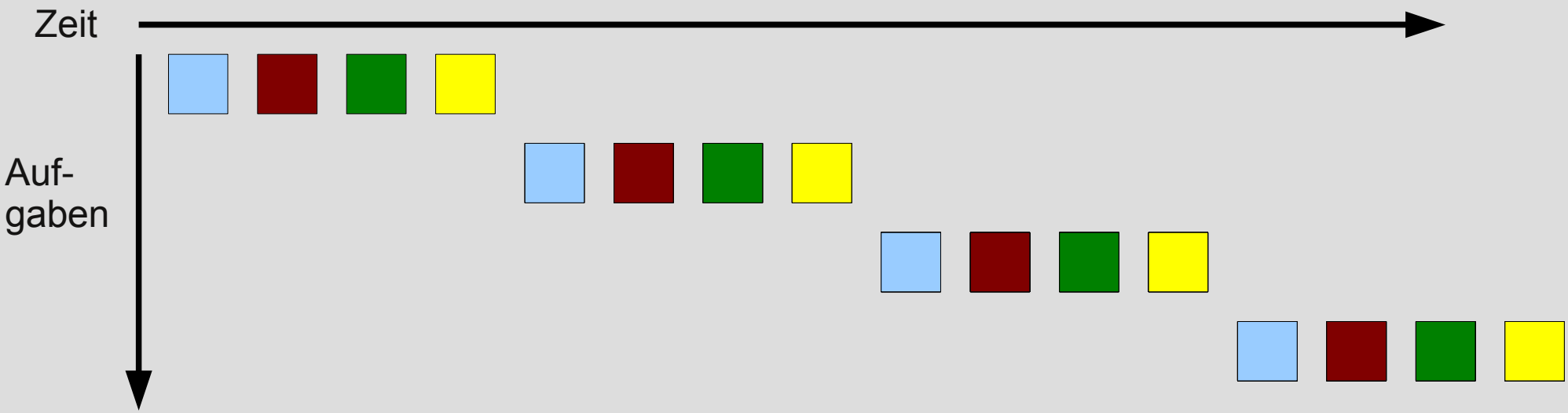
Parallele Rechnerarchitekturen

- Bisher behandelte:
- Vorlesung 7 (theoretische Grundkonzepte)
- Nun konkrete Ausprägungen

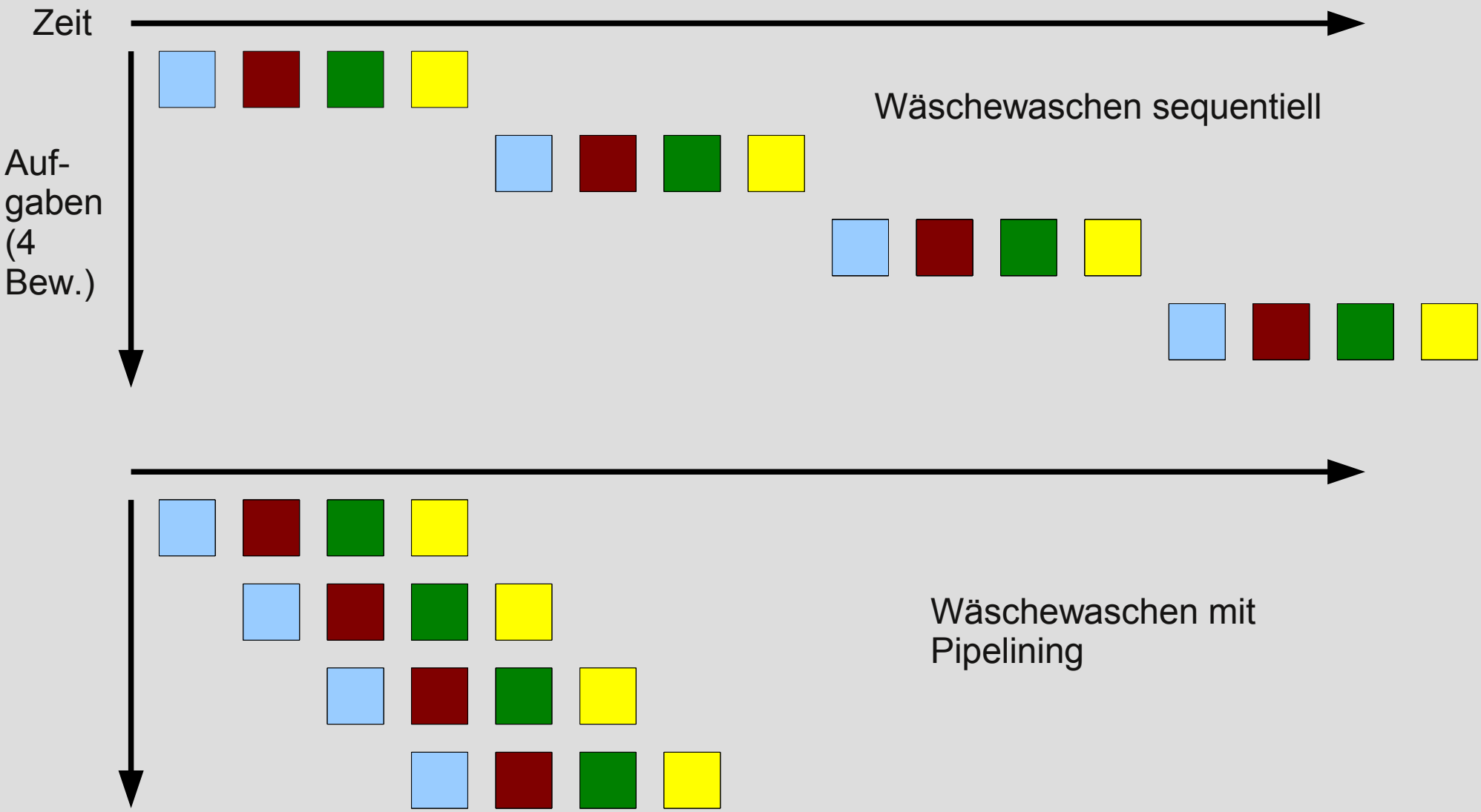
Pipelining

- Beispiel aus dem realen Leben (;-))
- Wäschewaschen in WG
  - Füllen der Waschmaschine mit schmutziger Wäsche
  - Umfüllen der gewaschenen Wäsche in den Trockner
  - nach dem Trocknen Wäsche zusammenlegen
  - Nach dem Zusammenlegen einen Mitbewohner bitte, Wäsche in den Schrank zu legen

Pipelining (2)

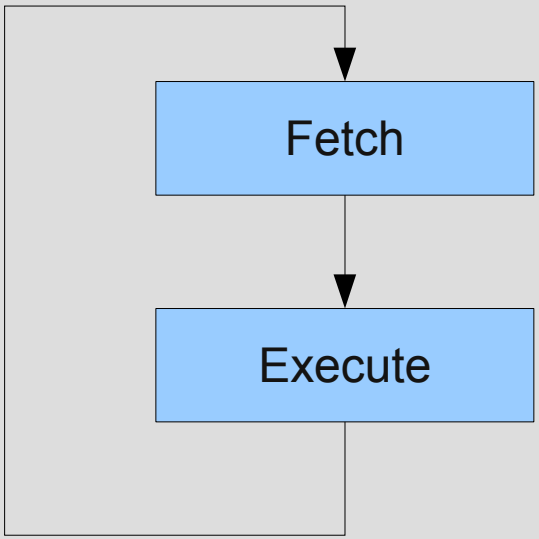


Pipelining (2)



Pipelining (3)

Zentrale Steuerschleife eines von-Neumann-Rechners:

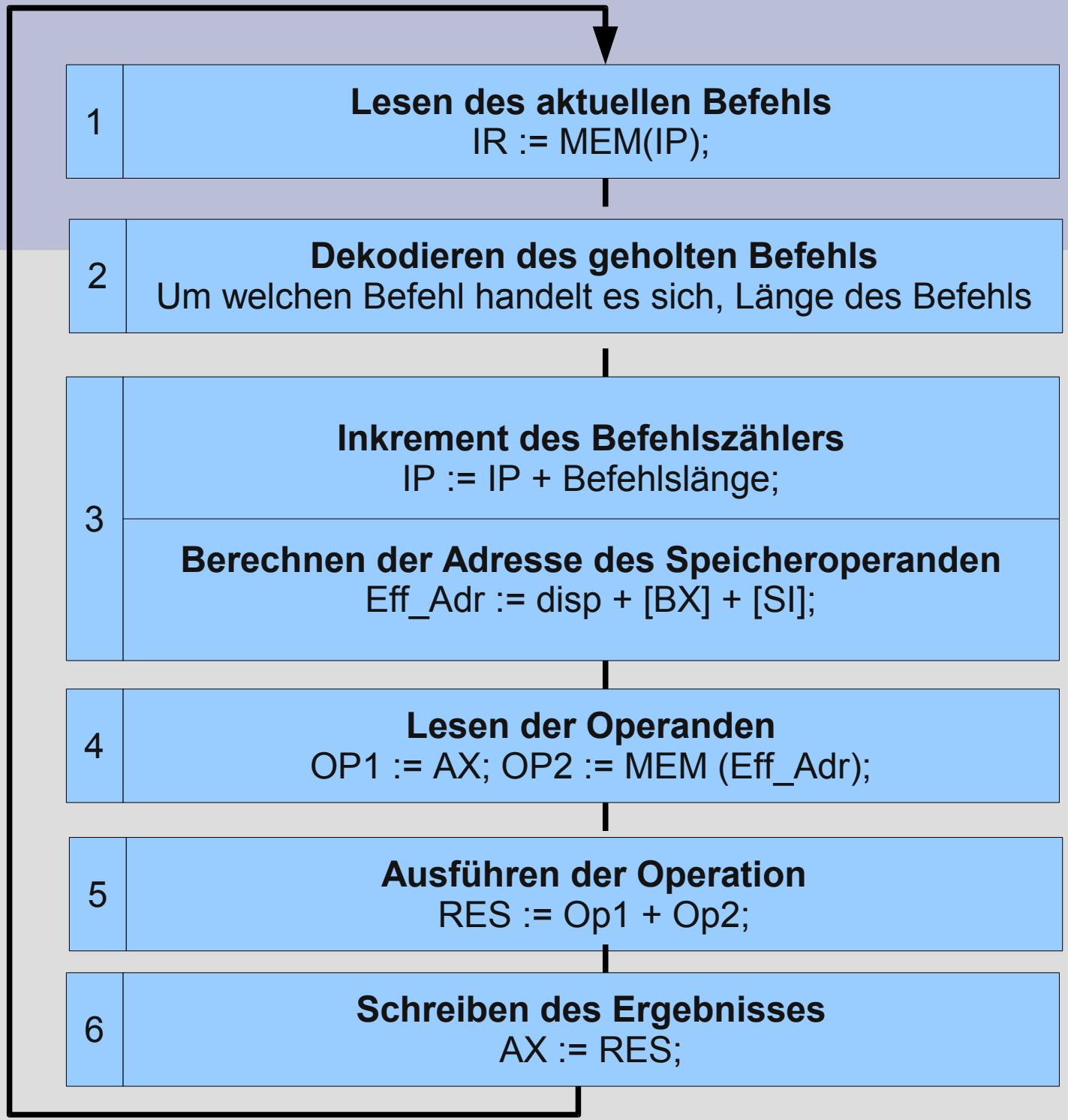


von-Neumann-Rechner:
entweder Lesen eines Befehls (fetch)
oder Ausführen eines Befehls (execute)

Pipelining (4)

- Realer Mikroprozessor (z.B. I8086):
  ADD AX,disp[BX][SI];
  AX := AX + MEM(disp + [BX] + [SI])

- **Detaillierung:**

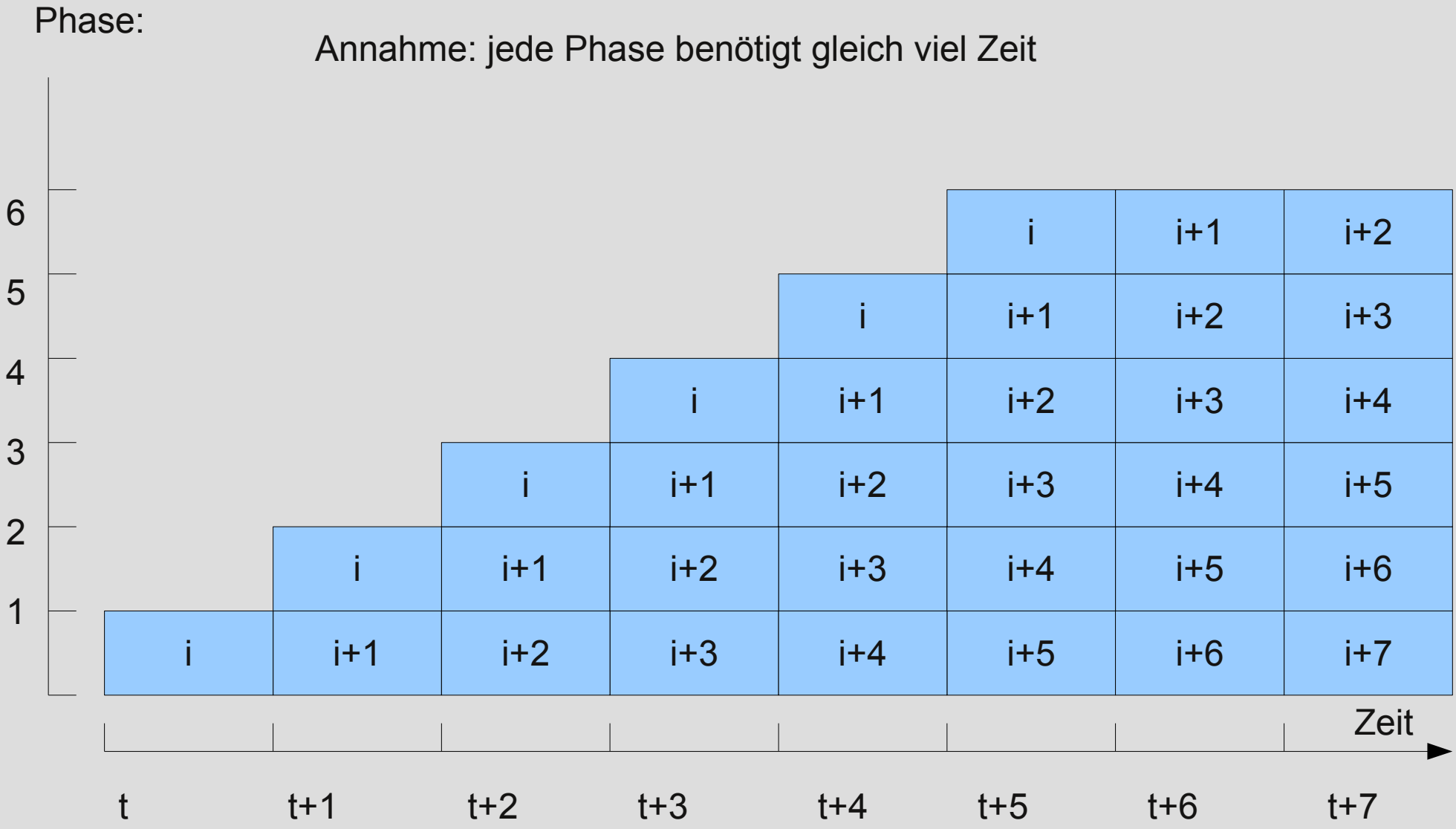

1
Lesen des aktuellen Befehls
IR := MEM(IP);
2
Dekodieren des geholten Befehls
Um welchen Befehl handelt es sich, Länge des Befehls
3
Inkrement des Befehlszählers
IP := IP + Befehlslänge;
Berechnen der Adresse des Speicheroperanden
Eff_Adr := disp + [BX] + [SI];
4
Lesen der Operanden
OP1 := AX; OP2 := MEM (Eff_Adr);
5
Ausführen der Operation
RES := Op1 + Op2;
6
Schreiben des Ergebnisses
AX := RES;

Pipelining (5)

- Realer Mikroprozessor (z.B. I8086):
  ADD AX,disp[BX][SI];
  AX := AX + MEM(disp + [BX] + [SI])

- Für jede der 6 Phasen gibt es spezielle Hardware, die brachliegt, wenn nicht gerade die jeweilige Phase ausgeführt wird.
- Ausweg:
- zeitlich überlappte Ausführung der Befehle (**Pipelining**)

Pipelining (6)



Pipelining (7)

- Konkretisierung:
- Beispiel MIPS-Befehlssatz (siehe auch Patterson/Hennessy: Rechnerorganistion- und Entwurf, Heidelberg, Spektrum 2005)
  - load word (lw)
  - store word (sw)
  - add (add)
  - subtract (sub)
  - and (and)
  - or (or)
  - set less then (slt)
  - branch on equal (beq)

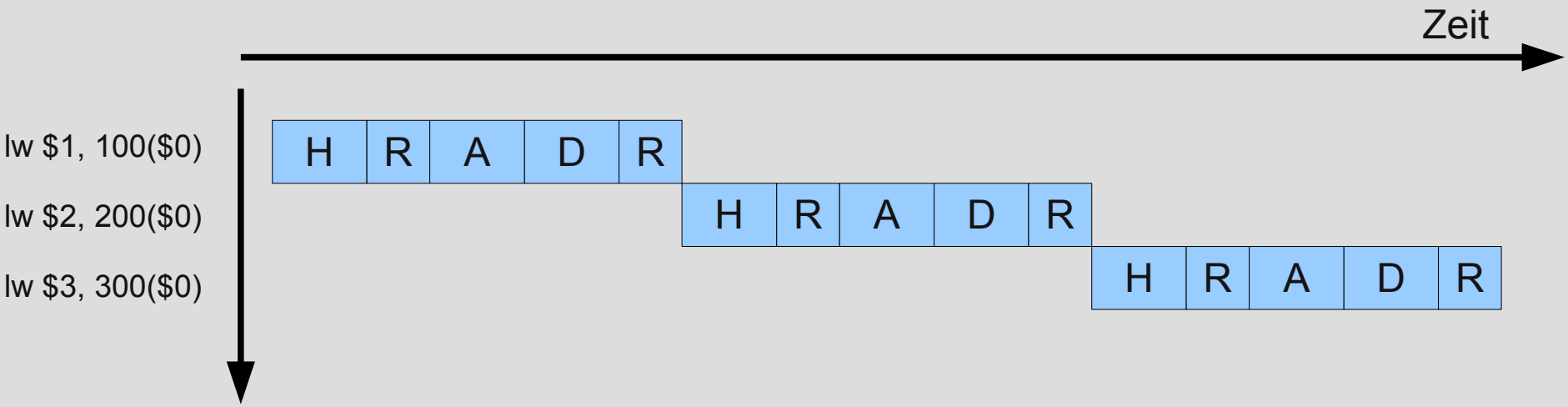
Pipelining (8)

- Konkretisierung:
  - load word (lw)
  - store word (sw)
  - add (add)
  - subtract (sub)
  - and (and)
  - or (or)
  - set less then (slt)
  - branch on equal (beq)

Klasse	Befehl hol.	Register les.	ALU-Op.	Dat.zgr.	Register schr.	Gesamt
lw	200ps	100 ps	200 ps	200 ps	100ps	800 ps
sw	200ps	100 ps	200 ps	200 ps		700 ps
R-Format						
(add,..)	200 ps	100 ps	200 ps		100 ps	600 ps
beq	200 ps	100 ps	200 ps			500 ps

Pipelining (9)

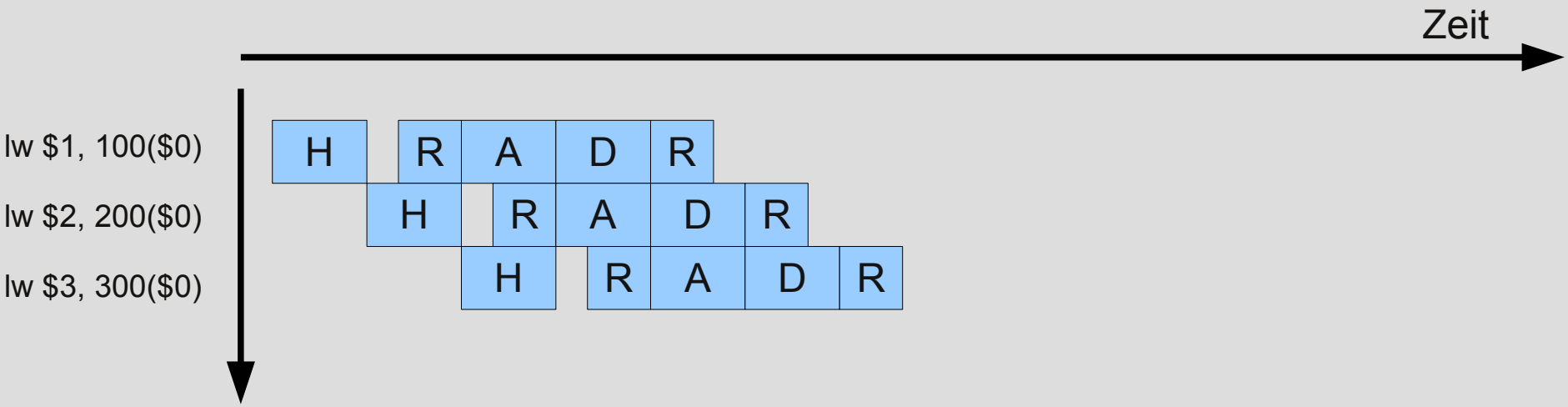
- Befehlsausführung lw sequentiell



H Befehl holen
R Registerzugriff
A ALU – Zugriff
D Datenzugriff

Pipelining (10)

- Befehlsausführung lw mit Pipelining



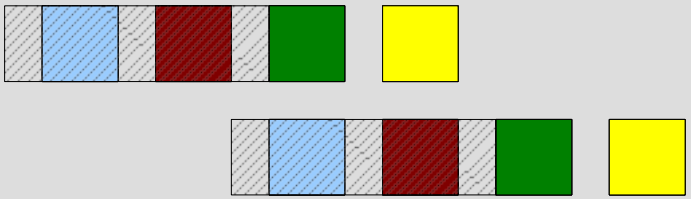
H Befehl holen
R Registerzugriff (lesen – 2. Hälfte, schreiben – 1. Hälfte des Taktzyklus)
A ALU – Zugriff
D Datenzugriff

Pipelining (11)

- Hemmnisse:
  - Strukturkonflikte
  - Datenkonflikte
  - Steuerkonflikt

Pipelining (12)

- Hemmnisse:
  - Strukturkonflikte
    - Ein Ereignis, bei dem ein Befehl nicht im vorgesehenen Taktzyklus ausgeführt werden kann, da die Hardware die Befehlskombination nicht unterstützt, die zum Ausführen im angegebenen Taktzyklus festgelegt wurde.
    - Beispiel Wäschewaschen:
      - es gibt keine separate Waschmaschine und keinen separaten Trockner, sondern ein Waschmaschinen-Trockner-Kombigerät



Pipelining (13)

- Hemmnisse:
  - Datenkonflikte
    - Ein Ereignis, bei dem ein Befehl nicht im vorgesehenen Taktzyklus ausgeführt werden kann, weil Daten zum Ausführen des Befehls noch nicht verfügbar sind.
    - Beispiel Wäschewaschen:
      - Sie stellen bei Zusammenlegen fest, dass eine Socke fehlt. Mögliche Strategie: Sie gehen in Ihr Zimmer und suchen nach der Socke. Dann muss das Zusammenlegen getrockneter Wäsche und das Trocknen gewaschener Wäsche warten.

Pipelining (14)

- Hemmnisse:
  - Datenkonflikte
    - Datenkonflikt aufgrund der Abhängigkeit eines Befehls von einem früher begonnenen Befehl
    - Beispiel:

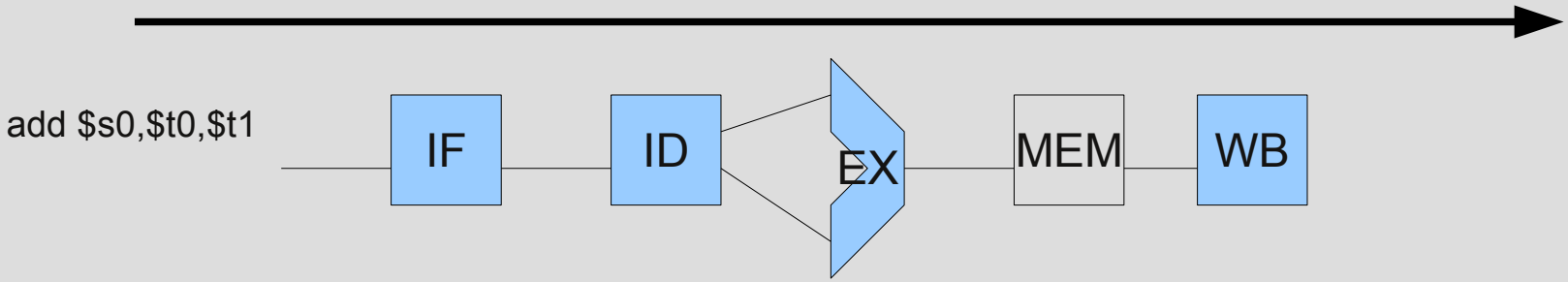
```
add   $s0, $t0, $t1
sub   $t2,  $s0, $t3
```

sub-Befehl benötigt als Eingabeoperand Ergebnis des add-Befehls

Bubbles (Wartetakte, Pipelineleerlauf)

Pipelining (15)

- Hemmnisse:
  - Datenkonflikte
    - Gegenmaßnahme: **Forwarding oder Bypassing:**
      Fehlende Datenelemente aus internen Pufferspeichern



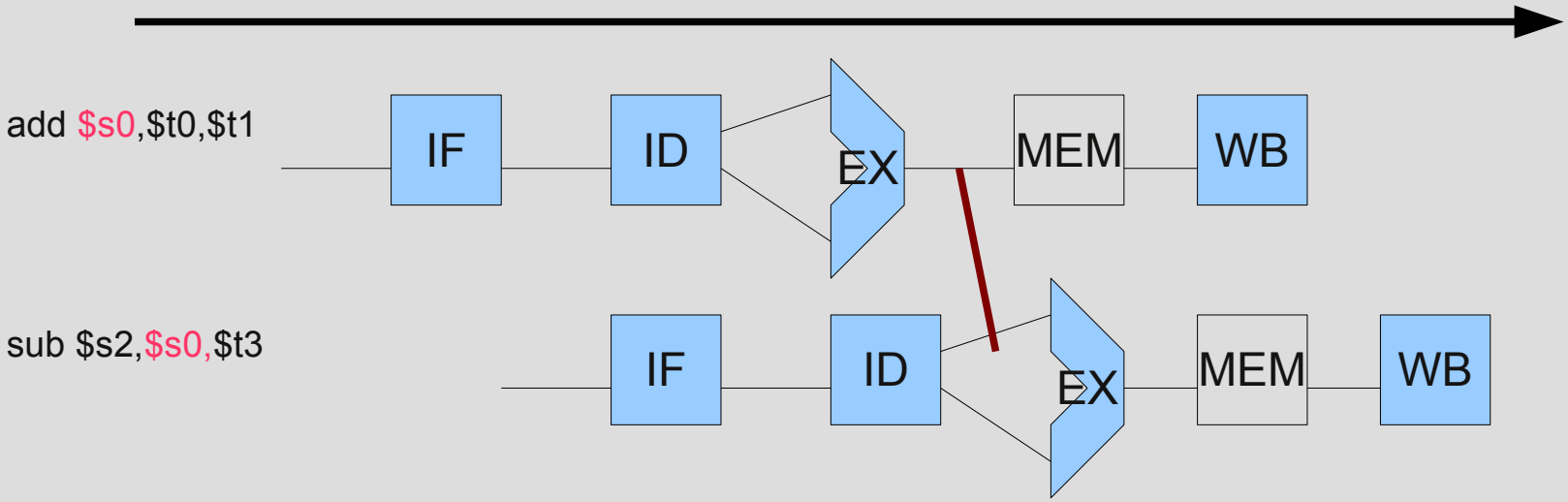
5 Pipelinestufen:

IF	Instruction Fetch (Befehl holen)
ID	Instruction Decode (Register lesen)
EX	Execute (Ausführungsstufe)
MEM	Memory Access (Speicherzugriff)
WB	Write Back (Register schreiben)

blau hinterlegt: Befehl benutzt die betreffende Pielinestufe

Pipelining (16)

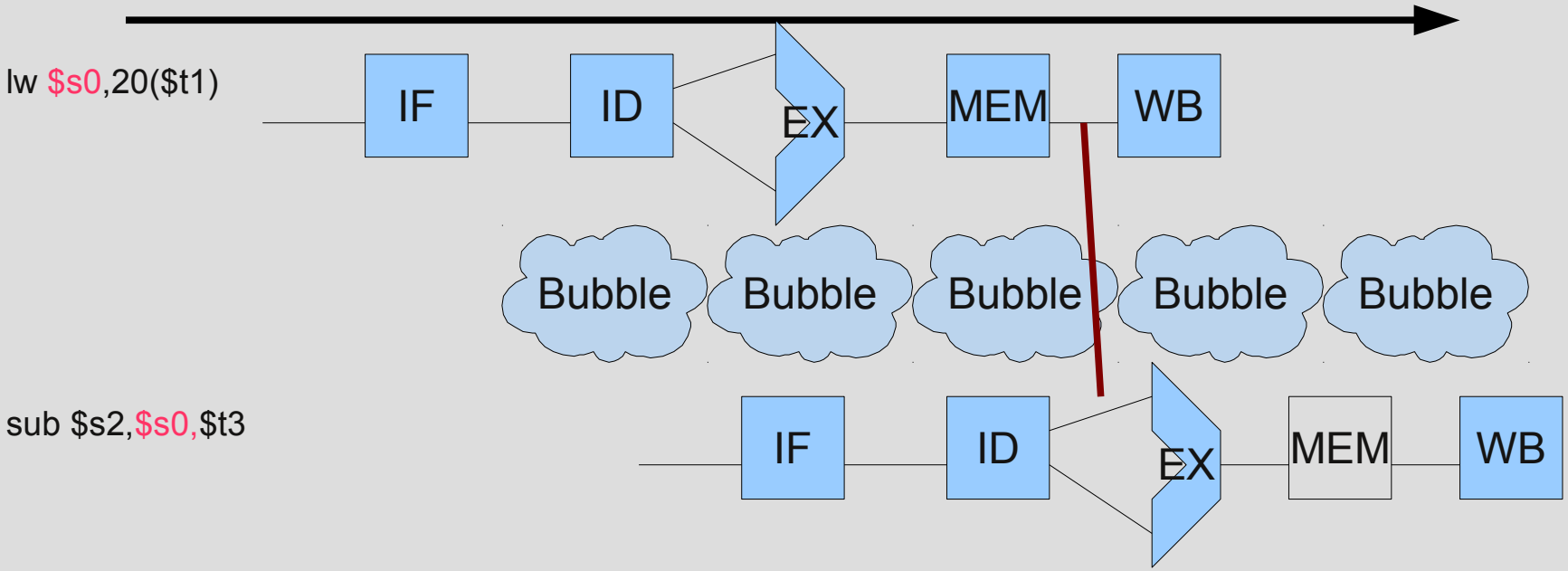
- Hemmnisse:
  - Datenkonflikte
    - Gegenmaßnahme: **Forwarding oder Bypassing:** Fehlende Datenelemente aus internen Pufferspeichern



Pipelining (17)

- Hemmnisse:
  - Datenkonflikte
    - Gegenmaßnahme: **Forwarding oder Bypassing:**

    Problem: R-Befehl verwendet direkt Daten aus Ladebefehl



Pipelining (18)

- Hemmnisse:
  - Datenkonflikte

    A = B + E;
    C = B + F;

    MIPS-Code: (alle Variablen im Speicher, Offset mit Register $t0 adressierbar)

```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
add   $t3, $t1, $t2
sw    $t3, 12($t0)
lw    $t4, 8($t0)
add   $t5, $t1,$t4
sw    $t5, 16($t0)
```

Pipelining (19)

- Hemmnisse:
  - Datenkonflikte

    A = B + E;
    C = B + F;

    MIPS-Code: (alle Variablen im Speicher, Offset mit Register $t0 adressierbar)

```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
add   $t3, $t1, $t2     Konflikt, wegen lw --> R-Befehl --> Bubble
sw    $t3, 12($t0)
lw    $t4, 8($t0)
add   $t5, $t1,$t4      Konflikt, wegen lw --> R-Befehl --> Bubble
sw    $t5, 16($t0)
```

Pipelining (20)

- Hemmnisse:
  - Datenkonflikte

    A = B + E;

    C = B + F;

    Umordnung:

```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
lw    $t4, 8($t0)
add   $t3, $t1, $t2
sw    $t3, 12($t0)
add   $t5, $t1,$t4
sw    $t5, 16($t0)
```

Pipelining (21)

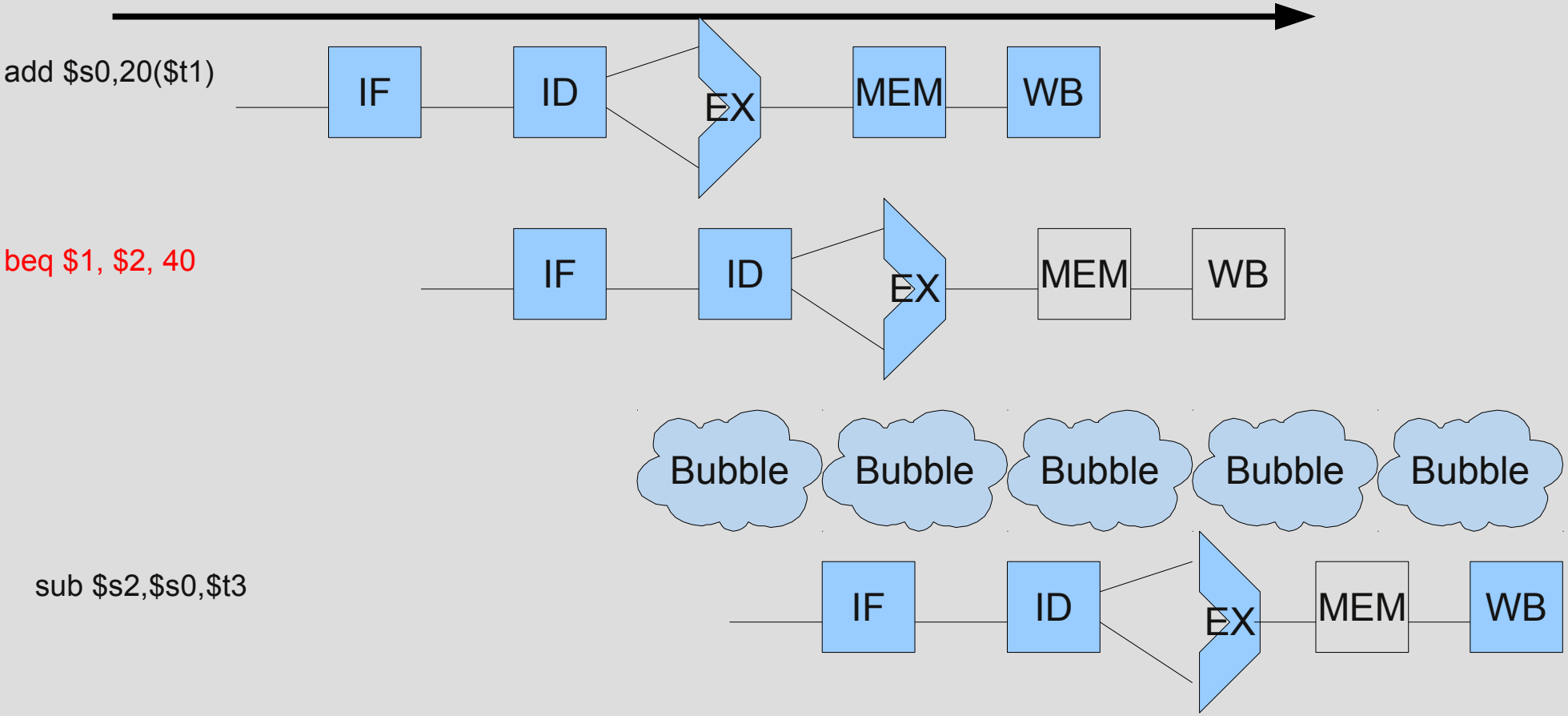
- Hemmnisse:
  - Steuerkonflikt:

  Ein Ereignis, bei dem der gewünschte Befehl nicht im gewünschten Taktzyklus ausgeführt werden kann, weil der Befehl, der geholt wurde, nicht der ist, der benötigt wird.

  im Computer: Entscheidungsbefehl
    - Adresse des zeitlich zunächst auszuführenden Befehls ist erst nach Abarbeitung des Entscheidungsbefehls bekannt!

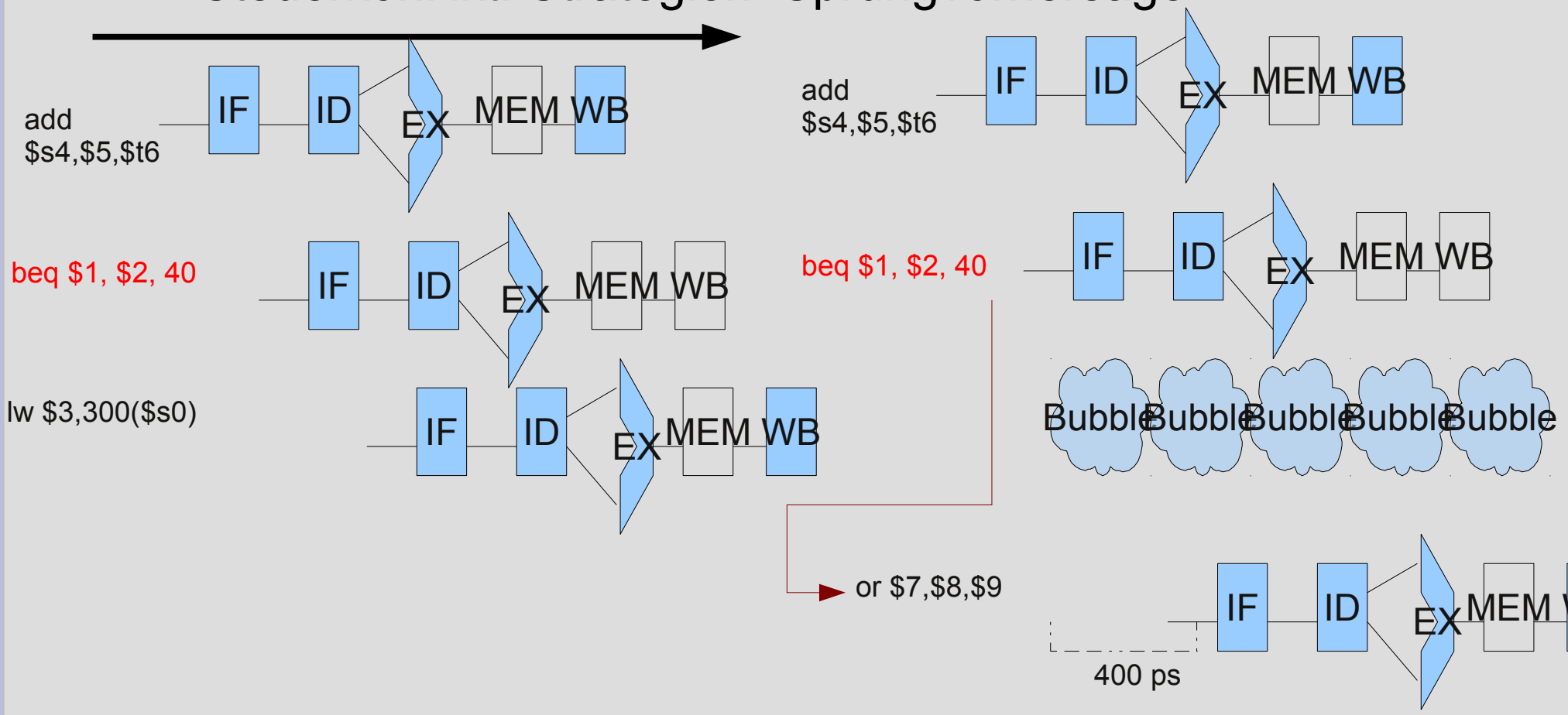
Pipelining (22)

- Hemmnisse:
  - Steuerkonflikt: Strategien Pipeline anhalten.



Pipelining (23)

- Hemmnisse:
  - Steuerkonflikt: Strategien Sprungvorhersage



Vektorrechner

Motivation/Hintergrund:

- **in wissenschaftlich-technischen Berechnungen haben Vektoren und Matrizen eine zentrale Bedeutung:**
  - **Abbildung von Vektoren auf eindimensionale, von Matrizen auf zweidimensionale Felder von reellen Zahlen**
- **Beispiel (C/C++)**

```
double vector[4];
double matrix[4][4];
```

Vektorrechner (2)

Verknüpfung zweier Vektorelemente:

Addition zweier Vektoren x und y wird zurückgeführt auf die Addition von Elementen mit gleichen Indizes:

```
for ( i= 0; i < 4; i++)
    z[i] = x[i] + y[i];
```

oder nach „Aufrollen“ der Schleife (dies ist eine gängige Aktion bei Optimierungsfunktionen des Compilers:

```
z[0] = x[0] + y[0]; ...
z[3] = x[3] + y[3];
```

Vektorrechner (3)

Die Datenflussanalyse der Anweisungen

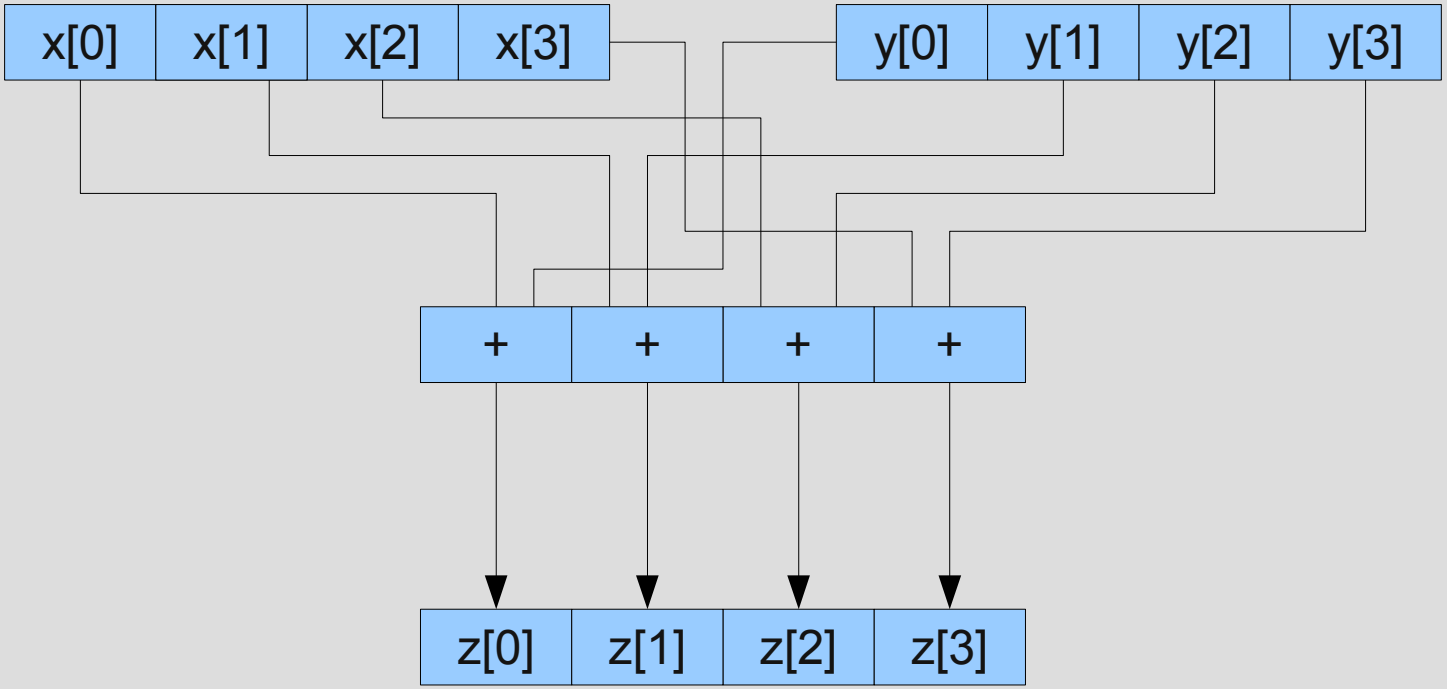
```
z[0] = x[0] + y[0]; ...
z[3] = x[3] + y[3];
```

zeigt, dass zwischen diesen Anweisungen keinerlei Abhängigkeiten bestehen. Sie sind also maximal parallel ausführbar.

Voraussetzung (siehe Schlussfolgerung 4 aus Kapitel Datenflussanalyse):

- Register für mehrere (hier 3 * 4) reelle Zahlen
- mehrere (hier 4) Gleitkommaaddierer

Vektorrechner (4)



Vektorrechner (5)

Probleme:

- Registerbreite und Anzahl der Gleitkommaverarbeitungseinheiten ist endlich
- wenn Dimension der Vektoren größer ist als die zur Verfügung stehenden Registerbreiten, muss in „normalen“ Architekturen auf die maximale Parallelität verzichtet werden
- in **Vektorrechnern** existiert eine spezielle (aber kostspielige) Speicherorganisation, die die Behandlung solcher Fälle wirkungsvoll unterstützt.

Vektorrechner (6)

des weiteren existieren spezielle Vektorbefehle
(auf Maschinenbefehlsniveau), mit denen
man mehrstellige Verknüpfungen formuliert
(siehe Programmiersprache **APL**)
meist existiert auch zusätzlich Pipelining

Vektorrechner (7)

Vektorprozessoren werden vor allem im High-Performance-Computing (HPC) genutzt.

Beispiele:

- Cray-Supercomputer (nutzten Vektorprozessoren)
- der über mehrere Jahre hinweg leistungsfähigste Computer der Welt, der Earth Simulator, arbeitet mit Vektorprozessoren von NEC.

Vektorrechner (8)

Weitere Anbieter von Vektorrechnern

- Convex Computer Corporation, z.B. mit der C38xx-Serie, die GaAs-Technologie einsetzte
- Fujitsu Siemens Computers mit ihrer VPP-Serie.

Superskalare und andere Techniken

Einführung

- Vektorrechner: Zusammenfassung mehrerer **gleichartiger** Befehle zu einem mehrstelligen Befehl (Vektorbefehl)
- Zusammenfassung mehrere parallel abarbeitbarer **unterschiedlicher** Befehle nicht möglich
- Voraussetzung dafür:
  zusätzliche Ressourcen

superskalare Architekturen

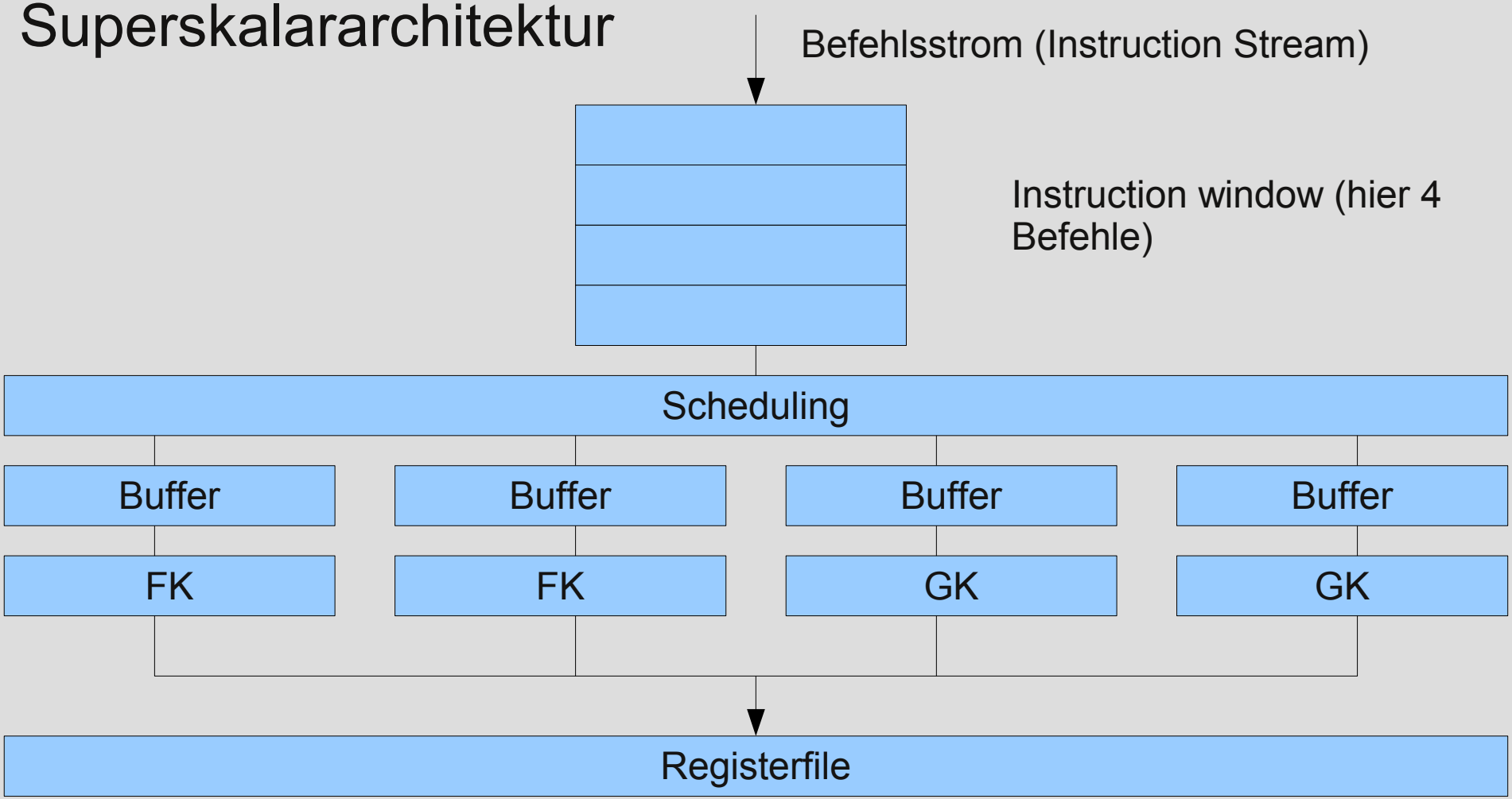
Siehe http://www.kreissl.info/ra_08.php

Superskalare und andere Techniken (2)

- Was bedeutet superskalar?
  - Prozessor muss in der Lage sein, mehrere Befehle gleichzeitig pro Takt zu laden
  - Branches dürfen möglichst nicht zur Behinderung des Befehlsflusses führen (durch Sprungvorhersage)
  - Datenabhängigkeiten treten hier in erhöhtem Maße auf und müssen beherrschbar sein
  - Um echte Datenabhängigkeiten auflösen zu können, ist Out-of-Order Ausführung mit anschließenden „Sortierung“ notwendig

Superskalare und andere Technologien (3)

Superskalararchitektur



Superskalare und andere Techniken (4)

- Anzahl parallel ausführbarer Befehle ist begrenzt durch:
  - Befehlsfolge selbst (Datenflussanalyse!)
  - Anzahl vorhandener Ausführungseinheiten
- Beispiel:
  - die Befehlsfolge bestehe nur aus unabhängigen Festkommabefehlen, der Superskalarrechner besitze 2 FK- und 2 GK-Einheiten und kann damit potentiell 4 Befehle parallel ausführen. Hier kann er aber stets nur 2 Befehle parallel ausführen.

Superskalare und andere Techniken (5)

- Wenn die Befehlsfolge aus unabhängigen FK- und GK-Befehlen besteht, dann kann der Rechner 4 Befehle gleichzeitig ausführen, aber nur dann, wenn die Befehlsfolge geeignet vorsortiert wird. Sie muss aus 4er-Paketen aus je 2 FK und 2 GK-Befehlen bestehen.
- Diese Vorsortierug müsste der Compiler realisieren, der damit Kenntnisse über die Zielarchitektur verfügen müsste (statisches Scheduling)

Superskalare und andere Techniken (6)

- in einem typischen Superskalarprozessor kann die Hardware ein bis acht Befehle pro Takt aussenden
- Die Befehle müssen unabhängig sein und bestimmte *issue criteria* erfüllen
- die Hardware entscheidet zur Laufzeit *dynamisch* über das Mehrfachaussenden (multiple issue) von Befehlen

Superskalare und andere Techniken (7)

- die variierende Zahl ausgesendeter Befehle wird typisch nach erweiterten Tomasulo-Techniken (mit out-of-order execution) dynamisch scheduled

Tomasulo-Algorithmus

- Der Tomasulo-Algorithmus verfolgt das Ziel, die Ausführung von Befehlen fortzusetzen, selbst wenn Datenabhängigkeiten vorliegen.
  - Read-after-write-Hazards (RAW) (Prozessor verfolgt, wann ein Operand zur Verfügung steht)
  - Write-after-write- (WAW) und Write-after-read-Hazards (WAR) (relevante Registerinhalte beim Decodieren eines Befehls werden in speziellen Reservation Stations gesichert und so vor vorzeitigem Überschreiben geschützt)

Tomasulo-Algorithmus

- Komponenten:
  Functional Units (FU):
  Prozessorbausteine, die logisch/arithmetische Berechnungen ausführen. Es gibt hiervon meist mehrere; sie unterscheiden sich in der Art der Operationen, welche sie ausführen können (floating point, integer, load/store, etc.). Bei modernen Prozessoren ist fünf eine typische Zahl für die Anzahl an FUs.

Tomasulo-Algorithmus

- Reservation Stations (RS):
  - Diese implementieren Register Renaming und werden wie temporäre Register behandelt. Für jede FU gibt es zwei bis acht Reservation Stations. Eine Reservation Station enthält die auszuführende Operation, zwei Felder für die Werte der Operanden und zwei Felder für die Herkunft der Operanden, falls sie zum aktuellen Zeitpunkt noch nicht zur Verfügung stehen bzw. noch nicht gültig sind.

Tomasulo-Algorithmus

- Fetch and Decode
  - Holt Instruktionen in einen Befehlscache. Die Decode-Unit holt sich einen Teil der Befehle und versucht mehrere gleichzeitig zu decodieren (In-Order).
  - Dabei wird versucht Sprünge vorher zusagen.

Tomasulo-Algorithmus

- Dispatch and Issue
  - holt dekodierte Befehle aus Befehlspuffer und übergibt sie In-Order an die Reservation Stations der Execute Units, sobald alle Operanden verfügbar sind (Issue). Solange im Reorder Buffer Platz ist, reserviert sie ein Feld für diesen Befehl mit Hilfe des Tags und gibt dieses an die RS weiter. Wenn nicht wartet sie, bis ein Platz frei wird. (Dispatch Phase)

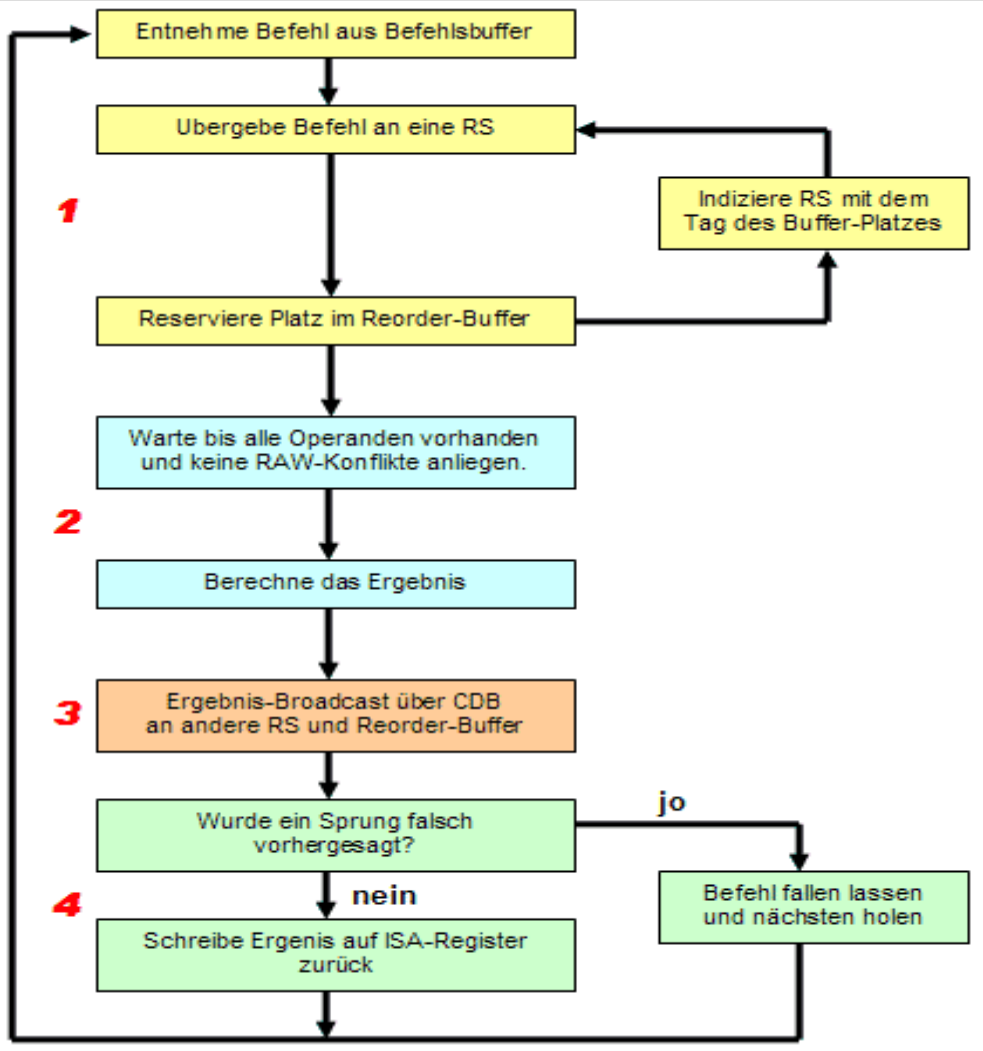
Tomasulo-Algorithmus

- Execution
  - Führt Befehle auf den Schattenregistern aus, um Data Hazards zu meiden.
  - Befehle werden in den Reorder-Buffer geschrieben und Out-Of-Order ausgeführt, solange es keine RAW-Konflikte gibt. Nach Beenden eines Befehls wird Ergebnis an alle RS gebroadcastet, so dass wartende Befehle fortfahren können.
  - (Write result)

Tomasulo-Algorithmus

- Commit
  - Die Commit/Completion oder Retire Einheit schreibt die Ergebnisse aus den Renaming Registern in die echten ISA Register zurück, nachdem sie geprüft hat, ob abhängige vorangehende Befehle ihre Ergebnisse geliefert haben und keine falsche Sprungvorhersage eingetreten war.

Tomasulo-Algorithmus



Tomasulo-Algorithmus

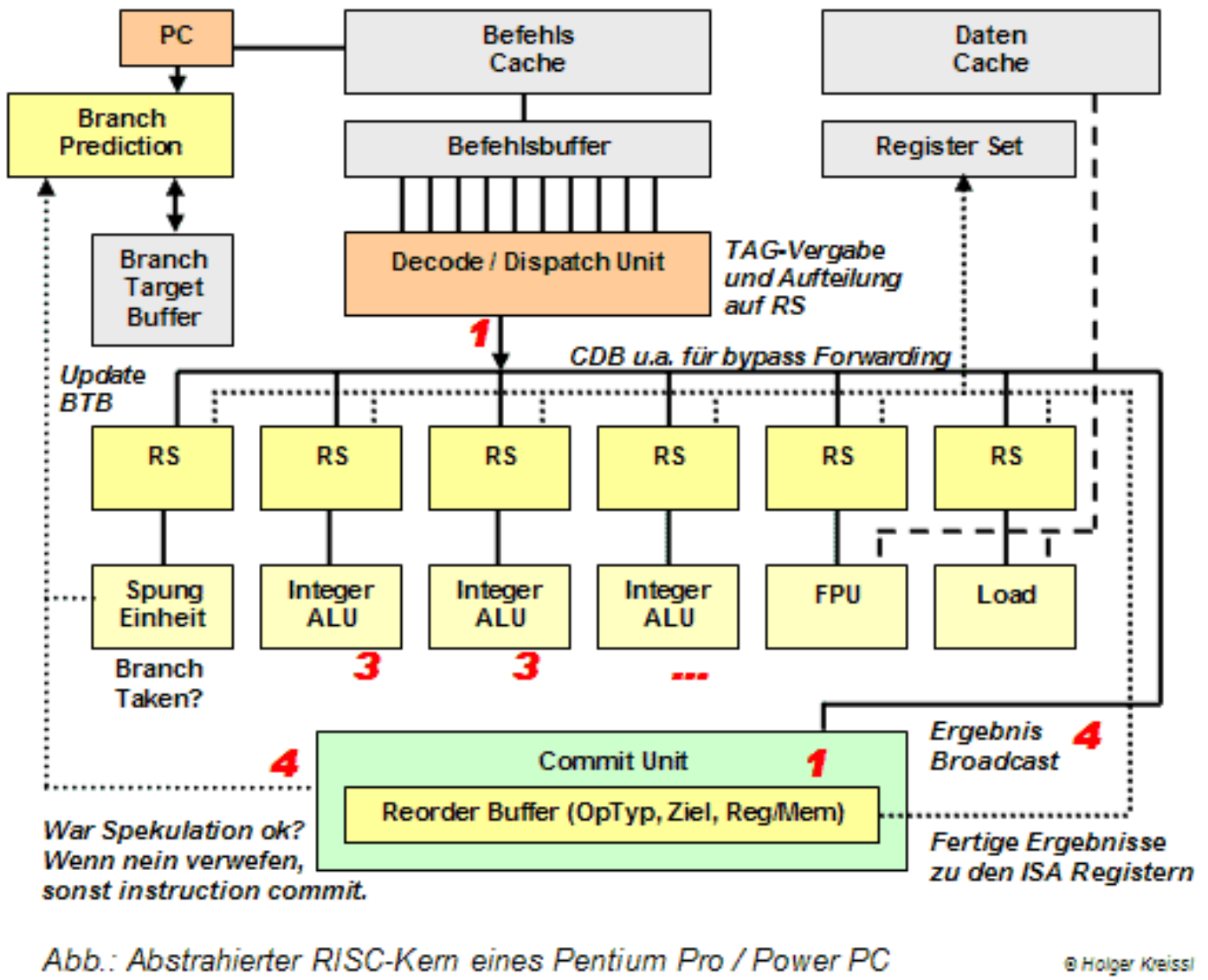
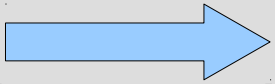


Abb.: Abstrahierter RISC-Kern eines Pentium Pro / Power PC

Superskalare und andere Techniken (7)

- Compilerscheduling (static) kann unterstützend angewendet werden, ist jedoch nicht unbedingt notwendig
- Programme für verschiedene Implementierungen einer Superskalararchitektur können objektkodekompatibel sein
- Im Vergleich zu VLIW-Prozessoren ist die Steuerlogik um ein Vielfaches komplexer.

Superskalare und andere Techniken (7)

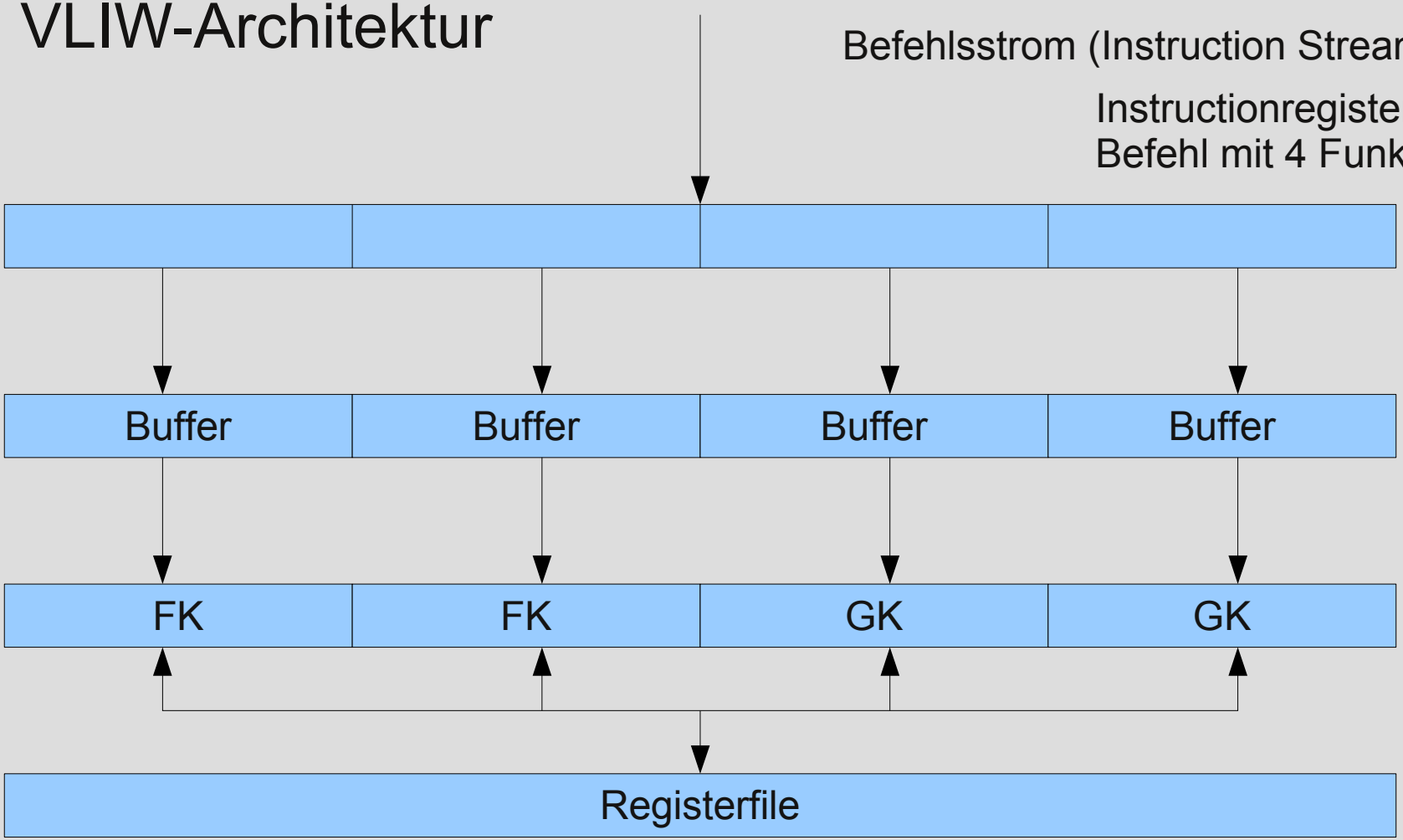
- Konsequente Weiterentwicklung dieses Denkansatzes 

  LIW (**l**ong **i**nstruction **w**ord) bzw. **VLIW** (**v**ery **l**ong **i**nstruction **w**ord) – Architekturen

  - Bestehen aus langen Befehlsworten, die eine Kettung parallel ausführbarer konventioneller Maschinenbefehle darstellen.
  - Compiler muss ausführen:
    - Vorsortierung
    - Scheduling (Zuordnung zu den Ausführungseinheiten), da die Stellung im VLIW-Befehl den Ausführungseinheiten fest zugeordnet ist.

Superskalare und andere Technologien (9)

VLIW-Architektur



Superskalare und andere Technologien (10)

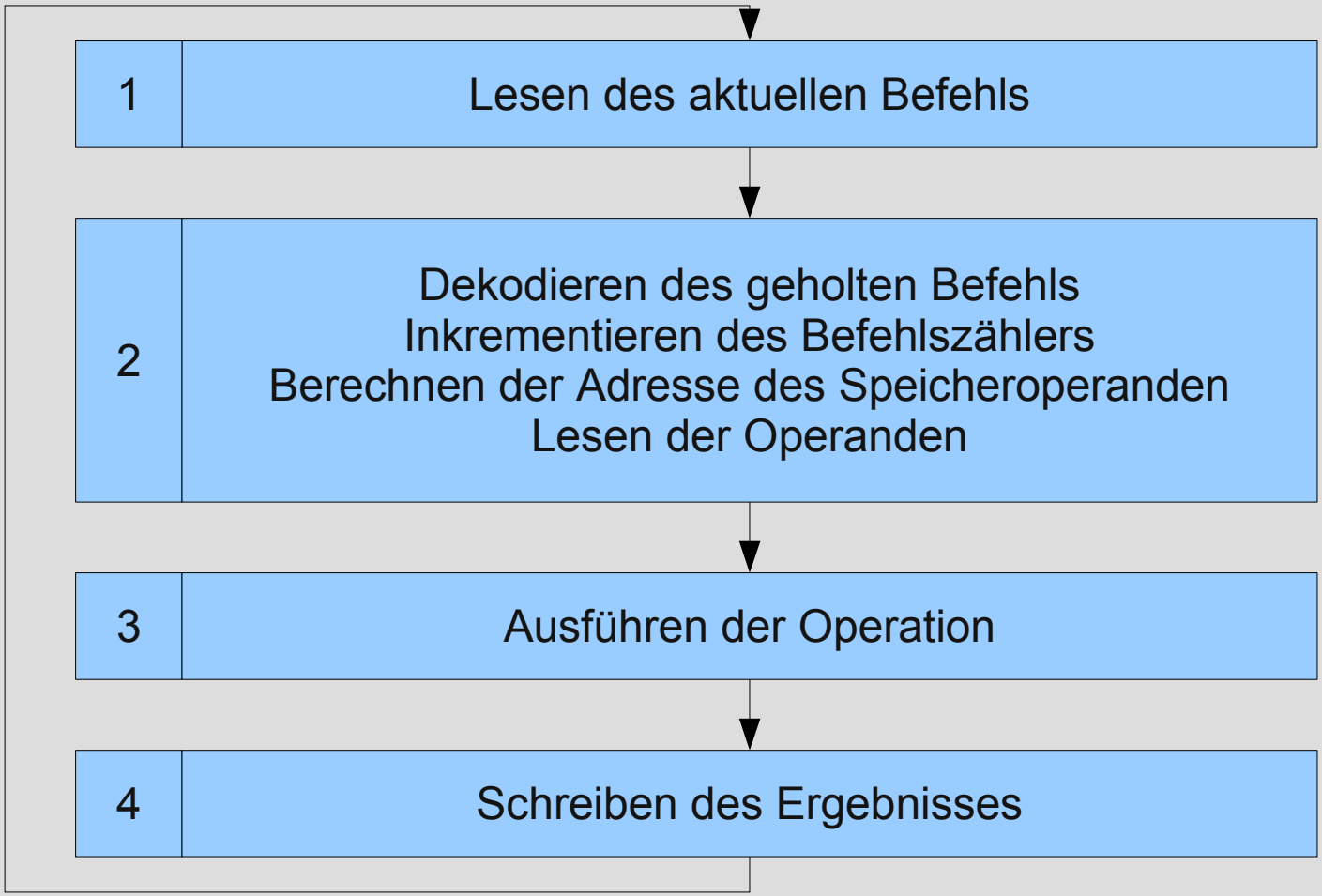
- ein VLIW-Prozessor sendet eine fest vordefinierte Anzahl von Befehlen aus, die als ein großer Befehl oder als ein festes Befehlspaket zusammen gefasst worden sind
- Der Compiler muss dazu ein solches Paket im Voraus (statisch) nach Möglichkeit zusammenstellen; notfalls sind NOP's einzuarbeiten (Kodeexpansion)
- Die Zuordnung der Befehle zu den einzelnen Ausführungseinheiten wird vom Compiler (statisch) vorgenommen. Er muss die Architektur genau kennen.

Superskalare und andere Technologien (11)

- Die Schedulinganalyse kann vergleichsweise zur Superskalararchitektur ein größeres Analysefenster (quai instruction window) bilden und diesbezüglich besser optimieren
- Da kein dynamisches Scheduling verwendet wird, entfällt ensprechend komplexe Steuerlogik sowie Schedulingoverhead.

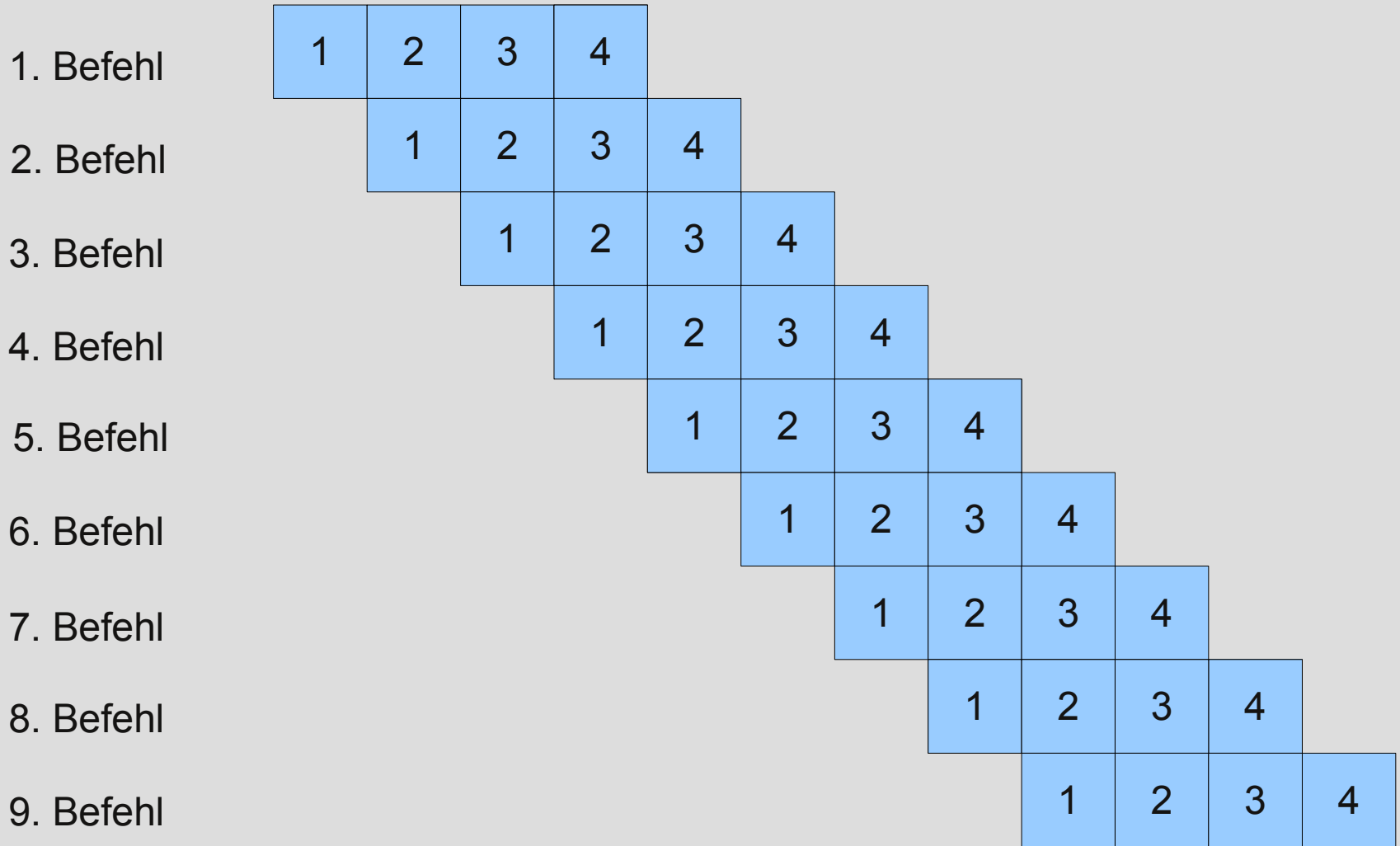
Superskalare und andere Techniken (12)

- Zusammenfassung:



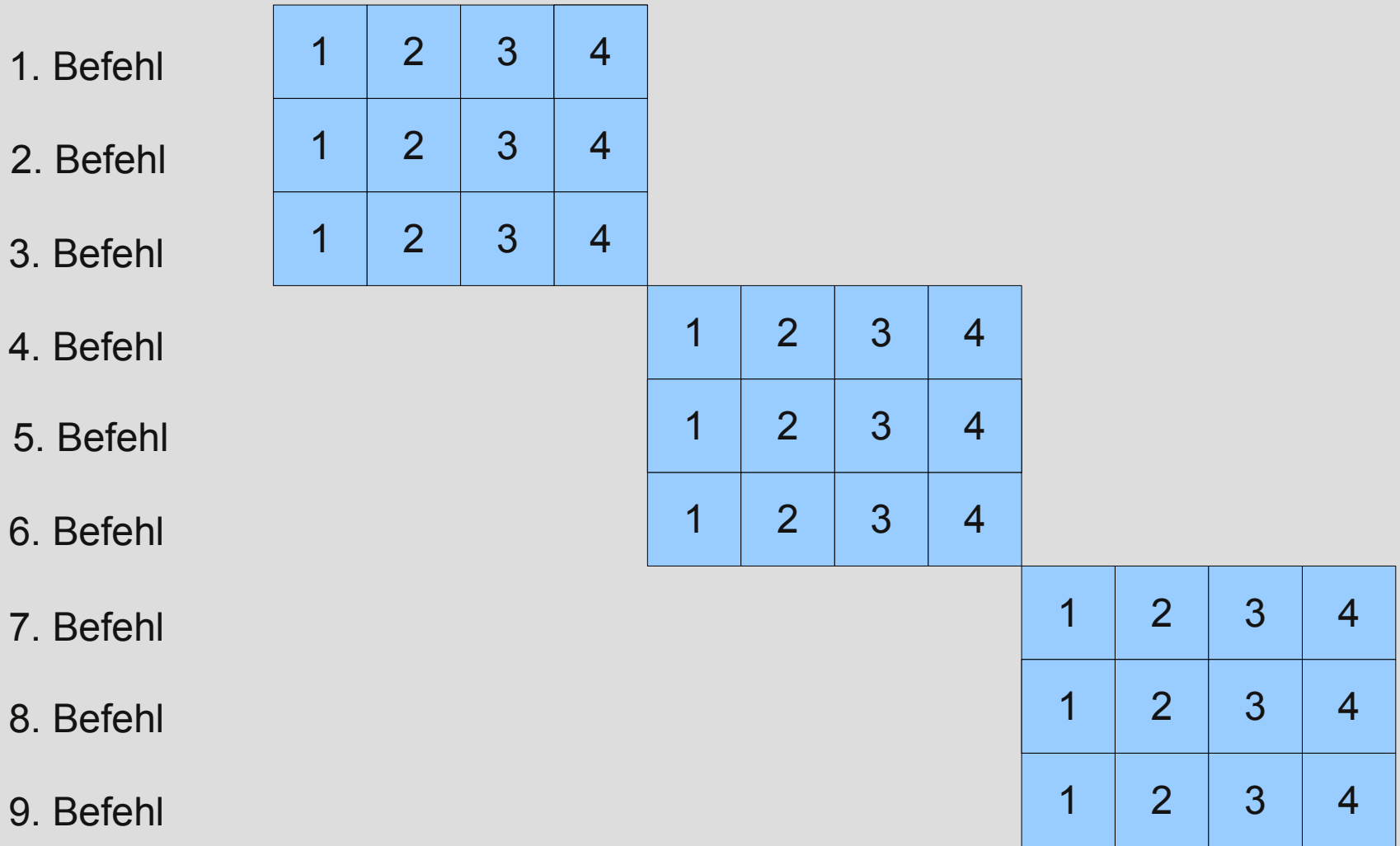
Superskalare und andere Techniken (13)

- Pipeline (4 stufig):



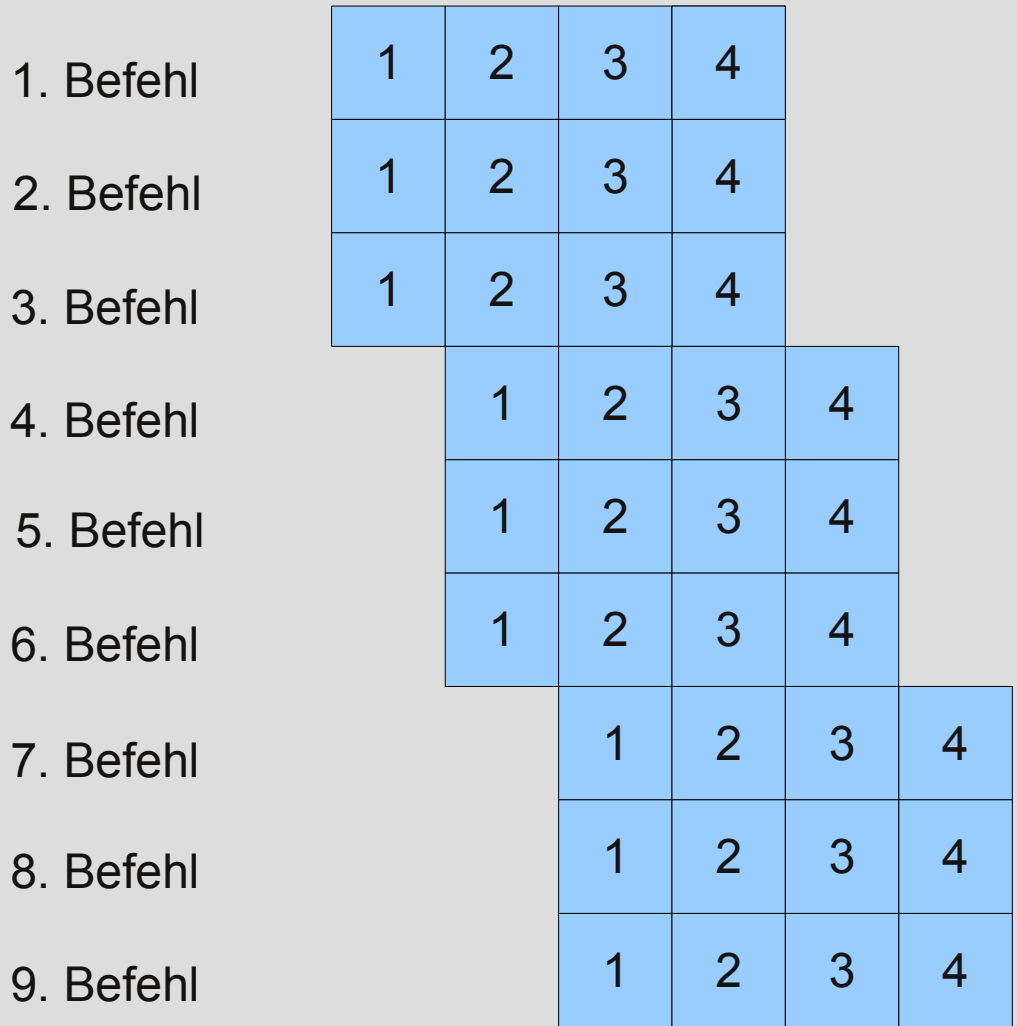
Superskalare und andere Techniken (14)

- Superskalar,(3 unabhängige Ausführungseinheiten, ohne Pipelining):



Superskalare und andere Techniken (15)

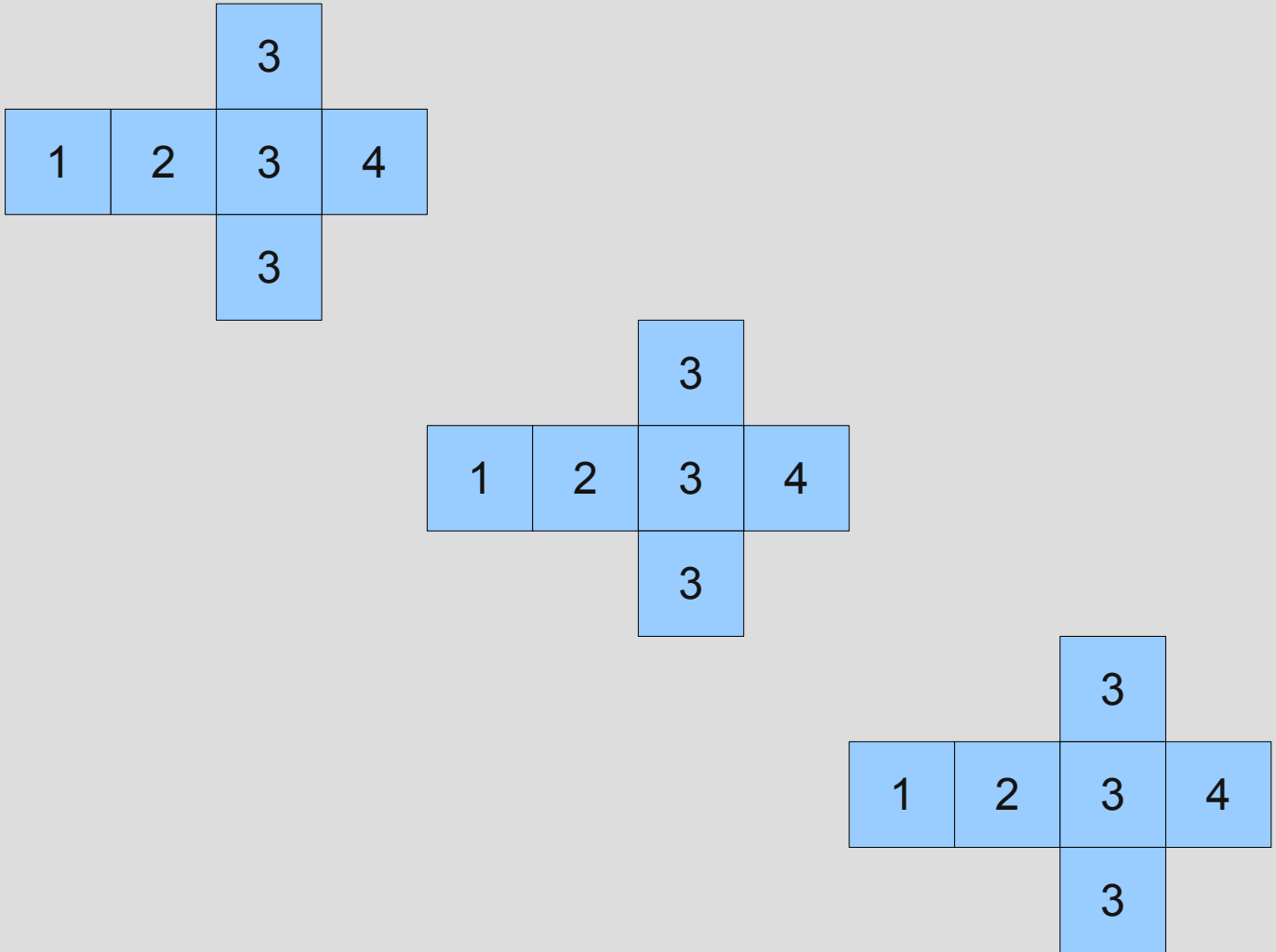
- Superskalar,(3 unabhängige Ausführungseinheiten, mit Pipelining):



Superskalare und andere Techniken (16)

- VLIW,(3 unabhängige Ausführungseinheiten, ohne Pipelining):

1. Befehl
2. Befehl
3. Befehl
4. Befehl
5. Befehl
6. Befehl
7. Befehl
8. Befehl
9. Befehl



Superskalare und andere Techniken (17)

- VLIW,(3 unabhängige Ausführungseinheiten, mit Pipelining):