Sie zudem für beide Fälle die Anzahl benötigter Puffer-Register an. Welchen Trend stellen Sie für eine steigende Anzahl der Pipeline-Stufen fest?

Lösung:

Taktperiode: $(130/n + 2 * 15)$

(Das Aufteilen in Unterstufen erfordert zusätzliche Register. Jedes Register hat eine Puffer-Latenz von $15 ps$ für das Lesen bzw. Schreiben)

Laufzeit: $(130/n + 2 * 15) * (5 * n + 99)$

$(5 * n - 1)$ Taktzyklen zum Auffüllen der Pipeline, dann 1 Taktzyklus pro Instruktion)

$n = 5$: Minimale Taktperiode = $56 ps$, Laufzeit für 100 Instr. = $6.944 ns$, Anzahl Register = 24

$n = 10$: Minimale Taktperiode = $43 ps$, Laufzeit für 100 Instr. = $6.407 ns$, Anzahl Register = 49

Trend: Für eine steigende Anzahl von Pipeline-Stufen nimmt der Speed-up aufgrund der konstanten Puffer-Latenzen immer langsamer zu, während die Anzahl der benötigten Puffer-Register schnell steigt und somit auch die Fläche und der Energieverbrauch des Prozessors.

2 Daten-Hazards und Forwarding

In dieser Aufgabe betrachten wir Anpassungen des Forwarding-Mechanismus und der Pipeline-Stufen des MIPS32 Prozessors. Nehmen Sie an, dass der Prozessor keine Zusatzlogik zur Vorzeitigen Sprungauflösung hat, d.h. Sprungentscheidungen sind nach der MEM-Stufe der jeweiligen Sprung-Instruktionen bekannt. Gehen Sie zudem davon aus, dass keine Branch Delay Slots eingesetzt werden.

Gegeben sei folgendes MIPS Assembler-Programm. Das Programm implementiert eine Funktion, die das Maximum in einem Array von positiven Integer-Werten ermittelt. Die Startadresse des Arrays wird im Register \$a0 übergeben, die Länge des Arrays im Register \$a1. Das Ergebnis der Funktion wird im Register \$v0 zurückgegeben. Das Register \$v0 sei vor dem Funktionsaufruf mit dem Wert 0 initialisiert.

```

loop:  lw    $t0,0($a0)
        slt  $t1,$v0,$t0
        beq  $t1,$zero,l1
        addi $v0,$t0,0
l1:    addi $a0,$a0,4
        addi $a1,$a1,-1
        bne  $a1,$zero,loop
        jr   $ra

```

Zur Vereinfachung nehmen wir für diese Aufgabe an, dass alle Sprünge im Programm perfekt vorhergesagt werden, d.h. dass es bei der Ausführung nie Ablauf-Hazards gibt.

Tabelle 2 zeigt die zeitliche Belegung der Pipeline (Pipelining Diagramm) für eine Schleifenausführung des obigen Programms auf der in der Vorlesung vorgestellten MIPS Prozessorarchitektur mit fünfstufiger Pipeline und vollem Forwarding. Dabei wird angenommen, dass das Programm in der dritten Instruktion (beq) nicht springt. Im Diagramm ist ersichtlich, dass die Pipeline trotz des verwendeten Forwarding-Mechanismus im vierten Taktzyklus gestallt werden muss. Dies ist aufgrund der Datenabhängigkeit der zweiten Instruktion (slt) von der ersten Instruktion (lw) nötig.

- (a) Betrachten Sie nun eine veränderte Version der MIPS Pipeline-Architektur in der ausschliesslich ALU-ALU Forwarding unterstützt wird. Das bedeutet, dass nur die Ergebnisse der EX Stufe an die EX Stufe der folgenden Instruktion weitergeleitet werden können, nicht aber die Ergebnisse der MEM Stufe. Um wie viele Taktzyklen verlängert sich die Schleifenausführung im Vergleich zur Architektur mit vollem Forwarding? Nehmen Sie wiederum an, dass in der dritten Instruktion (beq) nicht gesprungen wird.

Lösung:

Mit ausschliesslichem ALU-ALU Forwarding dauert die Programmausführung einen Taktzyklus länger. Der Grund ist ein zusätzlicher Stall für die Instruktion slt, da das MEM-ALU Forwarding nicht mehr möglich ist. Siehe Tabelle 3 für das vollständige Pipelining-Diagramm unter ALU-ALU Forwarding.

- (b) Für die zwei Arten der Pipeline-Architektur (volles Forwarding, nur ALU-ALU Forwarding) seien die folgenden Taktperioden gegeben.

Volles Forwarding	ALU-ALU Forwarding
200 ps	160 ps

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Loop: lw \$t0,0(\$a0)	IF	ID	EX	MEM↓	WB										
slt \$t1,\$v0,\$t0		IF	ID	-	↑EX↓	MEM	WB								
beq \$t1,\$zero,l1			IF	-	ID	↑EX	MEM	WB							
addi \$v0,\$t0,0					IF	ID	EX	MEM	WB						
l1: addi \$a0,\$a0,4						IF	ID	EX	MEM	WB					
addi \$a1,\$a1,-1							IF	ID	EX↓	MEM	WB				
bne \$a1,\$zero,loop								IF	ID	↑EX	MEM	WB			

Tabelle 2: Pipelining Diagramm für Ausführung mit vollem Forwarding

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
loop: lw \$t0,0(\$a0)	IF	ID	EX	MEM	WB										
slt \$t1,\$v0,\$t0		IF	-	-	ID	EX↓	MEM	WB							
beq \$t1,\$zero,l1					IF	ID	↑EX	MEM	WB						
addi \$v0,\$t0,0						IF	ID	EX	MEM	WB					
l1: addi \$a0,\$a0,4							IF	ID	EX	MEM	WB				
addi \$a1,\$a1,-1								IF	ID	EX↓	MEM	WB			
bne \$a1,\$zero,loop									IF	ID	↑EX	MEM	WB		

Tabelle 3: Pipelining Diagramm für Ausführung mit ALU-ALU Forwarding

Wie lange brauchen die zwei Architekturen, um das obige Programm für ein Array der Grösse 100 ($\$a1 = 100$) auszuführen? Nehmen Sie dabei an, dass in der dritten Instruktion der Schleife (beq) nie gesprungen wird. Für welche der zwei Architekturen entscheiden Sie sich wenn Sie die Programmlaufzeit minimieren wollen?

Lösung:

Taktzyklen mit vollem Forwarding: $4 + 100 * (7 + 1) + 1 = 805$

(4 Taktzyklen zum Füllen der Pipeline. Dann pro Schleifendurchlauf 7 ausgeführte Instruktionen und ein Stall, d.h. 8 Taktzyklen. Am Ende noch ein Taktzyklus für die Instruktion jr.) Taktzyklen

mit ALU-ALU Forwarding: $4 + 100 * (7 + 2) + 1 = 905$

Laufzeit mit vollem Forwarding: $805 * 200ps = 161ns$

Laufzeit mit ALU-ALU Forwarding: $905 * 160ps = 144.8ns$

Die kürzeste Laufzeit erreicht die Architektur mit ALU-ALU Forwarding.

- (c) Gegeben sei die Pipeline-Architektur mit ausschliesslichem ALU-ALU Forwarding wie oben beschrieben. Ordnen Sie die Instruktionen des Programms so um, dass während der Ausführung keine Stalls aufgrund von Daten-Hazards nötig sind. Die Semantik des Programms muss dabei unverändert bleiben.

Lösung:

```

loop:  lw    $t0,0($a0)
        addi $a0,$a0,4
        addi $a1,$a1,-1
        slt  $t1,$v0,$t0
        beq  $t1,$zero,11
        addi $v0,$t0,0
11:    bne  $a1,$zero,loop

```

- (d) Gegeben sei der in der Vorlesung besprochene Datenpfad mit vollem Forwarding zur Minderung von Daten-Hazards, siehe Abbildung 2. Passen Sie den Datenpfad so an, dass nur ALU-ALU Forwarding unterstützt wird.

- Streichen Sie insbesondere alle Daten- und Steuerungsleitungen durch, die nicht benötigt werden und passen Sie die Forwarding Unit sowie die Multiplexer entsprechend an. Beachten Sie, dass die in den Zwischenregistern gepufferten Steuerungssignale nicht nur von der Forwarding Unit, sondern auch zur Ansteuerung der Komponenten in der jeweiligen Stufe (ALU, Data Memory, Multiplexer usw.) verwendet werden.¹
- Beschreiben Sie die Forwarding-Funktionen zur Ansteuerung der angepassten Multiplexer (Als Vorlage dienen die Forwarding-Funktionen auf den Seiten 9-32 und 9-33 der Vorlesungsunterlagen).

Lösung:

Abbildung 3 zeigt welche Leitungen nicht mehr benötigt werden. Das Ergebnis der MEM Stufe muss nicht mehr an die ALU zurückgeführt werden, siehe Leitung (1) in Abbildung 3. Weiterhin muss die Forwarding Unit nicht mehr mit dem MEM/WB Puffer verbunden sein. Insbesondere muss die Forwarding Unit weder wissen ob die Instruktion im MEM/WB Puffer ein Register schreibt (vorher über Leitung (2) kommuniziert), noch welches Register geschrieben wird (vorher über Leitung (3) kommuniziert).

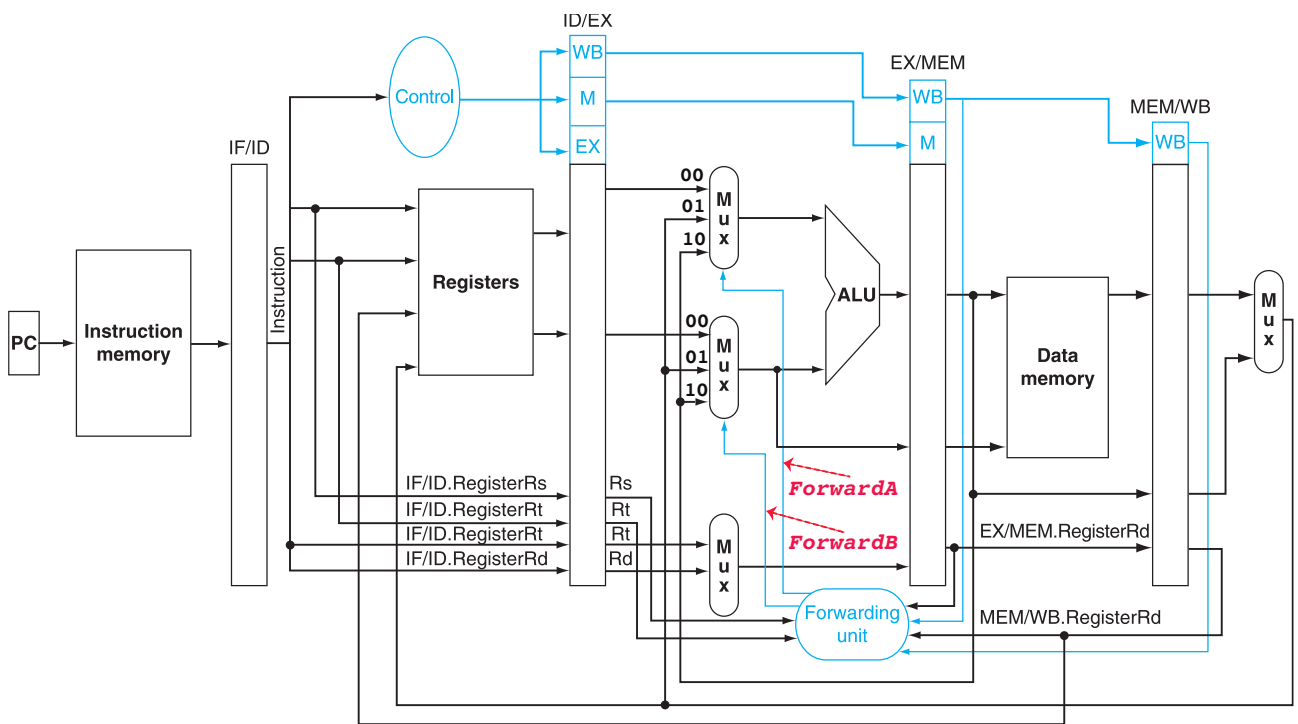


Abbildung 2: Daten-/Kontrollpfad für volles Forwarding

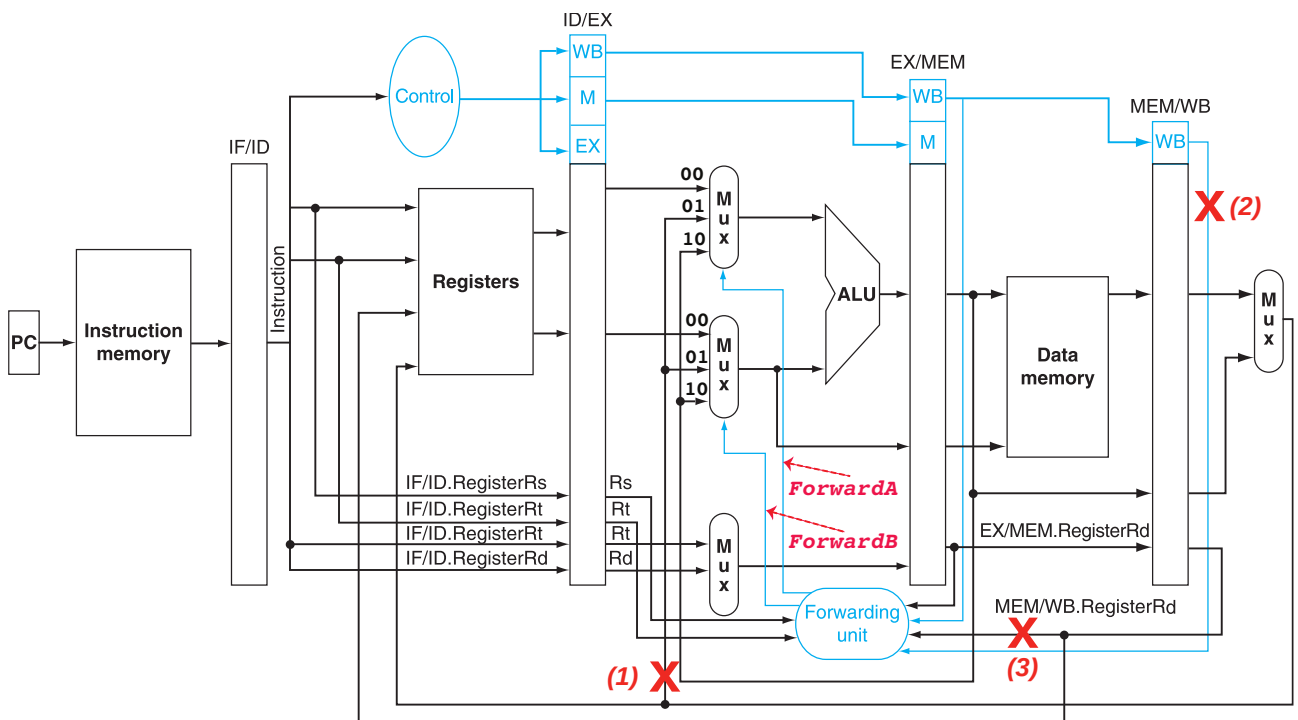


Abbildung 3: Nicht mehr benötigte Daten und Kontrollleitungen

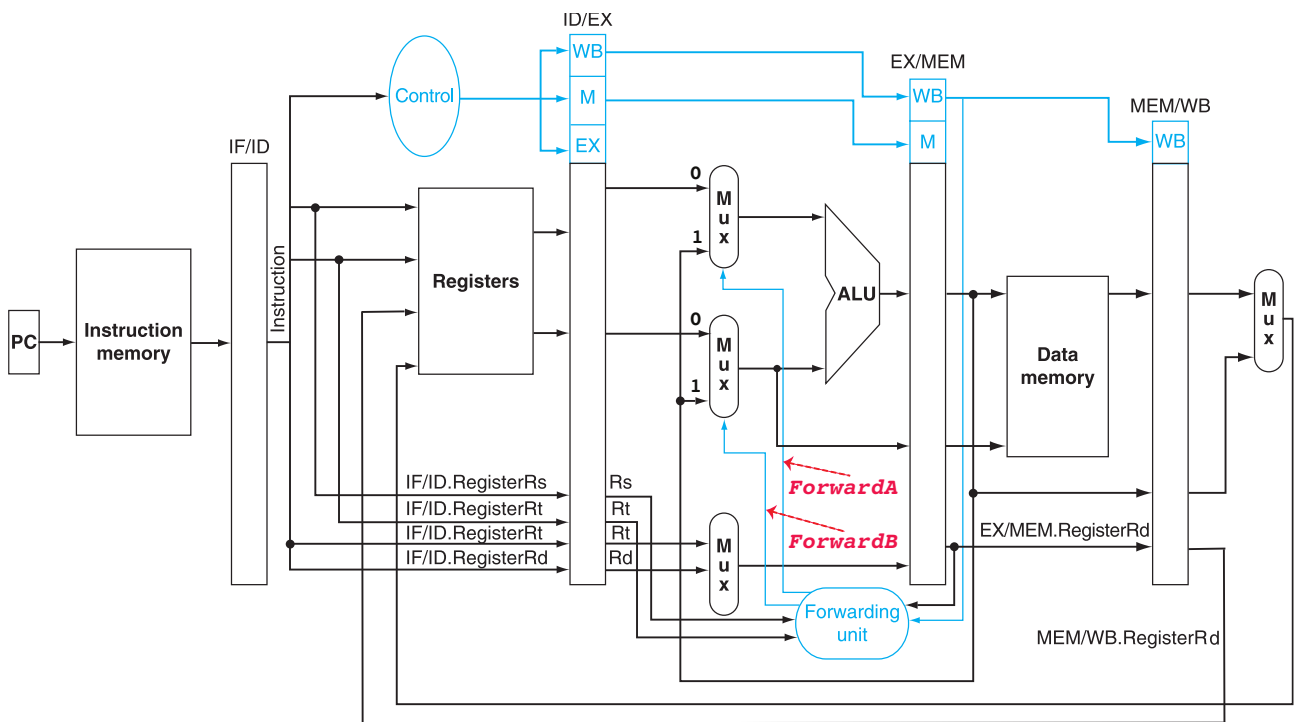


Abbildung 4: Daten-/Kontrollpfad für ALU-ALU Forwarding

Die zwei Multiplexer zur Auswahl der ALU-Operanden haben jetzt nur noch zwei Eingänge und können folglich mit jeweils einem Bit angesteuert werden. Das Ergebnis der Anpassung ist in Abbildung 4 dargestellt.

Zur Ansteuerung der Multiplexer werden folgende Forwarding-Funktionen verwendet:

```
if ( EX/MEM.RegWrite = 1 and
      EX/MEM.RegisterRd != 0 and
      EX/MEM.RegisterRd = ID/EX.RegisterRs)
{ForwardA = '1';}
```

```
if ( EX/MEM.RegWrite = 1 and
      EX/MEM.RegisterRd != 0 and
      EX/MEM.RegisterRd = ID/EX.RegisterRt)
{ForwardB = '1';}
```

Diese Funktionen entsprechen denen von Seite 9-32 der Vorlesungsunterlagen.

- (e) Nehmen Sie an, dass im MIPS Instruktionssatz die load/store Instruktionen so verändert werden, dass die Zugriffadresse im Speicher ausschliesslich durch ein Register angegeben wird (ohne Offset). In diesem Fall benötigen diese Instruktionen die ALU nicht mehr. Das bedeutet, dass die EX Stufe und die MEM Stufe überlappt werden können, da keine Instruktion beide Stufen verwendet. Als Ergebnis erhält man eine neue vierstufige Pipeline-Architektur: IF, ID, EX-MEM, WB.

Das obige Programm (in der ursprünglichen Instruktionenreihenfolge) soll nun auf der neuen Architektur ausgeführt werden.

- Passen Sie zunächst den Assemblercode des Programms auf den neuen Instruktionssatz an (Es reicht wenn Sie die Unterschiede angeben).

¹Zur Vereinfachung ist in Abbildung 2 die Ansteuerung der einzelnen Komponenten nicht gezeigt.

- Nehmen Sie an, dass die neue Pipelining-Architektur Forwarding von der EX-MEM Stufe zur EX-MEM Stufe unterstützt. Gibt es bei der Programmausführung noch Stalls aufgrund von Daten-Hazards? Begründen Sie Ihre Antwort.
- Wie lange dauert nun die Programmausführung für ein Array der Grösse 100 ($\$a1 = 100$)? Nehmen Sie dabei an, dass die Taktperiode weiterhin 160 ps beträgt und dass in der dritten Instruktion der Schleife (beq) nie gesprungen wird.
- Wie gross ist der Speed-up für die Programmausführung im Vergleich zur fünfstufigen Pipeline mit ALU-ALU Forwarding?

Lösung:

Es ändert sich nur die erste Instruktion: `loop: lw $t0,$a0`

Es gibt keine Stalls mehr aufgrund von Daten-Hazards. Unabhängig davon, ob man einen Wert mit der ALU berechnet, oder aus dem Speicher liest, kann der Wert immer mit Hilfe von Forwarding von der EX-MEM Stufe zur EX-MEM Stufe der folgenden Instruktion weitergeleitet werden.

Taktzyklen: $3 + 100 * 7 + 1 = 704$

Laufzeit: $704 * 160ps = 112.64ns$

Speed-up: $144.64/112.64 = 1.284$

3 Sprung-Vorhersage

In dieser Aufgabe betrachten wir statische und dynamische Vorhersage von Sprüngen. Es sei das Programm aus Aufgabe 2 mit der ursprünglichen Reihenfolge von Instruktionen gegeben. Weiterhin sei die in der Vorlesung besprochene fünfstufige Pipeline-Architektur mit vollem Forwarding und ohne Branch Delay Slot gegeben. Betrachten Sie die Ausführung des Programms für folgendes Array mit 10 Werten: [5, 10, 8, 15, 7, 4, 22, 26, 3, 30].

3.1 Statische Sprung-Vorhersage

- (a) Nehmen Sie an, die Sprung-Entscheidung sei nach der MEM Stufe der Sprung-Instruktionen bekannt. Welche statische Sprung-Vorhersage ist im gegebenen Programm besser für den Sprung in der Instruktion `beq` (Sprung nach `l1`)? Welche statische Sprung-Vorhersage ist besser für den Sprung in der Instruktion `bne` (Sprung nach `loop`)? Begründen Sie Ihre Antworten.

Lösung:

Beste statische Sprung-Vorhersagen

Sprung `beq` nach `l1`: NOT TAKEN (weil Sprung wird 4 von 10 Mal ausgeführt)

Sprung `bne` nach `loop`: TAKEN (weil Sprung wird 9 von 10 Mal ausgeführt)

- (b) Wie lange dauert die Programmausführung für das obige Array und den gewählten statischen Sprung-Vorhersagen? Nehmen Sie eine Taktperiode von 200 ps an.

Lösung:

Bei jeder falschen Sprungvorhersage gibt es 3 Stalls. Die Laufzeit des Programms kann wie folgt berechnet werden.

Taktzyklen: $4 + 6 * (7 + 1) + 4 * (6 + 1 + 3) + 3 + 1 = 96$

(4 Taktzyklen zum Füllen der Pipeline. 6 Schleifenausführungen mit jeweils 7 Instruktionen und 1 Stall wegen Data-Hazard, siehe Aufgabe 2. 4 Schleifenausführungen mit 6 Instruktionen, 1 Stall wegen Data-Hazard und 3 Stalls wegen falscher Sprung-Vorhersage für `beq`. Beim Verlassen der

Schleife eine falsche Sprung-Vorhersage für `bne`, d.h. 3 Stalls. Zuletzt noch ein Taktzyklus für die Instruktion `jr`.)

Laufzeit: $96 * 200ps = 19.2ns$

- (c) Nehmen Sie nun an, die Sprung-Entscheidung sei bereits nach der ID Stufe bekannt (siehe Seite 9-48 der Vorlesungsunterlagen). Durch die zusätzliche Logik für die Sprung-Auflösung in der ID Stufe steige die Taktperiode der Architektur auf 230 ps. Wie lange dauert die Programmausführung für das obige Array und den gewählten statischen Sprung-Vorhersagen? Lohnt sich die zusätzliche Logik zur frühzeitigen Sprung-Auflösung, um die Programmausführung zu beschleunigen?

Lösung:

Bei jeder falschen Sprungvorhersage gibt es jetzt nur noch einen Stall. Die Laufzeit des Programms ergibt sich wie folgt:

Taktzyklen: $4 + 6 * (7 + 1) + 4 * (6 + 1 + 1) + 1 + 1 = 86$

Laufzeit: $86 * 230ps = 19.78ns$

Die zusätzliche Logik zur Vorzeitigen Sprung-Entscheidung lohnt sich in diesem Fall nicht.

3.2 Dynamische Sprung-Vorhersage

Für den Sprung `beq` nach `l1` soll nun ein 2-Bit-Prädiktor P1 zur dynamischen Sprung-Vorhersage verwendet werden. Ebenso soll für den Sprung `bne` nach `loop` ein 2-Bit-Prädiktor P2 zum Einsatz kommen. Das Verhalten von 2-Bit-Prädiktoren wurde in der Vorlesung besprochen und ist in Abbildung 5 in Form eines endlichen Automaten beschrieben. Nehmen Sie an, dass sich zu Beginn der Programmausführung P1 im Zustand 'strongly taken' befindet, während sich P2 im Zustand 'strongly not taken' befindet.

Simulieren Sie die dynamische Sprung-Vorhersage für die oben beschriebene Programmausführung. Vervollständigen Sie dazu die Tabellen 4 und 5.²

Für welchen Sprung funktioniert die dynamische Prädiktion besser? Wann ist der Einsatz eines Sprung-Prädiktors empfehlenswert? Begründen Sie Ihre Antwort.

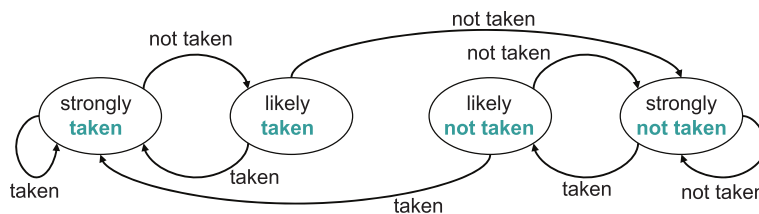


Abbildung 5: 2-Bit-Prädiktor für dynamische Sprung-Vorhersage

Iteration der Schleife	1	2	3	4	5	6	7	8	9	10
Zustand P1 vor <code>beq</code> Instruktion	ST	LT								
Sprung-Vorhersage für <code>beq</code>	T									
Sprung-Entscheidung für <code>beq</code>	NT									
Vorhersage korrekt	nein									

Tabelle 4: Dynamische Sprung-Vorhersage für `beq`

Lösung:

Siehe Tabellen 6 und 7.

²Verwenden Sie folgende Abkürzungen. Sprung-Vorhersagen und -Entscheidungen: T = 'taken', NT = 'not taken'. Zustände Prädiktoren: SNT = 'strongly not taken', LNT = 'likely not taken', ST = 'strongly taken', LT = 'likely taken'.

Iteration der Schleife	1	2	3	4	5	6	7	8	9	10
Zustand P2 vor bne Instruktion	SNT	LNT								
Sprung-Vorhersage für bne	NT									
Sprung-Entscheidung für bne	T									
Vorhersage korrekt	nein									

Tabelle 5: Dynamische Sprung-Vorhersage für bne

Iteration der Schleife	1	2	3	4	5	6	7	8	9	10
Zustand P1 vor beq Instruktion	ST	LT	SNT	LNT	SNT	LNT	ST	LT	SNT	LNT
Sprung-Vorhersage für beq	T	T	NT	NT	NT	NT	T	T	NT	NT
Sprung-Entscheidung für beq	NT	NT	T	NT	T	T	NT	NT	T	NT
Vorhersage korrekt	nein	nein	nein	ja	nein	nein	nein	nein	nein	ja

Tabelle 6: Dynamische Sprung-Vorhersage für beq

Iteration der Schleife	1	2	3	4	5	6	7	8	9	10
Zustand P2 vor bne Instruktion	SNT	LNT	ST	ST	ST	ST	ST	ST	ST	ST
Sprung-Vorhersage für bne	NT	NT	T	T	T	T	T	T	T	T
Sprung-Entscheidung für bne	T	T	T	T	T	T	T	T	T	NT
Vorhersage korrekt	nein	nein	ja	ja	ja	ja	ja	ja	ja	nein

Tabelle 7: Dynamische Sprung-Vorhersage für bne

Die dynamische Prädiktion funktioniert für den Sprung bne wesentlich besser. Der Grund dafür ist die für eine Schleife typische hohe Korrelation von aufeinander folgenden Sprung-Entscheidungen. Insbesondere lernt der Prädiktor das korrekte Verhalten nach wenigen Schleifeniterationen und liegt dann mit der Vorhersage bis zum Verlassen der Schleife richtig. Im Gegensatz dazu funktioniert die dynamische Prädiktion für den Sprung beq schlecht, da die aufeinander folgenden Sprung-Entscheidungen keine Korrelation aufweisen.

Im Allgemeinen ist der Einsatz von dynamischer Sprung-Vorhersage aufgrund der vergangenen Sprung-Entscheidungen nur dann hilfreich, wenn es eine gewisse Regelmässigkeit in der Abfolge von Sprung-Entscheidungen gibt.