

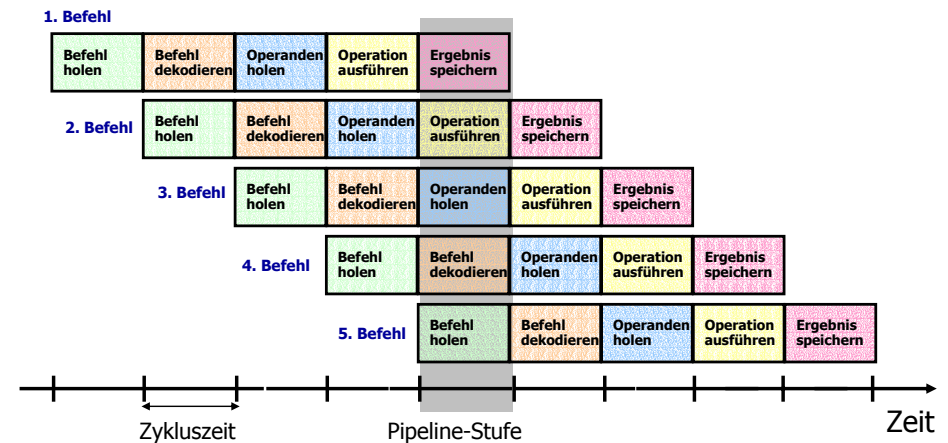
Pipelining „Fließband-Verarbeitung“

„Pipelines beschleunigen die Ausführungsgeschwindigkeit eines Rechners in gleicher Weise wie Henry Ford die Autoproduktion mit der Einführung des Fließbandes revolutionierte.“

(Peter Wayner 1992)

Aufgabe 1

Gegeben sei eine 5-stufige Pipeline:



Aufgabe 1

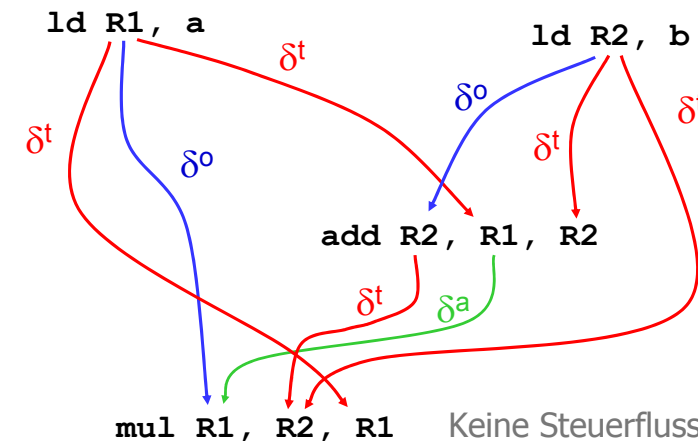
Betrachten Sie das folgende sequentielle Programmstück, in dem die Konstanten **a** und **b** Speicheradressen darstellen:

```
m1: ld    R1, a      ; R1 := [a]
m2: ld    R2, b      ; R2 := [b]
m3: add   R2, R1, R2  ; R2 := R1+R2
m4: mul   R1, R2, R1  ; R1 := R1*R2
```

In der Pipeline-Struktur ist ein Schreibvorgang in das entsprechende Zielregister erst am Ende der Ergebnis-Speichern-Phase abgeschlossen.

Aufgabe 1.1

1. Bestimmen Sie alle Daten- und Steuerflussabhängigkeiten in diesem Programmstück



Aufgabe 1.2

2. Wieviele Pipelinekonflikte treten auf?

ld R1, a



ld R2, b



add R2, R1, R2



mul R1, R2, R1



Aufgabe 1

3. Die auftretenden Pipelinekonflikte werden von der Hardware nicht erkannt und müssen vom Compiler durch Einfügen von NOOP-Befehlen behandelt werden.

Ergänzen Sie das Programmstück so, dass auch die Pipelinekonflikte berücksichtigt werden

Aufgabe 1.3

ld R1, a

ld R2, b

noop

noop

add R2, R1, R2

noop

noop

mul R1, R2, R1

ld R1, a



ld R2, b



add R2, R1, R2



mul R1, R2, R1



Aufgabe 1

4. Welche der NOOP-Befehle sind noch notwendig, falls die auftretenden Pipelinekonflikte von der Hardware erkannt werden und durch *Load Forwarding* und *Result Forwarding* behandelt werden?

Aufgabe 1.4

ld R1, a

Befehl holen	Befehl dekodieren	Operanden holen	Operation ausführen	Ergebnis speichern
--------------	-------------------	-----------------	---------------------	--------------------

ld R2, b

Befehl holen	Befehl dekodieren	Operanden holen	Operation ausführen	Ergebnis speichern
--------------	-------------------	-----------------	---------------------	--------------------

add R2, R1, R2

Befehl holen	Befehl dekodieren	Operanden holen	Operation ausführen	Ergebnis speichern
--------------	-------------------	-----------------	---------------------	--------------------

mul R1, R2, R1

Befehl holen	Befehl dekodieren	Operanden holen	Operation ausführen	Ergebnis speichern
--------------	-------------------	-----------------	---------------------	--------------------

Aufgabe 1.4

Keine !!!

ld R1, a

Befehl holen	Befehl dekodieren	Operanden holen	Operation ausführen	Ergebnis speichern
--------------	-------------------	-----------------	---------------------	--------------------

ld R2, b

Befehl holen	Befehl dekodieren	Operanden holen	Operation ausführen	Ergebnis speichern
--------------	-------------------	-----------------	---------------------	--------------------

add R2, R1, R2

Befehl holen	Befehl dekodieren	Operanden holen	Operation ausführen	Ergebnis speichern
--------------	-------------------	-----------------	---------------------	--------------------

mul R1, R2, R1

Befehl holen	Befehl dekodieren	Operanden holen	Operation ausführen	Ergebnis speichern
--------------	-------------------	-----------------	---------------------	--------------------

Aufgabe 2

Bei Pipeline-Prozessor ist die Kontrolle der Befehlspipeline vollständig dem Compiler übertragen. Die einzelnen Befehle werden in einer fünfstufigen Befehlspipeline (Befehl holen, Befehl dekodieren, Operanden holen, Operation ausführen und Ergebnis speichern) verarbeitet. **Erst am Ende der Ergebnis-speichern-Phase ist ein Schreibvorgang in das entsprechende Zielregister abgeschlossen.**

Betrachten Sie das folgende sequentielle Programmstück:

```
m1:  add  R1, R1, R1  ; R1 := R1+R1
m2:  add  R2, R1, R1  ; R2 := R1+R1
m3:  add  R2, R1, R2  ; R2 := R1+R2
```

Aufgabe 2.1

```
add  R1, R1, R1
add  R2, R1, R1
add  R2, R1, R2
```

1. Welchen Wert enthält das Register R2 nach Abarbeitung dieser Befehlsfolge, wenn R1 mit 4 und R2 mit 7 initialisiert ist?

add R1, R1, R1

F	D: add	R: 4, 4	E: 4 + 4	W: 8 → R1
---	--------	---------	----------	-----------

add R2, R1, R1

F	D: add	R: 4, 4	E: 4 + 4	W: 8 → R2
---	--------	---------	----------	-----------

add R2, R1, R2

F	D: add	R: 4, 7	E: 4 + 7	W: 11 → R2
---	--------	---------	----------	------------

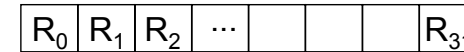
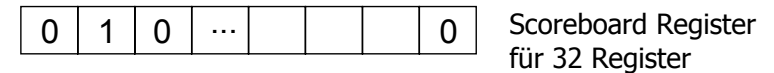
R2 = 11

Aufgabe 2.2

2. Wieviele Takte benötigt das Programm bis zur vollständigen Leerung *der Befehlspipeline*, wenn zusätzlich Scoreboarding und Result Forwarding eingesetzt werden?

Scoreboarding Technik

- Dient zum Erkennen von Datenabhängigkeiten.
- Jedem allgemeinen Register (und Fließkommaregister) wird ein Bit in einem **Scoreboard Register** zugeordnet.



- Ein Bit im Scoreboard Register wird nach der Decodierung gesetzt, wenn das entsprechende Register als Ziel einer Operation dient.
- Das Bit bleibt gesetzt, bis das Ergebnis in das Register geschrieben wird; danach wird es zurückgesetzt.

Scoreboarding Technik

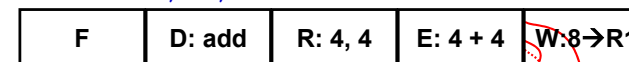
- Durch Prüfung der Scoreboard-Bits kann für jeden Befehl ein durch Datenabhängigkeit drohender Pipelinekonflikt erkannt werden.
- Behandlung des Konflikts durch die Verzögerung der Ausführung des Befehls, dessen Operandenregister ein gesetztes Scoreboard-Bit besitzt, bis zum Rücksetzen des Bits.

Scoreboarding ist eine Technik, mit der Datenabhängigkeiten durch die Hardware erkannt und im einfachsten Fall durch Verzögerungen behandelt werden können.

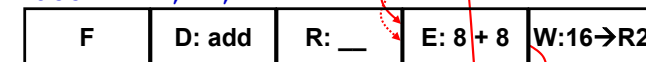
Aufgabe 2.2

2. Wieviele Takte benötigt das Programm bis zur vollständigen Leerung der Befehlspipeline, wenn zusätzlich Scoreboarding und Result Forwarding eingesetzt werden?

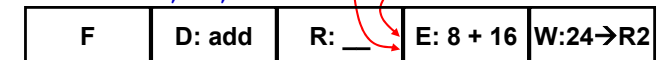
add R1, R1, R1



add R2, R1, R1



add R2, R1, R2



7 Takte

Aufgabe 3

Gegeben sei eine Pipeline-Struktur, bei der die absoluten Sprungbefehle (Assemblerbefehl: **ba**) mit einem Verzögerungszeitschlitz (*Delay-Slot*) ausgeführt werden.

Das folgende kleine Programmstück besteht aus fünf Assemblerbefehlen, die mit den Labels **m1**, **m2**, ..., **m5** markiert sind.

```
m1:    add    R2, R2, R2
m2:    ba     m1
m3:    ba     m4
m4:    add    R1, R2, R2
m5:    add    R1, R1, R1
```

Aufgabe 3

Geben Sie die Ausführungsreihenfolge der Befehle bei Ausführung des Programmstücks an. Die Befehle sollen durch die zugehörigen Labels abgekürzt werden, d.h. es ist eine Folge der Form **m_i**, **m_j**, **m_k**, ... anzugeben

```
m1:    add    R2, R2, R2
m2:    ba     m1
m3:    ba     m4
m4:    add    R1, R2, R2
m5:    add    R1, R1, R1
```

Ausführungsreihenfolge: **m1** **m2** **m3** **m1** **m4** **m5**

Aufgabe 3

Die ersten beiden Befehle werden sequentiell ausgeführt.

→ **m1** – **m2**. Da absolute Sprünge einen Delay-Slot besitzen, werden sie um einen Takt verzögert, d. h. der Befehl hinter dem Sprungbefehl wird auch noch ausgeführt.

Deshalb ist der Sprungbefehl **m3** bereits in der Pipeline, bevor der Sprungbefehl **m2** ausgeführt wird.

→ **m2** – **m3**.

Sobald der Sprungbefehl **m2** ausgeführt ist, wird der erste Befehl an der Zieladresse, also **m1**, in die Pipeline geladen.

→ **m3** – **m1**.

Dann wird der Sprungbefehl **m3** ausgeführt, so dass als nächster Befehl **m4** in die Pipeline geladen wird.

→ **m1** – **m4**. Da jetzt kein Sprungbefehl mehr ausgeführt wird, folgt der nächste Befehl nach **m4**, also **m5**.

→ **m4** – **m5**.