cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Rechnerarchitektur I (RAI)

Nutzung von Parallelität

Prof. Dr. Akash Kumar

Chair for Processor Design

Content

- Ebenen der Parallelität
- Parallelverarbeitung auf Bit-Ebene
- Parallelverarbeitung auf Wort-Ebene
- Parallelverarbeitung auf Befehls-Ebene
  - Befehls-Pipelining
  - Daten-Pipelining
  - Super-Pipelining
  - Superskalar-Pipelining
  - VLIW-Architekturen

Ebenen der Parallelität

Bit-Ebene (Bit Level Parallelism - BLP)

Zusammenfassung mehrerer Bits (Paralleladdierer)

Wort-Ebene (Word Level Parallelism - WLP)

Single Instruction Multiple Data - SIMD (SSE, MMX)

Befehls-Ebene (Instruction Level Parallelism - ILP)

Pipelining-, Superskalar-, VLIW-Architektur

Kontrollfluss-Ebene (Thread Level Parallelism - TLP)

Multithreaded-Architektur

Programm-Ebene (Program Level Parallelism - PLP)

Multiprozessor-, Multicore-, Multicomputer-Architektur

Verarbeitungsleistung

CPI - Anzahl der Taktzyklen pro Befehl (Cycles Per Instruction)

IPC - Abgearbeitete Befehle pro Taktzyklus (Instruction Per Cycle)

T_{EXE} - Abarbeitungszeit

T_C - Taktzykluszeit (Taktperiodendauer)

f_C - Taktfrequenz, $f_C = \frac{1}{T_C}$

N - Anzahl der abzuarbeitenden Befehle

Abarbeitungszeit

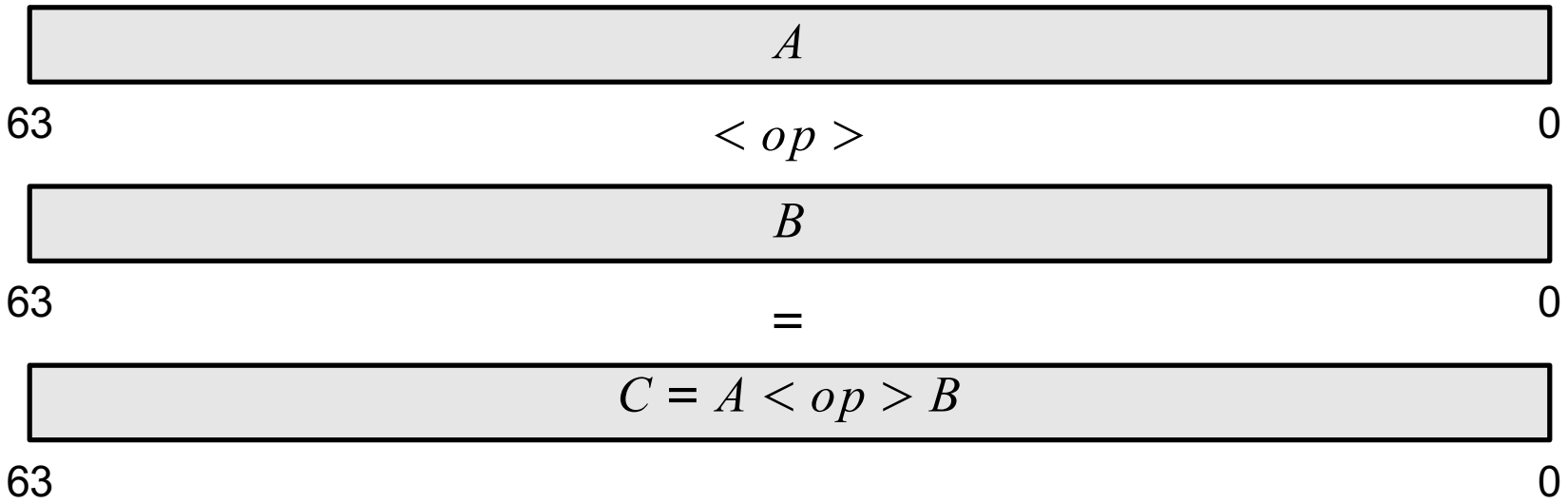
$$T_{EXE} = N \cdot CPI \cdot T_C; \qquad T_{EXE} = \frac{N \cdot T_C}{IPC}$$

CPI, IPC - charakteristische Durchschnittswerte – architekturspezifisch

Die Bewertung der Verarbeitungsleistung von Datenpfaden erfolgt typisch durch die Größen Cycles Per Instruction CPI und die mögliche Taktfrequenz f_C.

Parallelverarbeitung auf Bit-Ebene

64-Bit-Paralleloperation

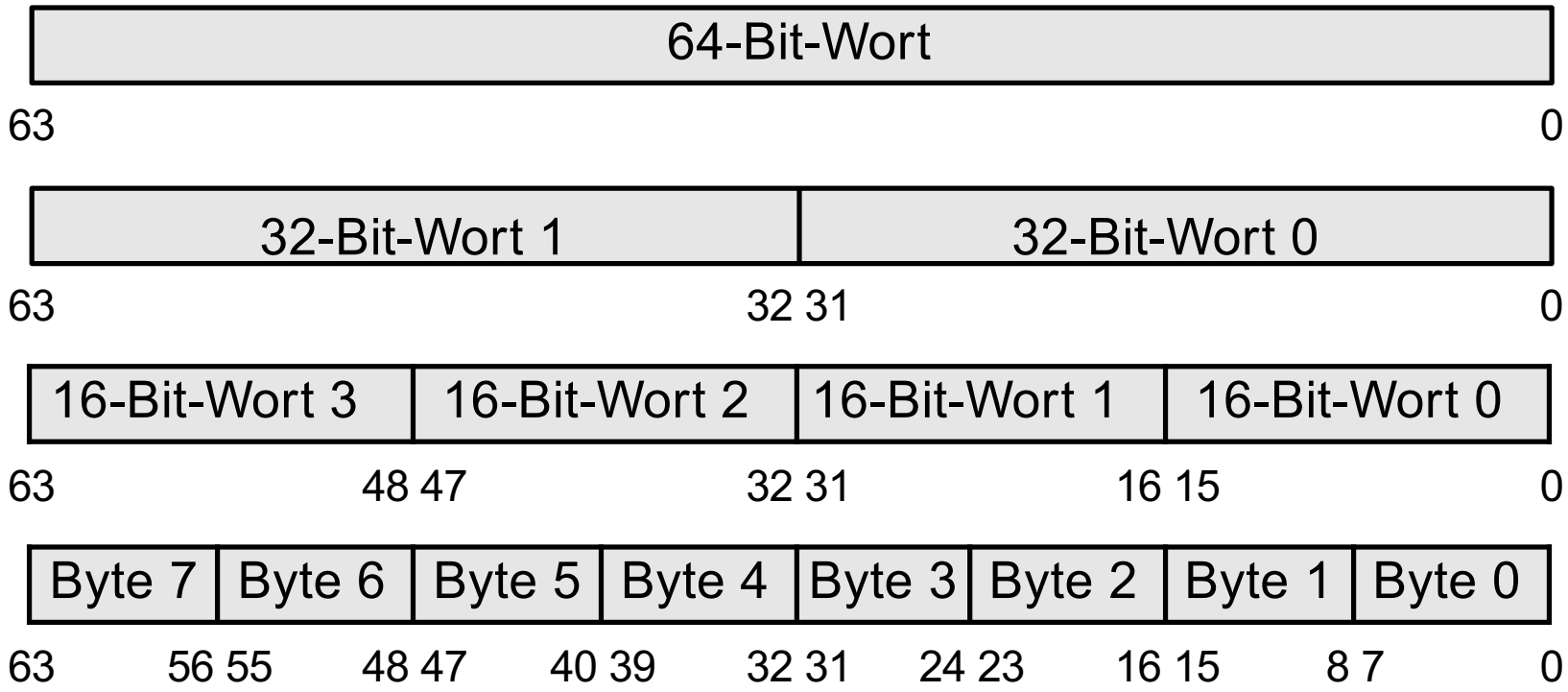


Bitparallele Verarbeitung des Darstellungsformates, in einem Taktzyklus möglich (logische Funktionen, parallele Addierer, Multiplizierer, Barrelshifter, . . .).

$$CPI = 1$$

Parallelverarbeitung auf Wort-Ebene

Subwort-Unterteilung eines 64-Bit-Wortes (Little Endian)



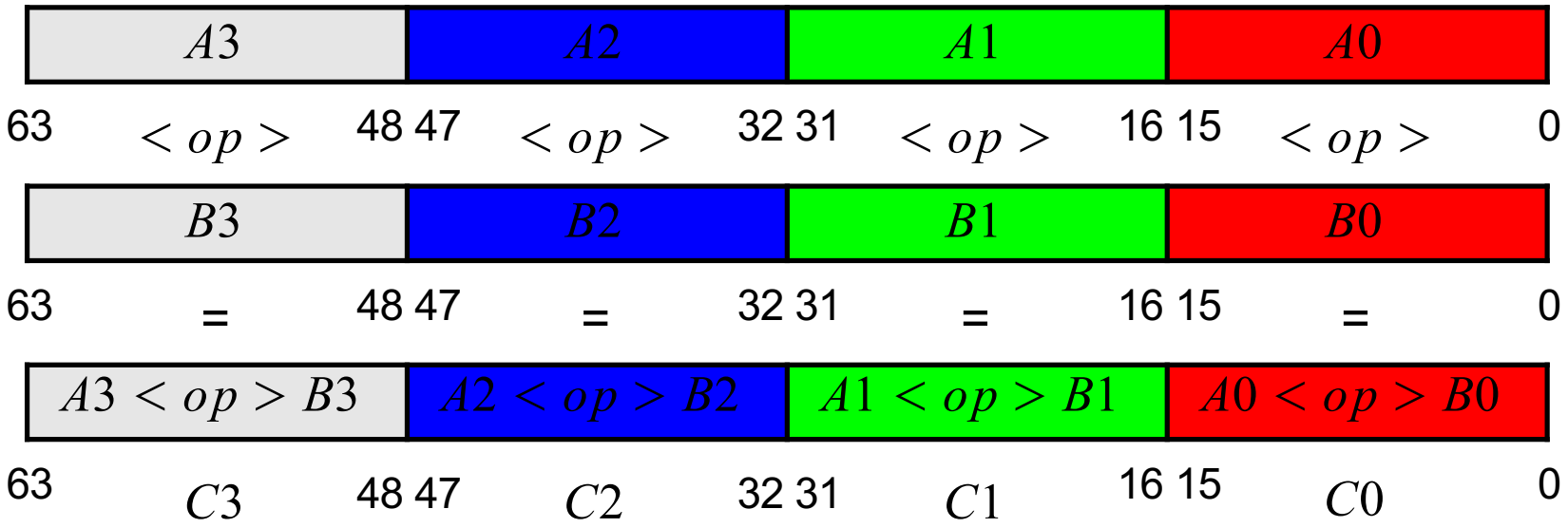
Unterteilung eines n-Bit-Wortes (internes Darstellungsformat) lückenlos (gepackt) in m n/m-Bit-Subworte gleicher Größe ($m = 2^i$, $i = 1, 2, 3, \ldots$).

SIMD-Prinzip

SIMD - Single Instruction Multiple Data (z.B. MMX- und SSE-Befehle)

Anwendung eines Befehls (Operation) auf mehrere Daten (Subwörter)

Parallelverarbeitung von 4 16-Bit-Subworten



Auftrennung der Übertragsweiterleitung . . . an den Subwortgrenzen

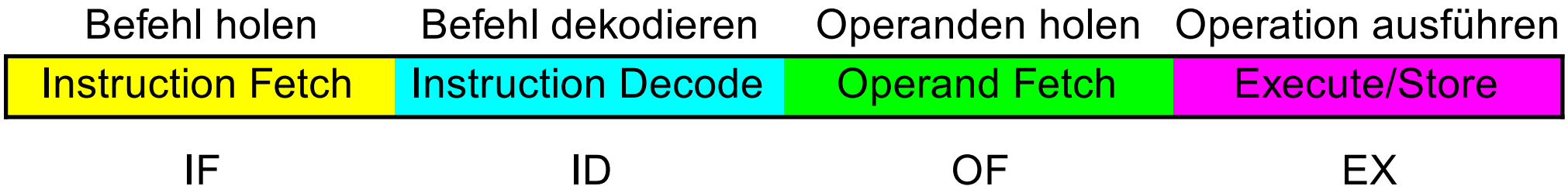
$$CPI = 0.25$$

Parallelverarbeitung auf Befehls-ebene

Varianten paralleler Abarbeitung mehrere Befehle

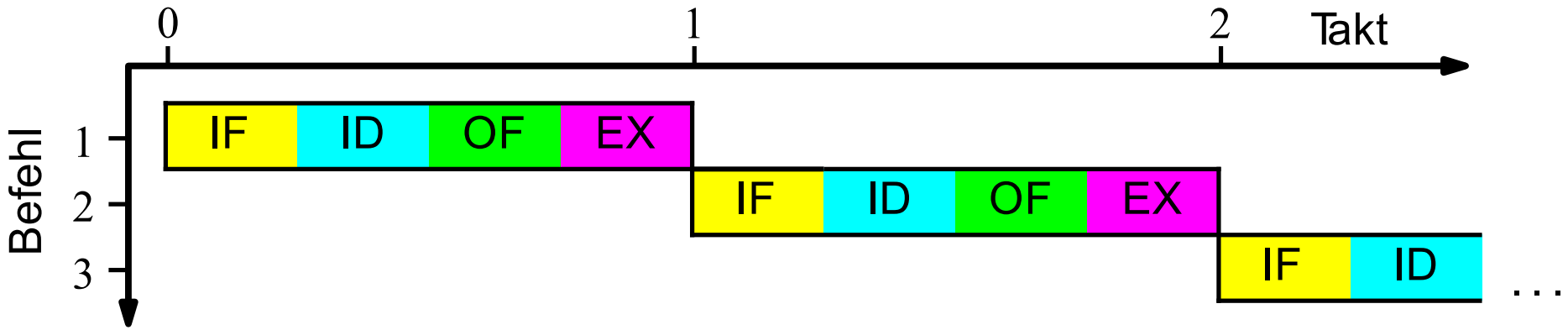
- Pipelining
- Superpipelining
- Superskalar
- VLIW (Very Long Instruction Word)

Phasen des Befehlszyklus (von Neumann)



Eintakt-Befehlsabarbeitung

1-Takt-Befehlsabarbeitung



$$CPI = 1$$

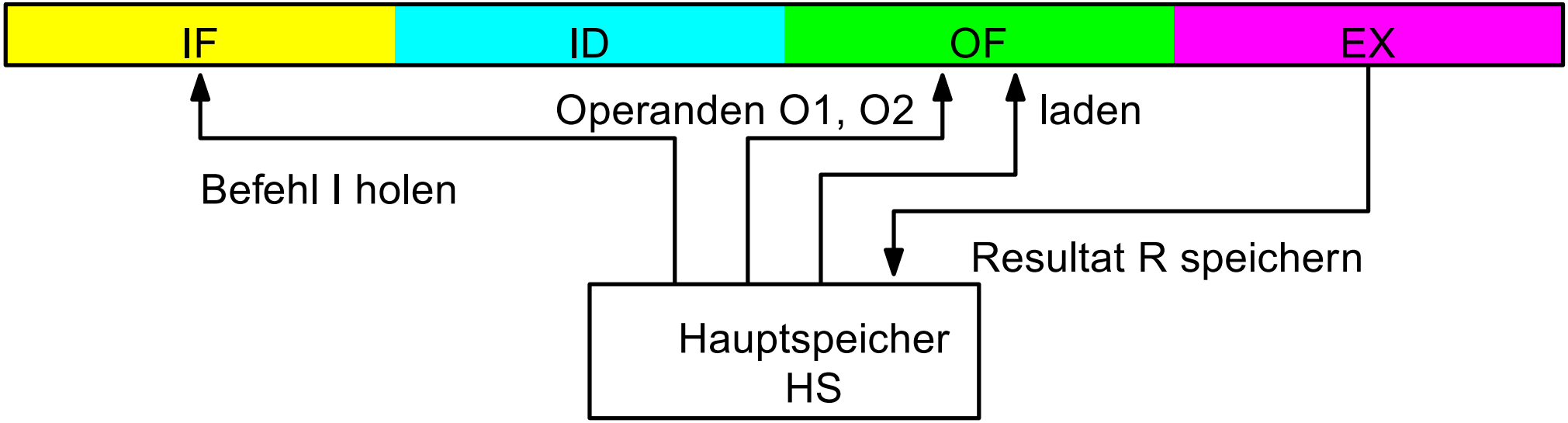
Problem: Nur ein Hauptspeicherzugriff pro Takt möglich.

⇒ Konflikte beim Speicherzugriff unvermeidbar → nicht praktisch realisierbar.

⇒ Taktperiode entspricht einem vollen Befehlszyklus → niedrige Taktfrequenz.

Speicher-Zugriffskonflikte

Speicher-Zugriffskonflikte bei 1-Takt-Befehlsabarbeitung

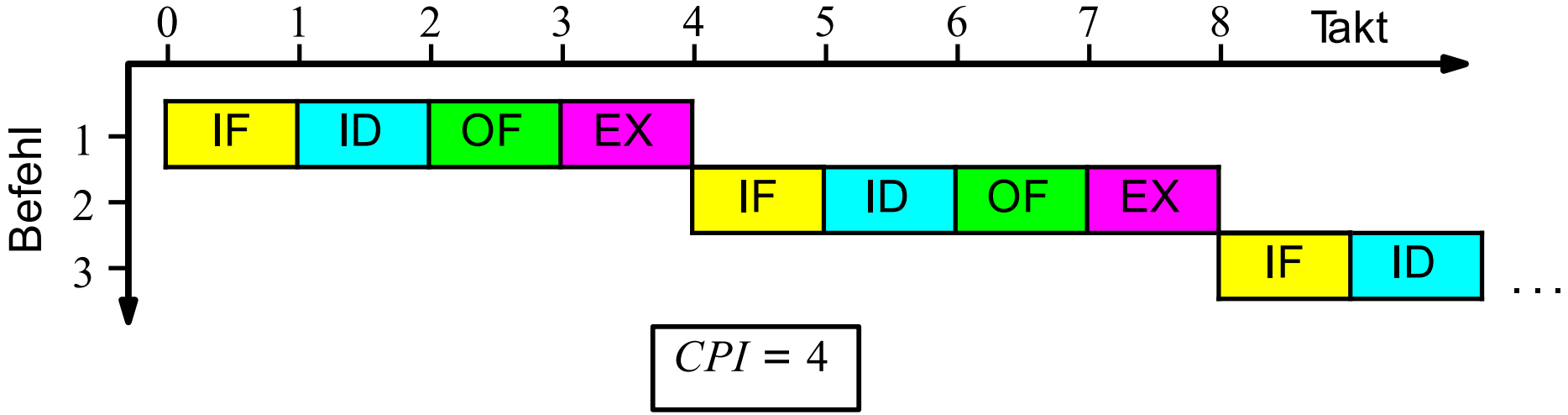


Bei einem typischen dyadischen Befehl I: R := O1 <op> O2 konkurrieren innerhalb eines Taktes 4 Speicherzugriffe. Mögliche Auswege sind:

- ⇒ Harvard-Architektur (getrennte Speicher und Busse für Daten und Befehle).
- ⇒ Load/Store Architektur mit Registersatz als Multiport-Speicher.
- ⇒ 4 Takte pro Befehl ⇒ Mehrtakt-Befehlsabarbeitung.

Mehrtakt-Befehlsabarbeitung

4-Takt-Befehlsabarbeitung



Unterteilung des Befehls in S einzelne unabhängige Phasen ($CPI = S$).

Trennung der einzelnen Phasen durch Einfügung von getakteten Registern.

Maximal S Speicherzugriffe pro Befehl möglich (pro Takt 1 Zugriff).

Problem: Maximale Taktfrequenz richtet sich nach der längsten Phase.

⇒ Optimierung der Phasenlängen durch Zusammenfassung, Unterteilung oder zusätzliche Phasen. Einfache kurze Phasen → hohe Taktfrequenz.

Fließband-Befehlsabarbeitung

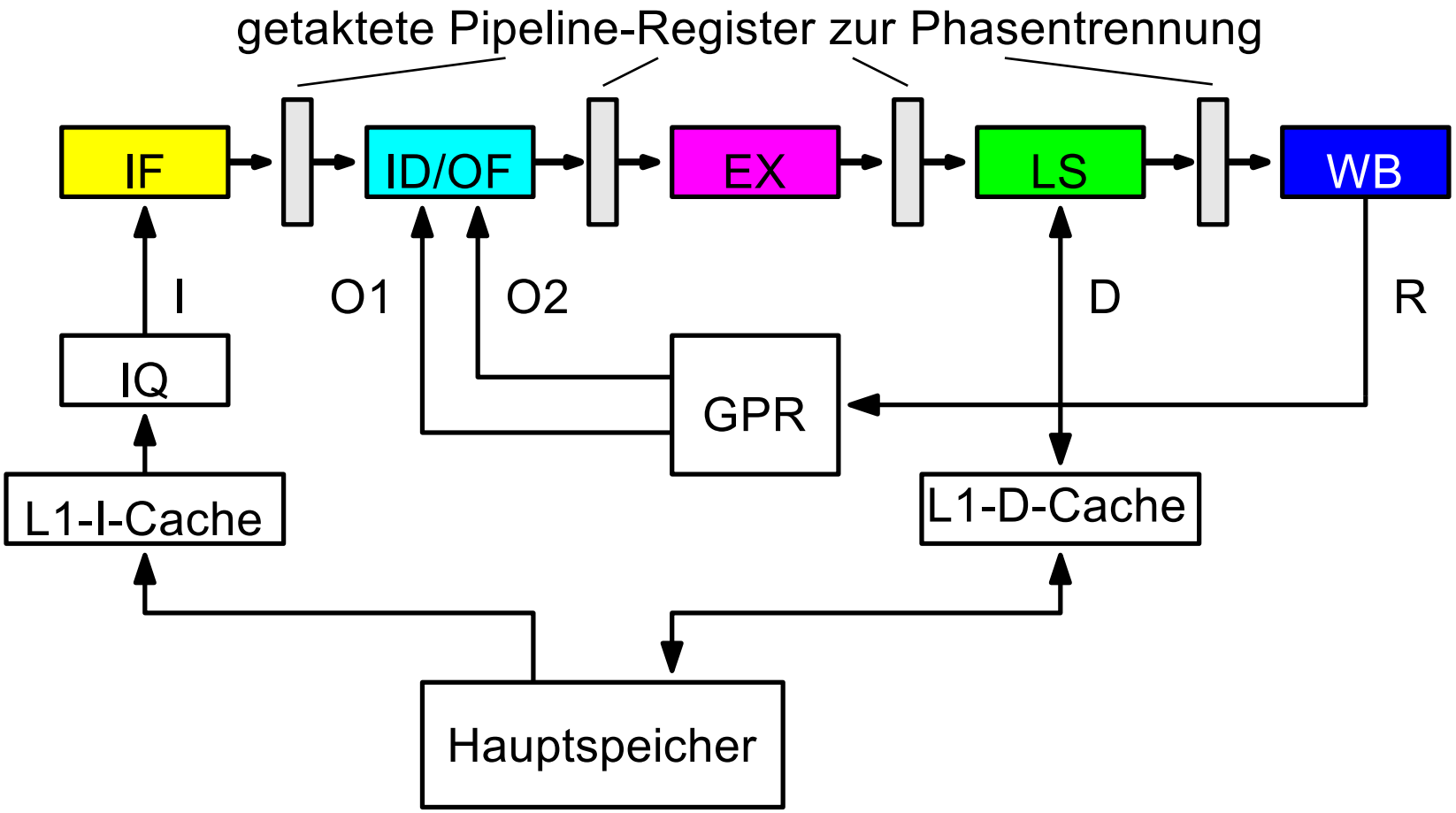
Umbau der Befehlsphasen analog Mehrtakt-Befehlsabarbeitung

IF-Phase	Zugriff auf Befehle kann über eine Befehlswarteschlange (Instruction Queue, Instruction Prefetching) erfolgen. Vermeidung von direkten Speicherzugriffen.
ID-Phase	Kann auch mit der OF-Phase zusammengelegt werden.
OF-Phase	Ausschließlich Zugriffe auf Registersatz, kein Speicherzugriff.
EX-Phase	Aufteilung auf mehrere Phasen möglich (EX1, EX2, EX3, . . .)
LS-Phase	Gesonderte Phase nur für Daten-Speicherzugriffe in einer Load/Store-Architektur (Load/Store, Memory).
WB-Phase	Gesonderte Phase nur für das Rückschreiben des Resultates in den Registersatz (Write Back), kein Speicherzugriff.

Es sind verschiedenste Phasen-Anordnungen und Pipeline-Längen denkbar.

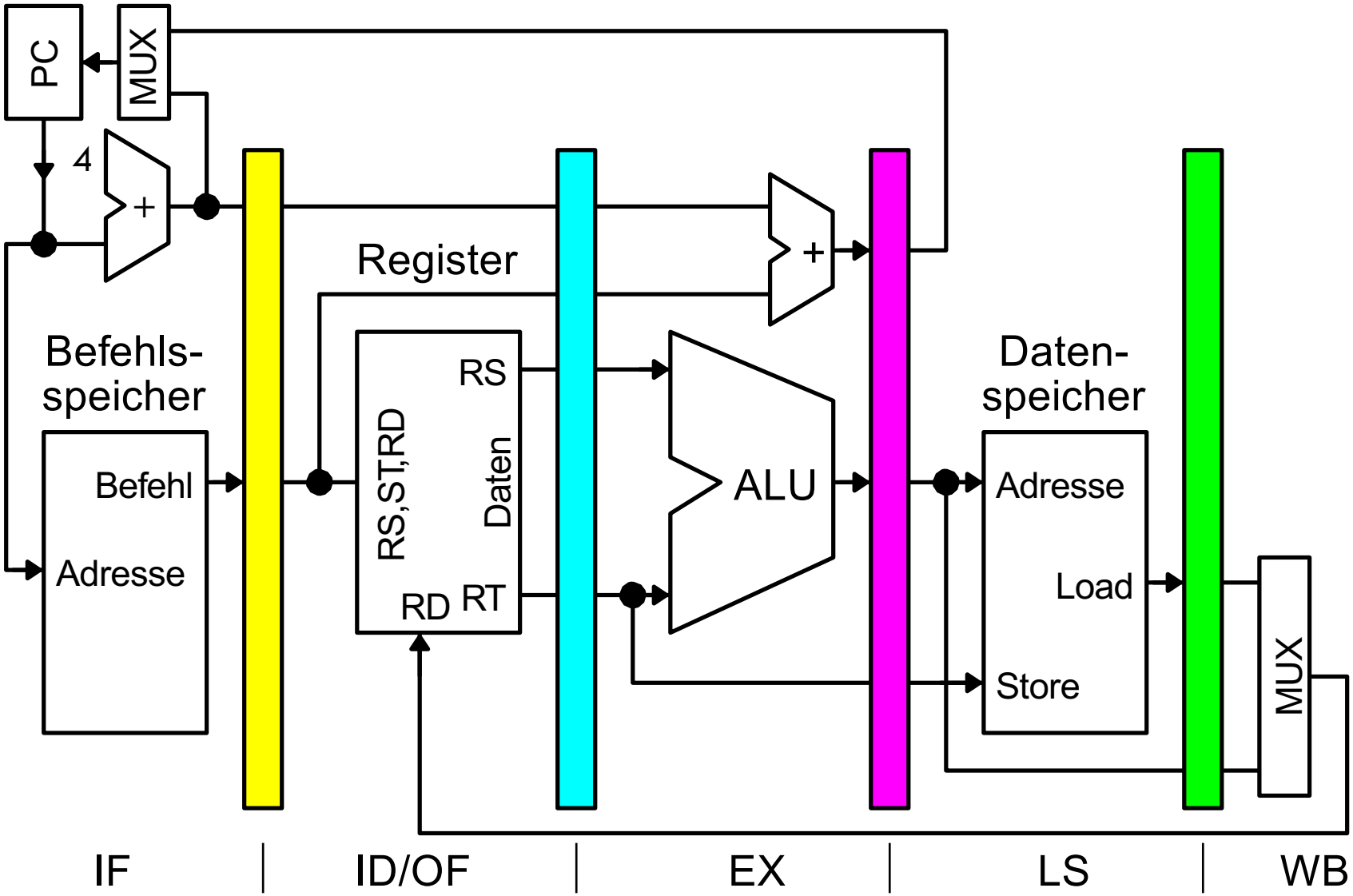
RISC-Pipelining

5-stufiger Befehlszyklus einer typischen RISC-Pipeline



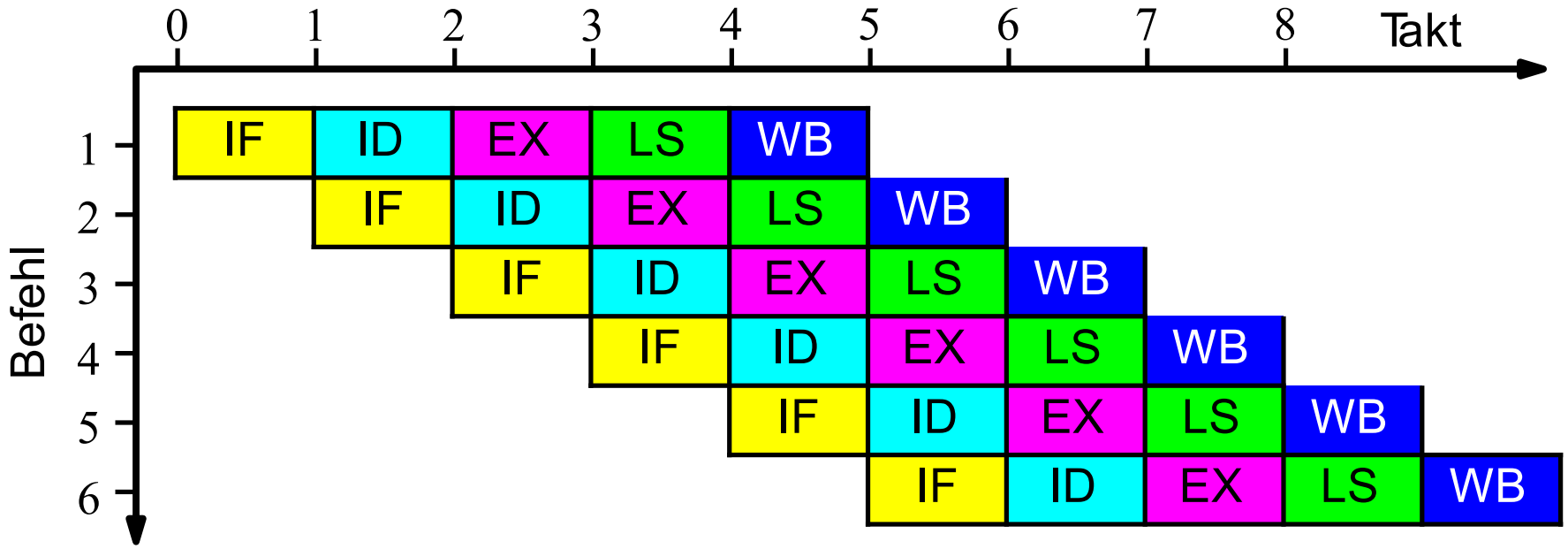
Entflechtung der Speicherzugriffe ⇒ Voraussetzung für Pipelining.

Pipeline-Architektur (stark vereinfacht)



Befehls-Pipelining

5-stufige RISC-Befehls-Pipeline



Die Befehle werden um einen Takt zeitversetzt überlappend gestartet.

Ab dem 5. Takt wird mit jedem Takt ein Befehl fertig gestellt. Die Pipeline ist gefüllt. Alle Pipeline-Stufen arbeiten parallel, jede an einem anderen Befehl.

⇒ Fließbandprinzip → Latenz: 5 Takte (Wartezeit)

Leistungsbetrachtung zum Pipelining (1)

N - Anzahl der Befehle

S - Anzahl der Pipeline-Stufen

T_C - Taktzykluszeit (Taktperiodendauer für eine Stufe)

Abarbeitungszeit ohne und mit Pipelining

seriell, S-Takt-Abarbeitung: TS_{EXE}

$$TS_{EXE} = N \cdot S \cdot T_C$$

parallel, S-Stufen-Pipeline: TP_{EXE}

$$TP_{EXE} = (S + N - 1) \cdot T_C$$

Leitungssteigerung durch Pipelining (Speed-Up SP)

$$SP = \frac{TS_{EXE}}{TP_{EXE}} = \frac{N \cdot S}{N + S - 1}$$

Leitungssteigerung pro Stufenzahl (Effizienz EF)

$$EF = \frac{SP}{S} = \frac{N}{N + S - 1}$$

Leistungsbetrachtung zum Pipelining (2)

Leistungssteigerung durch 5-stufige Pipeline ($S = 5$)

N	1	2	3	4	5	10	20	50	100	1000
SP	1	1.67	2.14	2.50	2.78	3.57	4.16	4.63	4.81	4.98

Grenzwerte für $N \to \infty$

$$\lim_{N\to\infty} SP = \lim_{N\to\infty} \frac{N \cdot S}{N + S - 1} = S$$

$$\lim_{N\to\infty} EF = \lim_{N\to\infty} \frac{N}{N + S - 1} = 1$$

$$\lim_{N\to\infty} CPI = \lim_{N\to\infty} \frac{TP_{EXE}}{N \cdot T_C} = \lim_{N\to\infty} \frac{N + S - 1}{N} = 1$$

⇒ Die Leistungssteigerung ist direkt von der Stufenzahl S abhängig.

⇒ Die Latenz der Pipeline entspricht der Stufenzahl S.

⇒ Viele einfache Stufen führen zu einer hohen Leistungssteigerung und gleichzeitig zu einer hohen möglichen Taktfrequenz.

Probleme, Konflikte beim Pipelining (1)

Pipeline-Konflikte (Hazards) bzw. Probleme

- Laden und Entladen der Pipeline führt zu zusätzlichen Latenzen.
- Stufenanzahl durch Granularität der Befehlsabarbeitung begrenzt.
- Minimale Stufen-Verzögerungszeit durch Pipeline-Register (setup) begrenzt.
- Ressourcenkonflikte, Strukturkonflikte (z.B. Speicherzugriffe).
- Datenkonflikte (Datenabhängigkeiten).
- Kontrollflusskonflikte (Programmverzweigungen).

Probleme, Konflikte beim Pipelining (2)

Wirkungen der Konflikte und Probleme

⇒ Pipeline wird nicht optimal ausgelastet, der Durchsatz sinkt.

⇒ Effektive Werte: $SP < S$, $EF < 1$ und $CPI > 1$.

⇒ Zusätzlicher Aufwand zur Konfliktvermeidung und Problembehandlung.

Strukturkonflikte (Structural Hazard)

Ursachen

- Mehrere Stufen wollen gleichzeitig auf eine Ressource zugreifen.
- Ressourcenzugriffe vorgelagerter Befehle sind noch nicht abgeschlossen.

Wirkung

⇒ Ressourcenzugriff nicht eindeutig möglich.

Vermeidung

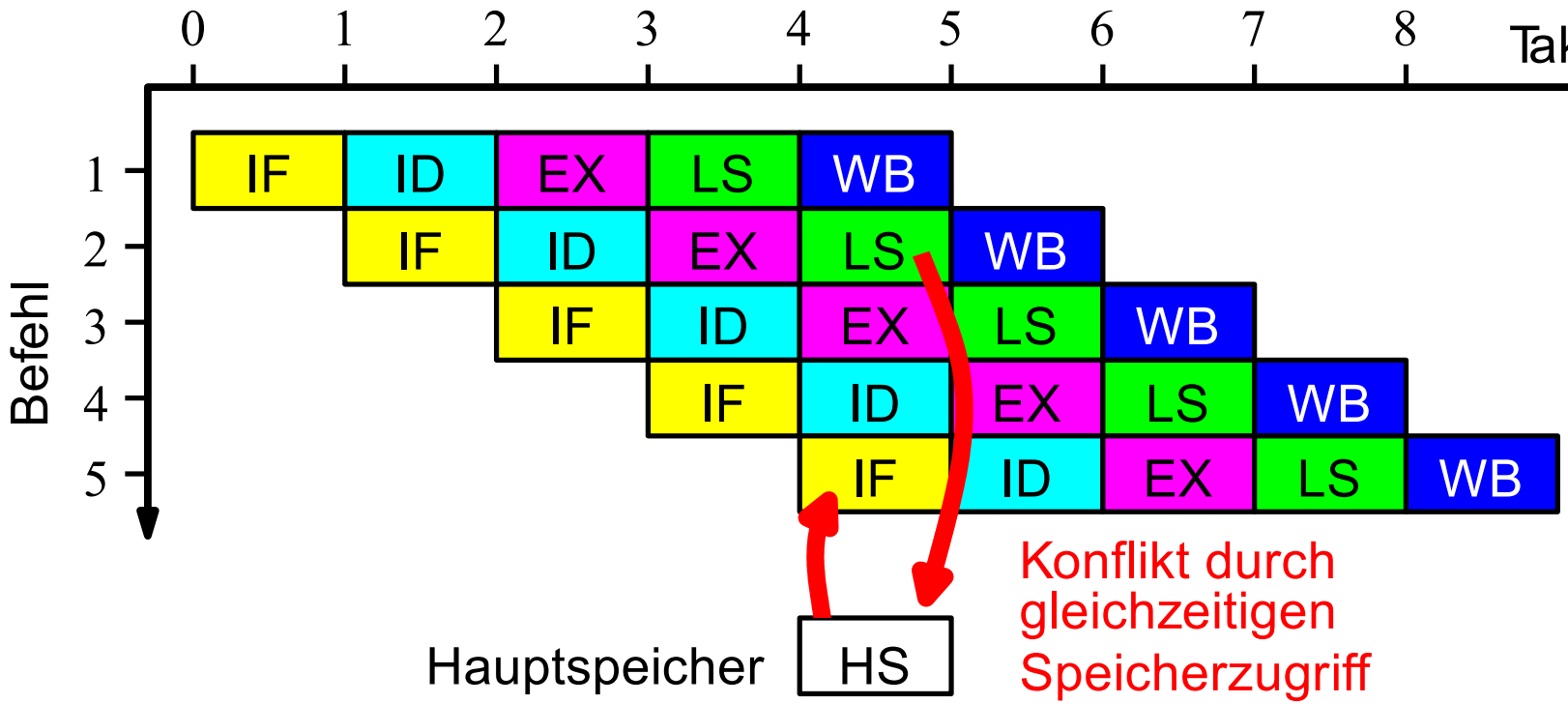
⇒ Harvard-Architektur, getrennte Busse und Caches.

⇒ Zusätzliche Funktionseinheiten, Prefetch Buffer (Instruction Queue).

⇒ Multi-Port Registerspeicher, Multiplexer-Netzwerke.

Beispiel Strukturkonflikte

Gleichzeitiger Hauptspeicherzugriff von der IF- und der LS-Stufe



⇒ Der 5. Befehl muss angehalten werden und mindestens einen Takt warten.

Datenkonflikte (Data Hazard)

Ursachen

- Im Befehl benötigte Registerinhalte sind vom Ergebnis eines vorgelagerten Befehls abhängig, der sich jedoch noch in der Pipeline befindet.
- In einer Stufe benötigte Daten stehen noch nicht zur Verfügung (z.B. nach Speicherzugriff).

Wirkung

⇒ Verarbeitung veralteter, nicht aktueller Daten.

Vermeidung

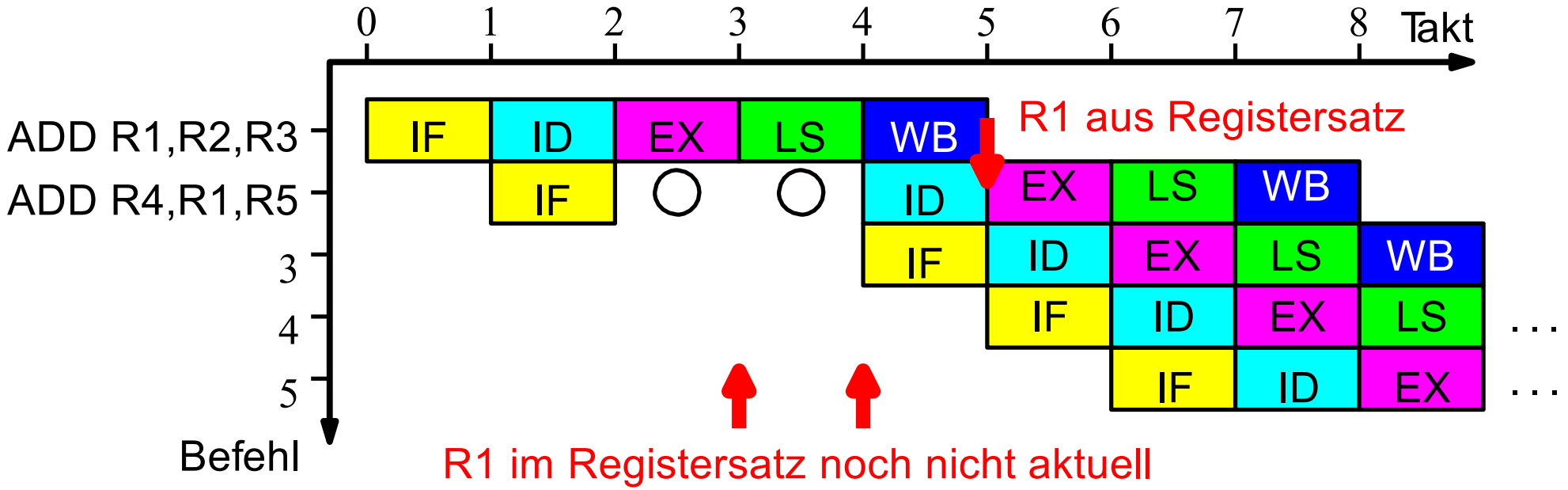
⇒ Anhalten der Pipeline, Einfügen von NOP-Befehlen

⇒ Umsortieren der Befehlsfolge, Out-of-Order Execution

⇒ Forwarding

Datenkonflikte – Read after Write (RAW)

Echte Datenabhängigkeit (Data Dependency)

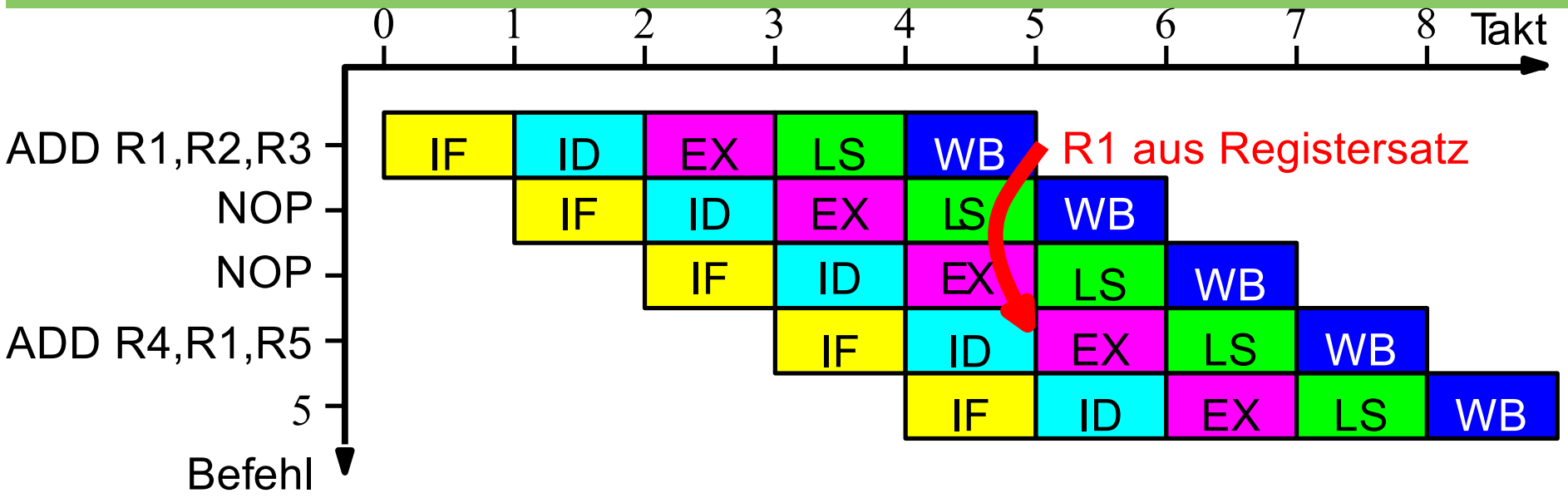


Der 2. Befehl muss für 2 Takte angehalten werden, bzw. 2 Takte warten.

Das Anhalten bzw. Warten der Befehle am Anfang der Pipeline (Pipeline Stall) erzeugt Lücken (Bubbles) in der Pipeline.

⇒ Die Pipeline ist nicht mehr optimal ausgelastet und der Durchsatz sinkt.

RAW-Konfliktlösung – Einfügen von NOP



Der RAW-Konflikt kann durch Einfügen von 2 NOP-Befehlen behoben werden.

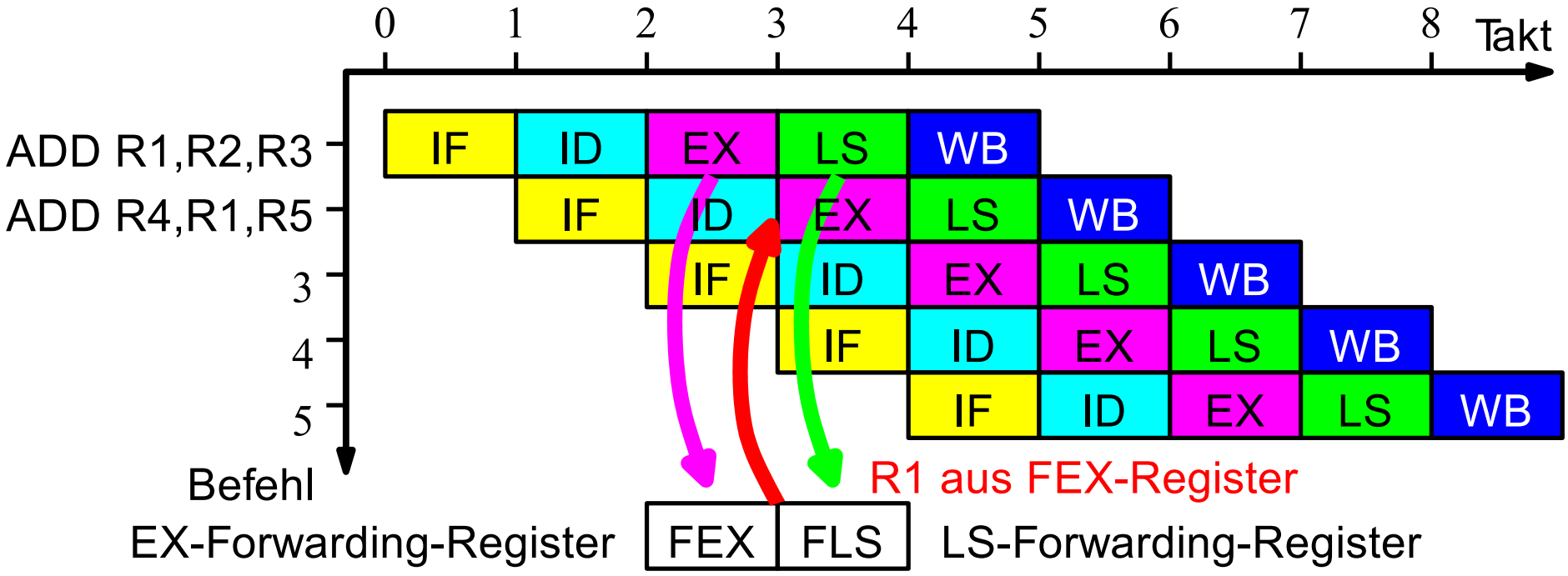
An die Stelle der NOP-Befehle können auch andere Befehle ohne Datenabhängigkeiten durch Umsortieren der Befehlsfolge (Vorziehen) eingefügt werden (Out-of-Order Execution).

Bei MIPS-Prozessoren kann als NOP-Befehl z.B. auch ADD R0,R0,R0 verwendet werden.

RAW-Konfliktlösung - Forwarding

Da das Ergebnis einer Operation bereits nach der EX-Stufe vorliegt, kann durch Forwarding der Pipeline-Register ein früherer Zugriff auf das Ergebnis erfolgen.

⇒ zusätzliche Hardware, Änderung der Pipeline-Steuerung, neue Datenpfade



⇒ Vermeidung von NOP-Befehlen, keine Konflikte.

Datenkonflikte – Write after Read (WAR)

Gegenabhängigkeit (Anti-Dependence)

Ursachen

- Im Befehl wird ein Registerinhalt überschrieben, auf den in einem vorhergehenden Befehl lesend zugegriffen wird.

Wirkung

⇒ Ein Konflikt tritt immer dann auf, wenn die aktuelle Schreiboperation schneller abgeschlossen ist als die vorhergehende Leseoperation.

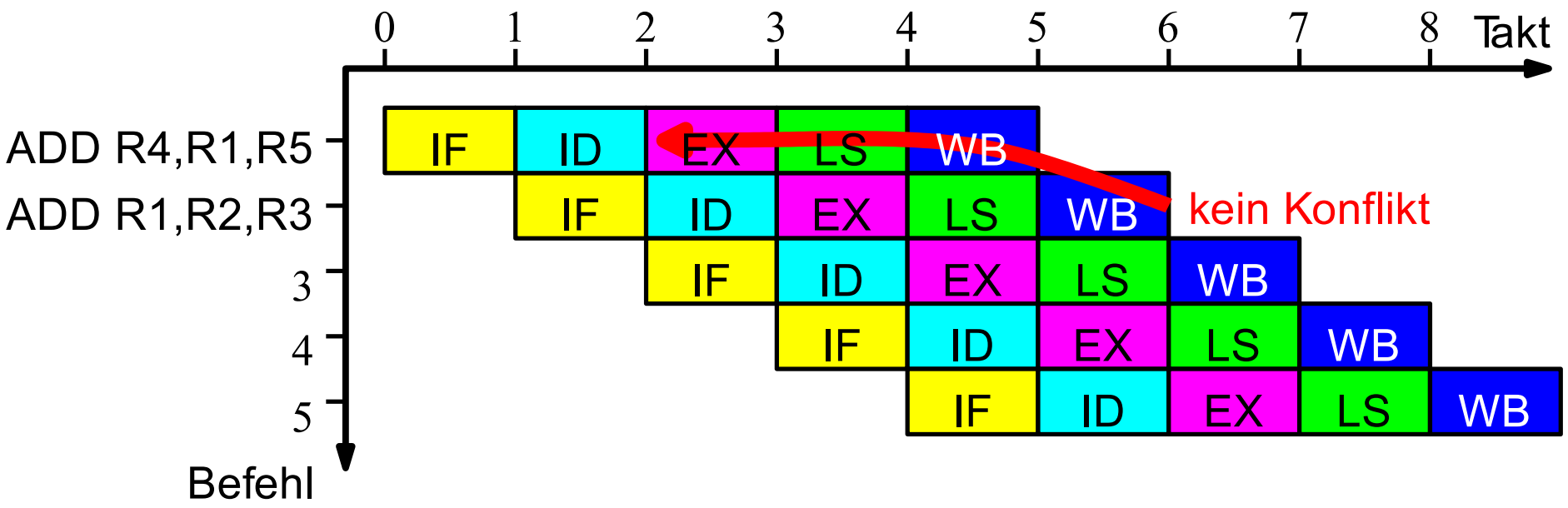
Vermeidung

⇒ Stellt bei der realen Pipeline kein Problem dar.

⇒ Keine Konfliktbehandlung erforderlich.

⇒ Beim Umsortieren der Befehlsfolge (z.B. für Lösung des RAW-Konflikt) zu beachten.

WAR-Konfliktlösung



Die Schreiboperation erfolgt 4 Takte nach der Leseoperation.

Es ist davon auszugehen, dass die Leseoperation vor der Schreiboperation beendet ist.

Beim Umsortieren (Vorziehen des 2. Befehls) entsteht hier jedoch ein WAR-Konflikt.

Datenkonflikte – Write after Write (WAW)

Gegenabhängigkeit (Anti-Dependence)

Ursachen

- Im Befehl wird ein Registerinhalt überschrieben, auf den in einem vorhergehenden Befehl ebenfalls geschrieben wird.

Wirkung

⇒ Ein Konflikt tritt immer dann auf, wenn die aktuelle Schreiboperation schneller abgeschlossen ist als die vorhergehende Schreiboperation.

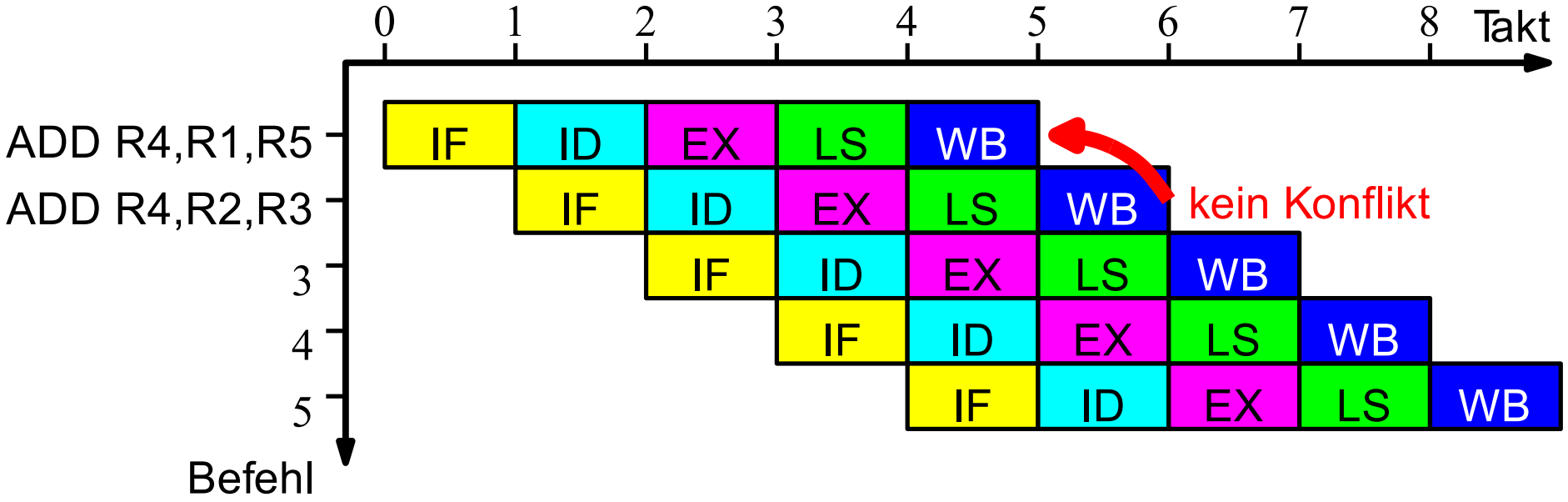
Vermeidung

⇒ Stellt bei der realen Pipeline kein Problem dar.

⇒ Keine Konfliktbehandlung erforderlich.

⇒ Beim Umsortieren der Befehlsfolge zu beachten.

WAW-Konfliktlösung



Die aktuelle Schreiboperation erfolgt 1 Takt nach der vorherigen Schreiboperation.

Es ist davon auszugehen, dass die vorherige Schreiboperation vor der aktuellen Schreiboperation beendet ist.

Steuerkonflikte (Control Hazard)

Ursachen

- Verzweigungsbefehle können den sequentiellen Befehlsablauf unterbrechen.
- Bei bedingten Verzweigungen steht erst relativ spät fest, ob verzweigt wird oder nicht. Das trifft auch auf das Sprungziel zu.

Wirkung

⇒ Befehle nach Verzweigungsbefehlen werden in der Pipeline abgearbeitet, obwohl sie eigentlich übersprungen werden sollten.

Vermeidung

⇒ Anhalten der Pipeline, Einfügen von NOP-Befehlen

⇒ Umsortieren der Befehlsfolge, Out-of-Order Execution

⇒ Sprungvorhersage, spekulative Befehlsabarbeitung.

Beispiel: Bedingte Verzweigung

Zweistufige Realisierung

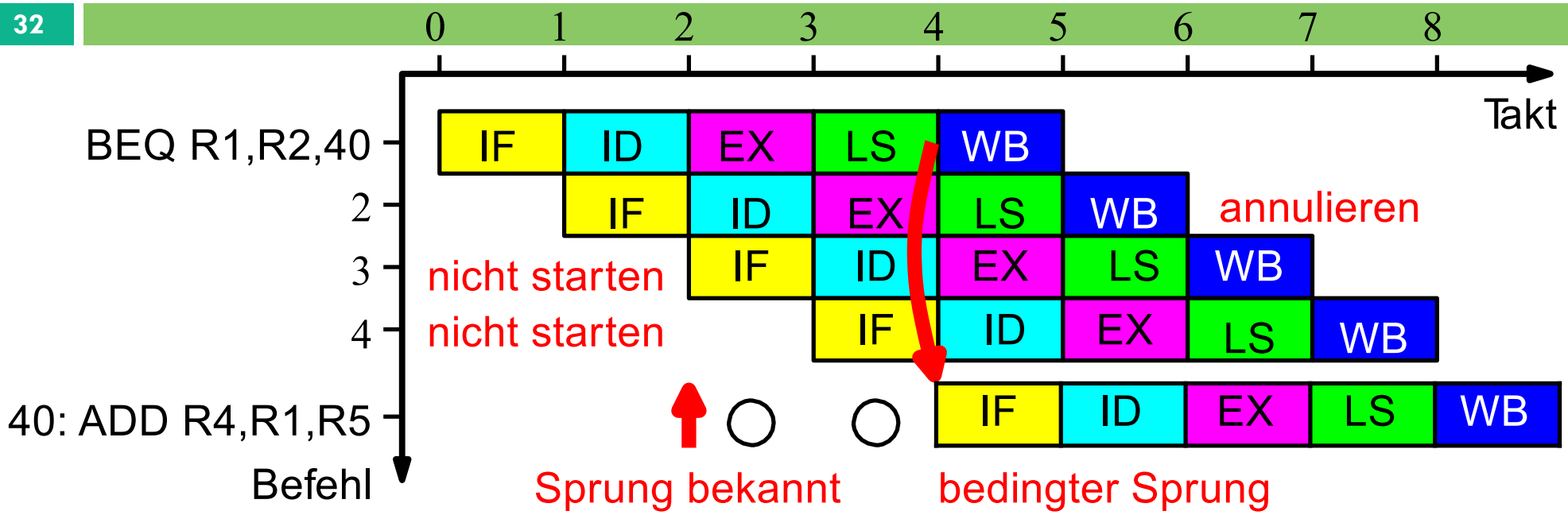
1. Bestimmung des Bedingungskode (Condition Code, cc) nach einer arithmetischen Operation.
2. Bestimmung, Berechnung der Zieladresse in Abhängigkeit vom letzten Bedingungskode und setzen des Befehlszählers.

Es gibt auch die Möglichkeit der Kombination von arithmetischer Operation (Bedingungscode) mit der Verzweigungsoperation in einem Befehl.

Randbedingungen

- Der Bedingungskode liegt erst nach der EX-Stufe vor.
- Ob es sich um einen Verzweigungsbefehl handelt, wird erst in der ID-Stufe festgestellt.
- Der Befehlszähler (Program Counter) wird erst in der LS-Stufe mit der neu berechneten Verzweigungsadresse beschrieben.

Steuerkonflikte – Bedingte Verzweigung

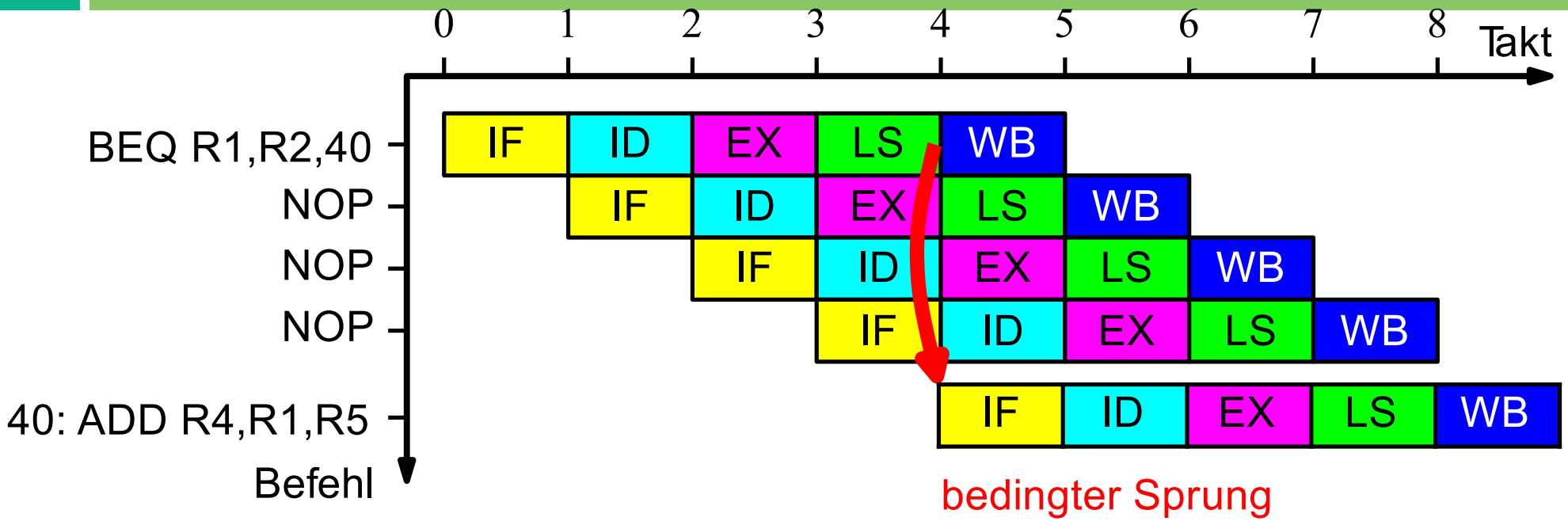


Da der Sprungbefehl erst in der ID-Stufe bekannt wird, wird der 2. Befehl bereits parallel gestartet. Die Verzweigungsadresse ist erst in der LS-Stufe bekannt.

- Der bereits gestartete 2. Befehl ist zu annulieren (→ Branch Delay Slot).
- Der 3. und der 4. Befehl sind nicht zu starten. Die Pipeline muss 2 Takte angehalten werden (stall).

⇒ Die Pipeline ist nicht mehr optimal ausgelastet und der Durchsatz sinkt.

Bedingte Verzweigung – Konfliktlösung mit NOP



Der Steuerkonflikt bei bedingter Verzweigung kann durch Einfügen von 3 NOP-Befehlen direkt nach dem Verzweigungsbefehl behoben werden.

An die Stelle der NOP-Befehle können auch andere verzweigungsunabhängige Befehle durch Umsortieren der Befehlsfolge eingefügt werden.

Auf den Verzweigungsbefehl kann auch ein fester **Branch Delay Slot** folgen, der mit verzweigungsunabhängigen Befehlen (auch NOP) gefüllt werden kann.

Vermeidung, Reduktion von Steuerkonflikte

Da Steuerkonflikte den Durchsatz und damit die Leistungsfähigkeit einer Pipeline wesentlich beeinflussen, ist eine Vermeidung bzw. Reduktion von Steuerkonflikten besonders wichtig.

Maßnahmen zur Reduktion von Steuerkonflikten

1. Allgemeine Vermeidung von Verzweigungsbefehlen:
   - Aufrollen von Schleifen (Loop Unrolling).
   - Direktes Einfügen von Unterprogrammen.
   - Spezialbefehle, die in Abhängigkeit einer Bedingung ausgeführt werden.
2. Änderungen der Architektur:
   - Vorverlagerung der Sprungentscheidung (z.B. in die EX-Stufe).
   - Vorverlagerung der Sprungadressenberechnung (z.B. in die ID-Stufe). → Look-Ahead-Resolution.
   - Sonderbehandlung unbedingter Verzweigungen.

Vermeidung, Reduktion von Steuerkonflikte

3. Spekulative Befehlsausführung:
   - Spekulative Befehlsabarbeitung mit einer Vorzugsverzweigung.
   - Spekulative Befehlsabarbeitung beider Verzweigungsrichtungen.
4. Verzweigungsbefehle mit Branch Delay Slot:
   - Befehlsfolge umsortieren, einfügen verzweigungsunabhängiger Befehle.
   - Spekulatives Einfügen der Befehle der Vorzugsverzweigung.
5. Verzweigungsvorhersage (Branch Prediction):
   - Statische Verzweigungsvorhersage (Verzweigungsstatistik, Compiler).
   - Dynamische Verzweigungsvorhersage: Verzweigungs-Mustererkennung
     Verzweigungssammlung (Branch-History-Table),
     Verzweigungsmuster (Pattern-History-Table),
     Verzweigungsadressen (Branch-Target-Buffer, Prediction-Cache).

Datenpfad-Pipelining

Pipelining wird ebenfalls zur Parallelisierung von Datenpfaden eingesetzt (Ausführungs-Pipeline).

Stufen einer Gleitkomma-Pipeline

Gleitkomma-Addition

1. Exponenten-Subtrahierer
2. Exponenten-Angleichung
3. Mantissen-Addierer
4. Normalisierer

Gleitkomma-Multiplikation

1. Exponenten-Addierer
2. Mantissen-Multiplizierer
3. Rundung
4. Normalisierer

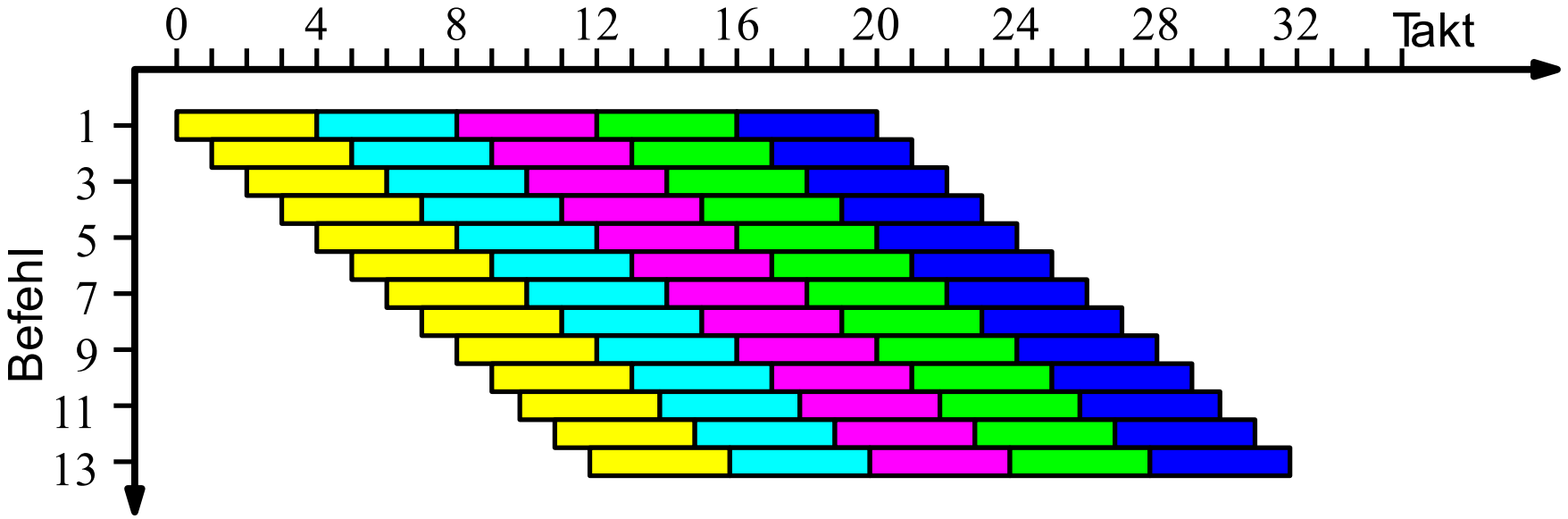
Diese Stufen werden typisch in einer Gleitkomma-Pipeline zusammengefasst.

Super-Pipelining

Weitere Unterteilung der einzelnen Stufen einer Pipeline mit den Zielen:

- Feinere Abstimmung der Pipeline, bessere Auslastung der Stufen.
- Weitere Erhöhung der Taktfrequenz und des Pipeline-Durchsatzes.

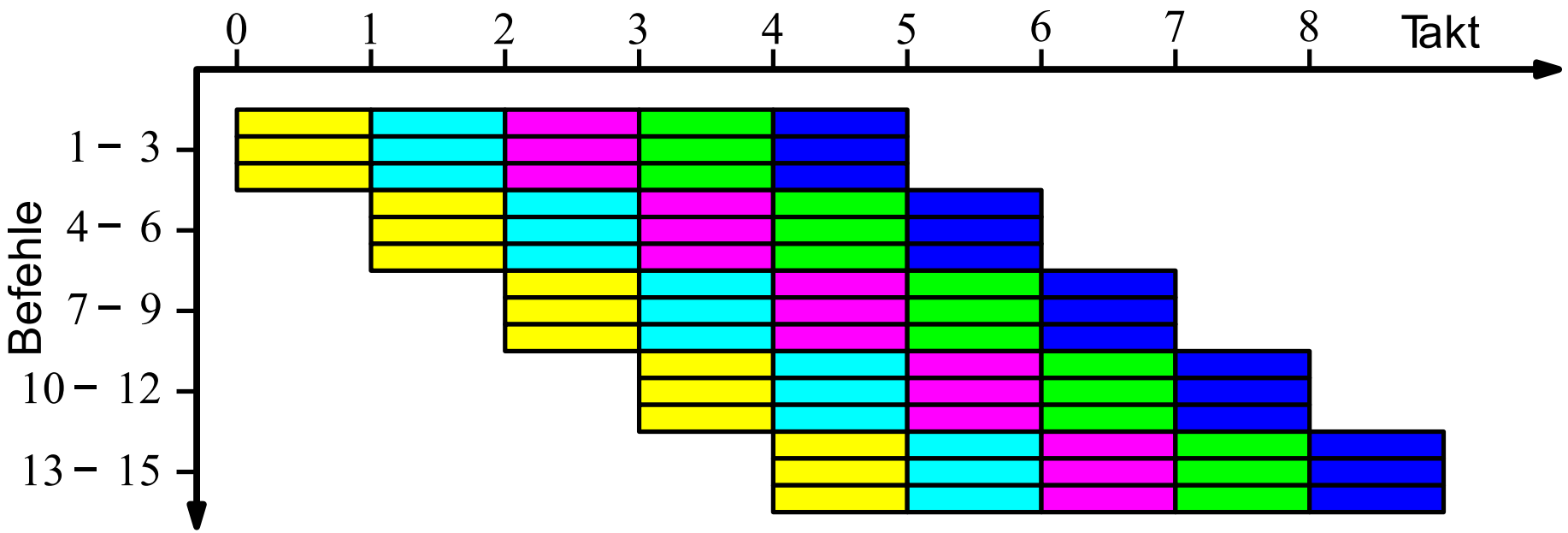
20-stufige RISC-Pipeline



Superskalar-Pipelining

- Innerhalb der Pipeline werden mehrere Verarbeitungseinheiten parallel angeordnet (mehrere EX-Einheiten).
- Die einzelnen Verarbeitungseinheiten sind typisch auf bestimmte Gebiete spezialisiert (INT, FP, LS, . . .)
- Die Hardware verteilt die abzuarbeitenden Befehle entsprechend auf die einzelnen Verarbeitungseinheiten.
- Der Befehlsstrom kann so innerhalb der Pipeline auf mehrere Verarbeitungseinheiten verteilt, parallelisiert werden.
- Der Parallelisierungsgrad durch ILP kann um die Anzahl der parallelen Verarbeitungseinheiten erhöht werden (wird praktisch nicht erreicht).
- Die Anzahl der Pipeline-Konflikte nimmt aufgrund der parallelen Abarbeitung mehrerer Befehle innerhalb der EX-Phase stark zu.

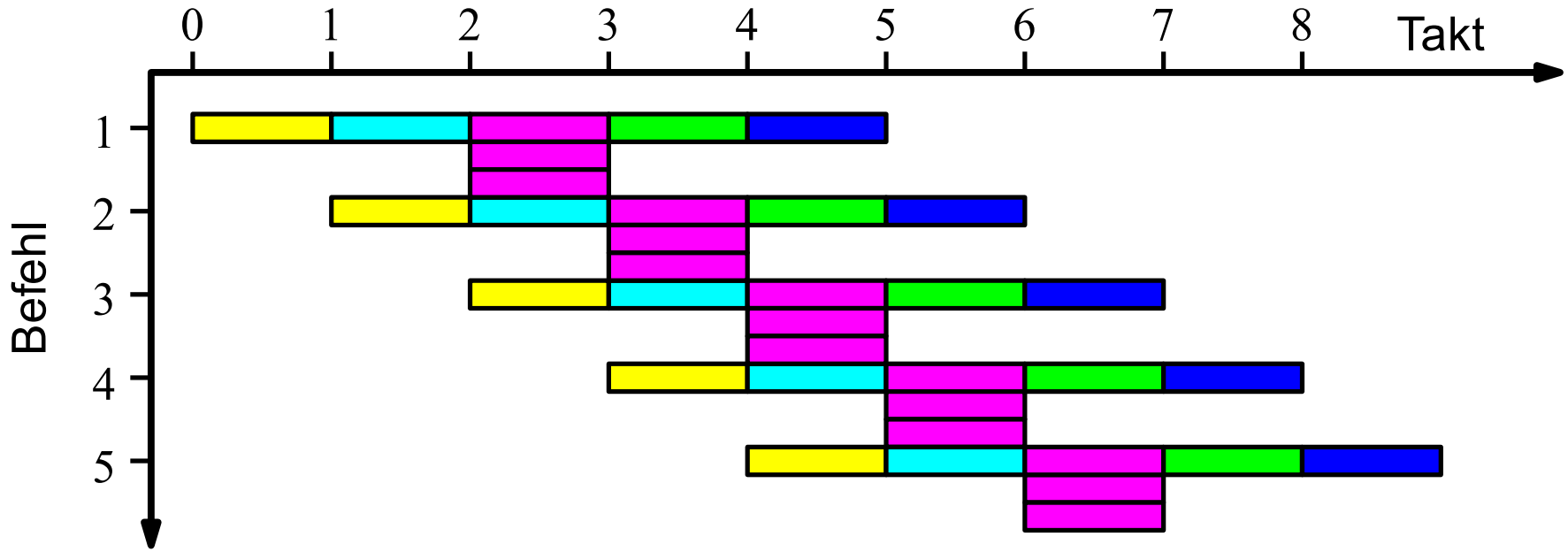
5-stufige RISC-Pipeline, 3-fach superskalar



$$\lim_{N\to\infty} CPI = \lim_{N\to\infty} \frac{TP_{EXE}}{N \cdot T_C} = \lim_{N\to\infty} \frac{\frac{1}{3}N + S - 1}{N} = 0.33$$

VLIW-Architekturen

5-stufige VLIW-Pipeline, 3 parallele Verarbeitungseinheiten



$\lim_{N \to \infty} CPI = 1$ wie bei „normaler“ Pipeline

dafür mehrere (ALU-)Operationen pro Befehl