Prozessorarchitektur

Klassifikationsmöglichkeiten

M. Schölzel

Gliederung

- Beispielprozessoren
- Klassifikation nach Befehlssatzarchitekturen
- Klassifikation nach Arten der Parallelität

Gliederung

- Beispielprozessoren
- Klassifikation nach Befehlssatzarchitekturen
- Klassifikation nach Arten der Parallelität

Beispielprozessoren

- Intel IA32-Architektur am Beispiel des Core i7
- ARM Cortex-A9
- Mikrocontroller Atmel ATmega168

Core i7

- Desktopprozessor von Intel

- Eingeführt 2008 mit
  - 4 Prozessorkernen, 731 Mio Transistoren, 45nm Technologie, 3.2 GHz Takt
  - 64-Bit-Architektur
  - Leistungsaufnahme zwischen 17 und 150 W (je nach Modell)

- Befehlssatz ist abwärtskompatibel mit den Vorgängern:
  - Pentium 4, Pentium III, Pentium Pro, Pentium II, Pentium, 80486, 80386

- Jeder Kern
  - ist 4-fach superskalar (kann gleichzeitig vier Befehle starten)
  - unterstützt Hyperthreading (… führt Befehle verschiedener Threads aus)
  - hat 32 KB L1-Datencache und 32 KB L1-Befehlscache und
  - 256 KB L2-Cache für Daten und Programmcode

- Zwischen 4 und 15 MB L3-Cache für alle Kerne

[http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html]

Kurze Historie der Intel-Architekturen

Jahr	Prozessor	Features
1978	8088/8086	16-Bit Prozessor mit Segmentierung , 1 MB Speicher adressierbar
1982	80286	Protected Mode (erlaubt 24-Bit Adressen zum adressieren von 16 MB Speicher, virtuelles Speichermanagement mit Segmenten und Privilegebenen)
1985	80386	32-Bit Prozessor (Operanden und Adressen haben 32 Bit, 4 GB Speicher adressierbar), Segmentierung und Paging für virtuelles Speichermanagement
1989	80486	Dekodierung und Ausführung von Operationen in 5-stufiger Pipeline, 8 KB-On-Chip Cache, integrierte FPU
1993	Pentium	Superskalar (u- und v-Pipeline), separierter Daten- und Code-Cache - jeweils 8KB, Branch-Prediction, interne Datenbusse mit 128- und 256-Bit
1995-1999	P6	Pentium Pro mit 3-fach superskalarem Datenpfad, out-of-order Befehlsausführung, spekulative Befehlsausführung Pentium II mit MMX-Technology, Low-Power Zustände Pentium II Xeon, Intel Celeron, Pentium III
2000-2006	Pentium 4	NetBurst Microarchitektur, 64-Bit Architektur mit sehr hohen Taktraten (3,6 GHz) durch bis zu 31-stufige Pipeline
2001-2007	Intel Xeon	Hyperthreading, basiert noch auf NetBurst-Architektur
2006	Core 2	Core-Mikroarchitektur (basierend auf P6-Architektur) mit 14-stufiger Pipeline
2008	Intel Core i7	Nehalem-Architektur basierend auf Core-Architektur mit Turbo Boost, Hyperthreading, integriertem Speichercontroller, …

ARM Cortex-A9

- General-Purpose Prozessor mit bis zu 4 ARMv7 Prozessorkernen:
  - 2008 auf den Markt gebracht
  - Gefertigt in 45 nm Technologie mit 1 GHz Taktfrequenz
- Geringer Stromverbrauch:
  - Einsatz in mobilen Systemen (Smartphones, Tablets, Unterhaltungselektronik im Automobil)
  - Nur 650 mW bei voller Leistung; nur 100µW im Schlafmodus
- Jeder Kern:
  - 2-fach superskalar, out-of-order Befehlsausführung
  - 16 bis 64 KB L1-Cache für Daten und 16 bis 64 KB L1-Cache für Programmcode
- 1 MB L2 Cache für Befehle und Daten

[http://www.arm.com/cortex-a9.php]

Atmel ATmega168

- 8-Bit Mikrocontroller von Atmel für eingebettete Systeme:
  - 16 KB Flash – hauptsächlich für Programme
  - 20 MHz Taktfrequenz
  - 512 Byte EEPROM
  - 1 KB SRAM
- Ein-Chip-Lösung
- Verwendung z.B. in Haushaltsgeräten (Kosten ca. 2.50 US$ pro Stück bei > 1000 Stück)
- 28-Pins am Gehäuse – hauptsächlich zum Anschluss von Peripherie (23 digitale E/A-Anschlüsse)

[http://www.atmel.com/devices/atmega168.aspx]

Gliederung

- Beispielprozessoren
- Klassifikation nach Befehlssatzarchitekturen
- Klassifikation nach Arten der Parallelität

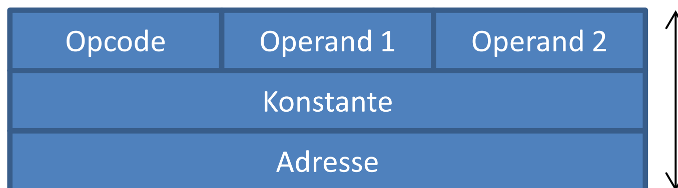
Befehlssatzarchitektur

- Ein Befehl besteht aus:

Operation (add, sub, mul,...)	Operand 1 (r3, [bx+5], 678,...)	...	Operand *n* (r3, [bx+5], 678,...)
	Registeroperand Speicheroperand Konstante		Registeroperand Speicheroperand Konstante

Assemblerbefehl ist textuelle Repräsentation eines Befehls

- Befehlssatzarchitektur beschreibt:
  - Unterstützte Operationen
  - Zulässige Datenformate für die Operanden
  - Anzahl, Art und Verwendung der Operanden in einem Befehl
    - Registerarchitektur
    - Adressierungsmöglichkeiten des Speichers
  - Kodierung der Befehle



Anzahl der Datenworte zur Kodierung eines Befehls

Maschinencode ist Binärcodierung eines Befehls

Gliederung Befehlssatzarchitektur

- Unterstützte Operationen
- Zulässige Datenformate für die Operanden
- Anzahl, Art und Verwendung der Operanden in einem Befehl
  - Registerarchitektur
  - Adressierungsmöglichkeiten des Speichers
- Kodierung der Befehle

Befehlsgruppen

- Befehle können in verschiedene Gruppen eingeteilt werden
  - Transportbefehle
    - Kopieren Daten von einem Speicherort zu einem anderen (Speicher <-> Register, Register -> Register, Speicher -> Speicher)
  - Dyadische Operationen
    - Verknüpfen zwei Operanden zu einem Ergebnis (arithmetische und logische Operationen)
  - Monadische Operationen
    - Ergebnis wird aus nur einem Operanden berechnet
  - Bedingte Verzweigungsbefehle
    - In Abhängigkeit des Wertes eines Operanden wird an eine angegebene Zieladresse verzweigt oder der nachfolgende Befehl verarbeitet
  - Unbedingte Sprünge und Funktionsaufrufe
    - Verzweigen an eine gegebene Programmadresse und sicher ggf. aktuelle Programmadresse
  - Ein-/Ausgabe
    - Zum Zugriff auf die Peripherie

- Vollständige Übersicht liefert die Befehlssatzreferenz zu einem Prozessor (Instruction Set Manual)

Ausgewählte Befehle der IA32

- Operationen haben maximal 2 Operanden
- Höchstens einer davon ist ein Operand im Speicher
- Komplexe Adressierungen für Speicheroperanden werden unterstützt
- Weitere Befehlsgruppen:
  - BCD-Arithmetik: DAA (Korrektur nach Addition)
  - Test-/Vergleichsbefehle: CMP src1, src2
  - Programmsteuerung: JMP addr ; …
  - Zeichenfolgenbefehle: MOVS; …
  - Flagbefehle: STC, CLC, …
  - Misc: ENTER size; LEAVE; …

Arithmetikbefehle	Beschreibung
Add dst,src	Addiere src zu dst
Adc dst,src	Addiere src zu dst mit Übertrag
Sub dst,src	Subtrahiere src von dst
Sbb dst,src	Subtrahiere src von dst mit Übertrag
Mul Src	Multipliziere EAX mit src (vorzeichenlos)
iMul Src	Multipliziere EAX mit src (vorzeichenbehaftet)
Div Src	Dividiere EDX:EAX durch src
iDiv Src	Dividiere EDX:EAX durch src (vorzeichenbehaftet)
…	…

Transportbefehle	Beschreibung
Mov dst,src	Kopiere src nach dst
Push src	Lege src auf Keller
Pop dst	Hole ein Datum vom Keller
Xchg dst1, dst2	Vertausche dst1 und dst2
…	…

Befehlssatzreferenz

Ausgewählte Befehle des ARM

- Operationen haben bis zu 4 Operanden
- Keiner davon ist ein Operand im Speicher (Ausnahme Lade-/Speicherbefehle)
- Weitere Befehlsgruppen:
  - Schiebe- und Rotationsbefehle: LSL dst, src1, src2Imm; …
  - Logikbefehle: AND dst, src1, src2Imm
  - Ablaufsteuerung: Bcc imm (Springe zu PC + imm); …

Arithmetikbefehle	Beschreibung
Add dst,src1,src2Imm	Addieren
Adc dst,src1,src2Imm	Addiere mit Übertrag
Sub dst,src1,src2Imm	Subtrahiere src von dst
Sbb dst,src1,src2Imm	Subtrahiere src von dst mit Übertrag
Mul dst, src1, src2	Multiplizieren
Mla dst, src1, src2, src3	Multipliziere EAX mit src (vorzeichenbehaftet)
Umull dst1, dst2, src1, src2	Multipliziere vorzeichenlos lang
Smull dst1, dst2, src1, src2	Multipliziere vorzeichenbehaftet lang
…	…

Kein Divisionsbefehl

Lade-/Speicherbefehle	Beschreibung
Ldrb dst, addr	Lade vorzeichenloses Byte aus Speicher
Ldrh dst, addr	Lade vorzeichenloses Halbwort aus Speicher
Ldr dst, addr	Lade vorzeichenloses Wort aus Speicher
Strb dst, src	
…	…

Befehlssatzreferenz

Ausgewählte Befehle des AVR

- Operationen haben maximal 2 Operanden
- In der Regel Registeroperanden
- Lade-/Speicherbefehle für Speicherzugriffe

Arithmetikbefehle	Beschreibung
Add dst,src	Addieren
Adc dst,src	Addiere mit Übertrag
Sub dst,src	Subtrahiere src von dst
Subi dst, imm	Subtrahiere src von dst
Mul dst, src	Multipliziere vorzeichenlos
Muls dst, src	Multipliziere vorzeichenbehaftet
…	…

Lade-/Speicherbefehle	Beschreibung
LDS dst,addr	Lade Byte aus Speicher von addr
STS addr,src	Schreibe Byte in Speicher an Adresse addr
…	…

Befehlssatzreferenz

Gliederung Befehlssatzarchitektur

- Unterstützte Operationen
- Zulässige Datenformate für die Operanden
- Anzahl, Art und Verwendung der Operanden in einem Befehl
  - Registerarchitektur
  - Adressierungsmöglichkeiten des Speichers
- Kodierung der Befehle

Datentypen Core i7

Typ	8 Bit	16 Bit	32 Bit	64 Bit
Vorzeichenbehaftete Ganzzahl	x	x	x	x
Vorzeichenlose Ganzzahl	x	x	x	x
Binärcodierte Dezimalzahl (BCD)	x			
Gleitkommazahlen			x	x

- Darstellung negativer Ganzzahlen im Zweierkomplement:
  - Keine Unterstützung für Addition/Subtraktion erforderlich
  - Unterstützung für Multipliktion/Division vorhanden:
    - Grund: 0xFFFF x 0xFFFF = 0xFFFE0001 (-1 x -1 = -131071)
- BCD-Darstellung: Kodierung einer Dezimalziffer pro Digit
  - Operationen zur Konvertierung des Ergebnisses in eine gültige BCD-Zahl
- Darstellung von Gleitkommazahlen nach IEEE 754 Standard

Datentypen ARM-A9

Typ	8 Bit	16 Bit	32 Bit	64 Bit
Vorzeichenbehaftete Ganzzahl	x	x	x	
Vorzeichenlose Ganzzahl	x	x	x	
Binärcodierte Dezimalzahl (BCD)				
Gleitkommazahlen			x	x

- Darstellung negativer Ganzzahlen im Zweierkomplement:
  - Keine Unterstützung für Addition/Subtraktion erforderlich
  - Unterstützung für Multipliktion vorhanden
- Darstellung von Gleitkommazahlen nach IEEE 754 Standard

Datentypen AVR Atmega168

Typ	8 Bit	16 Bit	32 Bit	64 Bit
Vorzeichenbehaftete Ganzzahl	x			
Vorzeichenlose Ganzzahl	x	x		
Binärcodierte Dezimalzahl (BCD)				
Gleitkommazahlen				

- Fast alle Operationen arbeiten nur auf 8-Bit Daten
- Ausnahme:
  - 16-Bit Adressen in den Registerpaaren X = R26/R27, Y = R28/R29, Z = R30/R31
  - Unterstützung der Increment-Funktion für 16-Bit Zeiger in diesen Registerpaaren

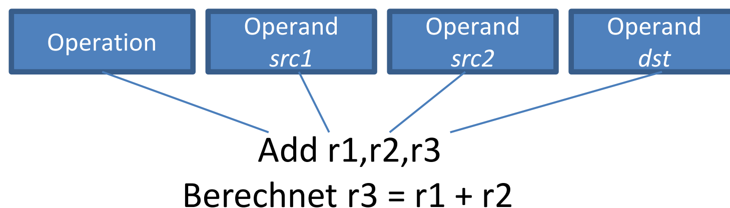
Gliederung Befehlssatzarchitektur

- Unterstützte Operationen
- Zulässige Datenformate für die Operanden
- Anzahl, Art und Verwendung der Operanden in einem Befehl
  - Registerarchitektur
  - Adressierungsmöglichkeiten des Speichers
- Kodierung der Befehle

Anzahl der Operanden

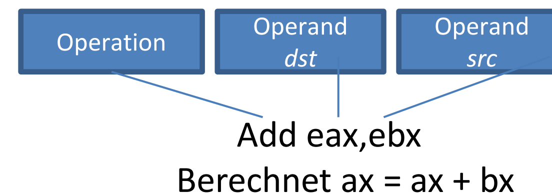
3-Adress-Code-Befehlsformat

- In einer Operation sind separat kodiert:
  - Zwei Quelloperanden (*src1*, *src2*)
  - Ein Zieloperand (*dst*)
- Vorteil:
  - Werte beider Quelloperanden können erhalten bleiben
- Nachteil:
  - Platzbedarf für Kodierung
- Beispiel: ARM-A9



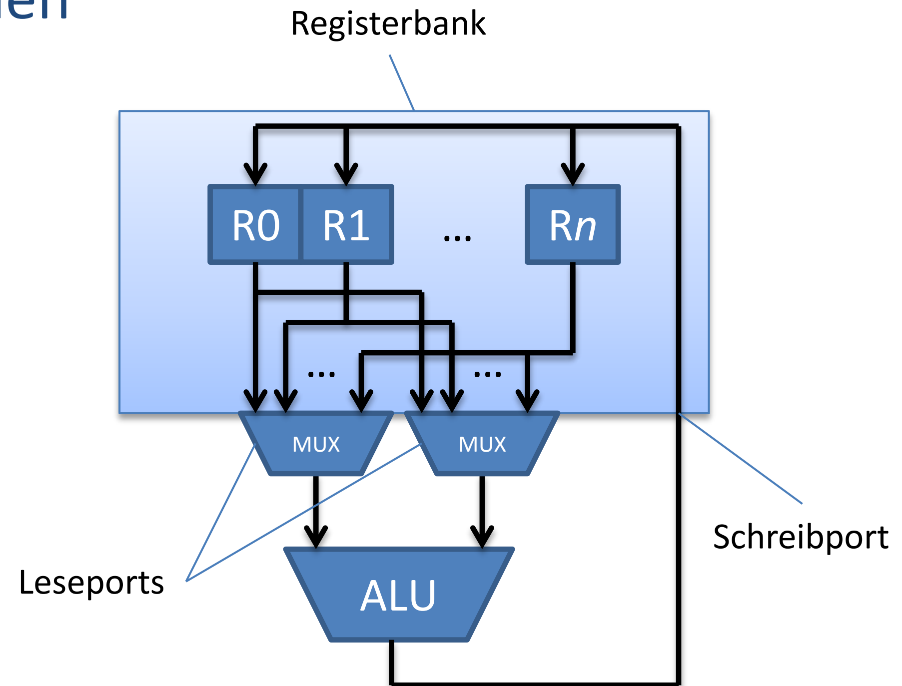
2-Adress-Code-Befehlsformat

- In einer Operation sind kodiert:
  - Zwei Quelloperanden (*src*, *dst*)
  - Ein Quelloperand ist gleichzeitig Zieloperand (*dst*)
- Vorteil:
  - Geringer Platzbedarf bei Kodierung
- Nachteil:
  - Ein Quelloperand muss immer überschrieben werden
  - Kann zusätzliche Kopieroperationen zum Retten eines Wertes erfordern
- Beispiel: IA32, ATmega



Universalregisterarchitektur

- Registerbank mit *n* Registern
- Register der Registerbank können in gleichartiger Weise in den Befehlen verwendet werden
- Einfache Kodierung möglich
- Beispiel: ARM-A9, MIPS



Registersatz ARM-Cortex-A9

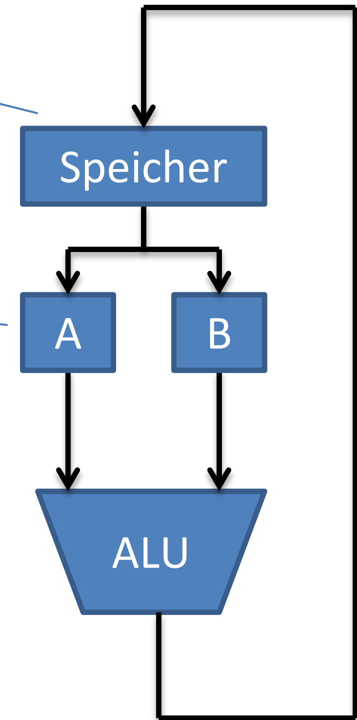
Register	Funktion
R0 – R3	Nimmt Parameter für die aufzurufende Funktion auf*
R4 – R11	Nimmt lokale Variablen für die aktuelle Prozedur auf*
R12	Prozedurübergreifendes Aufrufregister (für weit entfernte Prozeduren)
R13	Kellerzeiger*
R14	Link-Register (Rücksprungadresse bei Funktionsaufrufen) *
R15	Programmzähler

*Compilerspezifische Konventionen

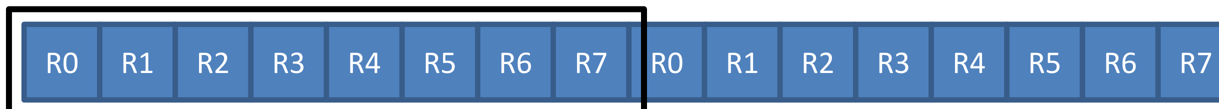
- Typischer Universalregistersatz:
  - 16 allgemeine 32-Bit-Integer-Register
  - 32 32-Bit-Gleitkommaregister (wenn FP-Coprozessor unterstützt wird
  - PC gehört zu den Universalregistern

Memory-Mapped-Register

- Register sind im Speicher abgebildet (schneller Speicher erforderlich)
- Vorteile:
  - Große Anzahl an Registern ist realisierbar ohne große Multiplexer für Lesezugriff
  - Vereinfachung des Aufbaus des Rechenwerks
- Nachteile:
  - Lesen zweier Operanden erfordert zwei Takte und Zwischenspeichern der Werte im Prozessor (oder Verwendung eines Dual-Ported-Memories)
  - Große Anzahl an Registern erfordert viel Speicherplatz für die Kodierung der Registeradresse
  - Vermeidbar durch Verwendung mehrerer Registerbänke:
    - Unterteilung aller Register in Registerbänke zu *k* Registern
    - Auswahl einer Registerbank z.B. über Statusregister
- Beispiel für zwei Registerbänke mit je 8 Registern:



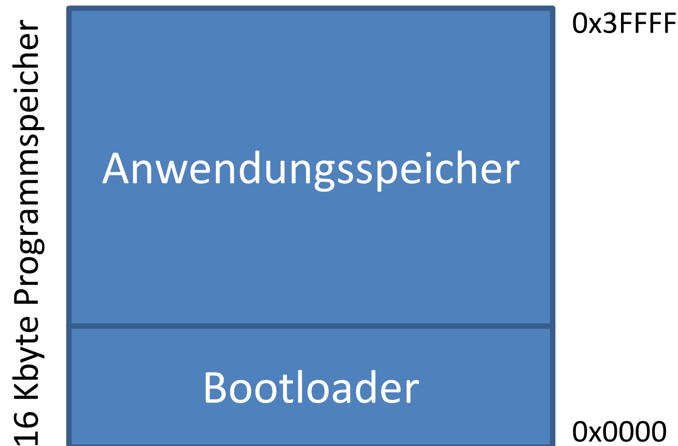
Speicheradresse 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15



Aktuelle Registerbank

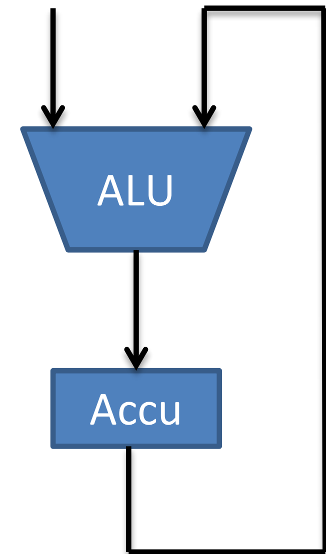
Registersatz AVR ATmege168

- Registersatz
  - 32 8-Bit-Register (R0 – R31)
  - Registersatz ist im Speicher abgebildet (R*x* = MEM[*x*])
  - X = RR26/27, Y = R28/R29, Z = R30/R31
- Speicherorganisation:
  - Daten- und Programmspeicher mit separatem Adressraum (Harvard-Architektur)



Akkumulatorarchitektur

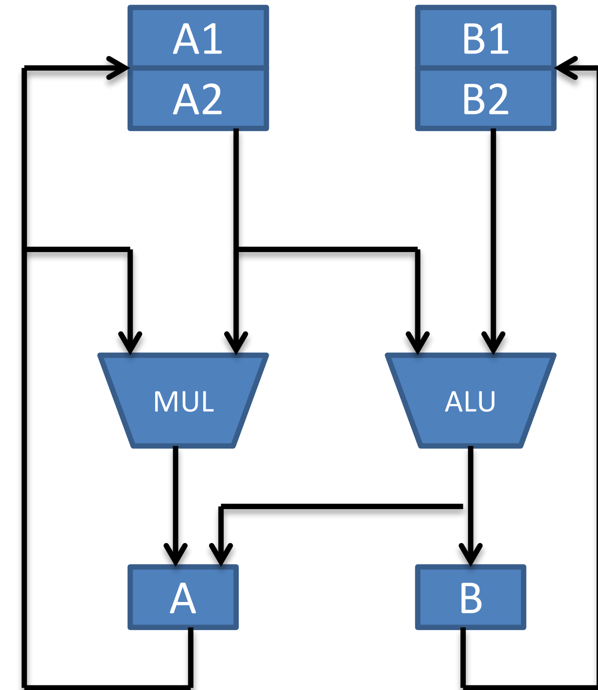
- Akkumulator (Accu) ist für die meisten Operationen
  - Zieloperand und
  - Quelloperand



- Vorteil:
  - Accu muss nicht extra als Operand kodiert werden
  - Geringer Hardwarebedarf, da Multiplexer eingespart werden
- Nachteil:
  - Häufigeres Auslagern des Accu-Inhalts verlängert Programmcode und Laufzeit

Spezialregisterarchitektur

- Datenpfad enthält verschiedene funktionale Einheiten
- Einzelne Register sind nur von bestimmten funktionalen Einheiten als Quell-/Zieloperand nutzbar
- Vorteil:
  - Platzsparende Kodierung der Operanden möglich
  - Einsparung bei den Multiplexern
  - Bitbreite der Register kann an die Erfordernisse der zugehörigen Operation angepasst werden
- Nachteil:
  - Zusätzliche Transferoperationen zwischen den Registern können erforderlich sein



Hauptregistersatz IA32

Akkumulator
(effiziente Kodierung in Befehlen)

Basisregister
(zur Adressberechnung)

Count-Register
(Zählerwert für Hardwareschleifen)

DX-Register
(Verwendung bei Div/Mul)

Indexregister (Source/Destination)
für Speicherkopierbefehle

Basepointer

Stackpointer

Segmentregister
(zur virtuellen Speicherverwaltung)

Gliederung Befehlssatzarchitektur

- Unterstützte Operationen
- Zulässige Datenformate für die Operanden
- Anzahl, Art und Verwendung der Operanden in einem Befehl
  - Registerarchitektur
  - Adressierungsmöglichkeiten des Speichers
- Kodierung der Befehle

Überblick Adressierungsarten

Adressierung von Operanden

- in einem Prozessorregister: Adresse ist die Registerbezeichnung (Registeroperand)
- im Speicher:
  - an einer angegebenen Adresse (Speicheroperand)
  - im Befehlscode als Konstante (Immediate)

Adressierung dieser Operanden

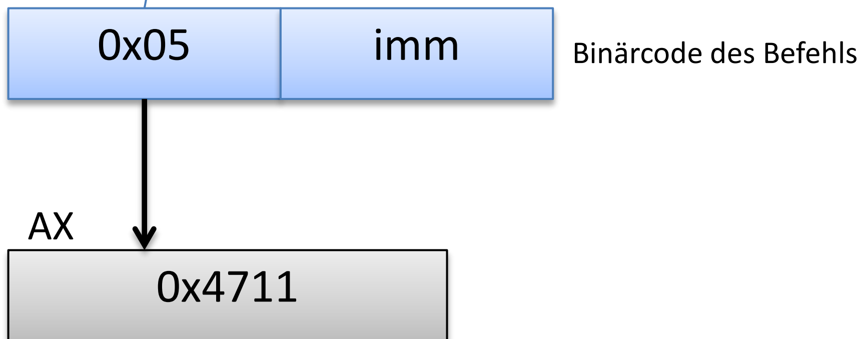
- Implizite Adressierung
- Unmittelbare Adressierung
- Registeradressierung
- Absolute Speicheradressierung
- Registerindirekte Speicheradressierung (mit Displacement / mit Autoincrement/Dekrement)
- Basisindizierte Adressierung (skaliert / mit Displacement)

Load-/Store-Architekturen erlauben Speicheroperanden nur in speziellen Befehlen für den Speicherzugriff (z.B. ATmege, ARM)

Adressierungsarten

Implizite Adressierung

- Registeroperand ist implizit im Befehl kodiert
- Beispiel IA-32:
  - MUL EBX — berechnet EDX:EAX := EAX · EBX
  - ADD AX, imm — berechent AX := AX + imm



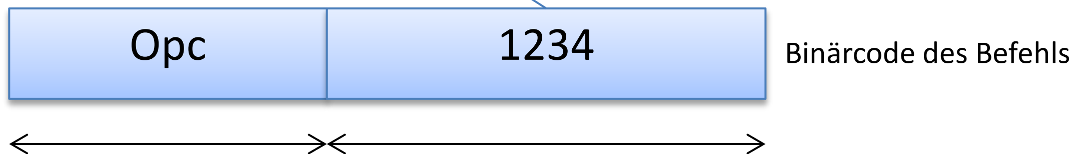
Binärcode des Befehls

Quell-/Zieloperand AX ist implizit adressiert durch den opcode 0x05

Adressierungsarten

Unmittelbare (immediate) Adressierung

- Operand ist eine Konstante die im Befehl kodiert ist
- Beispiel ARM
  - ADD R2, R3, #5 berechnet R2 := R3 + 5
- Beispiel Atmega:
  - SUBI R3, $89 berechnet R3 := R3 - 89
- Beispiel IA-32:
  - ADD AX, 1234 berechnet AX := AX + 1234

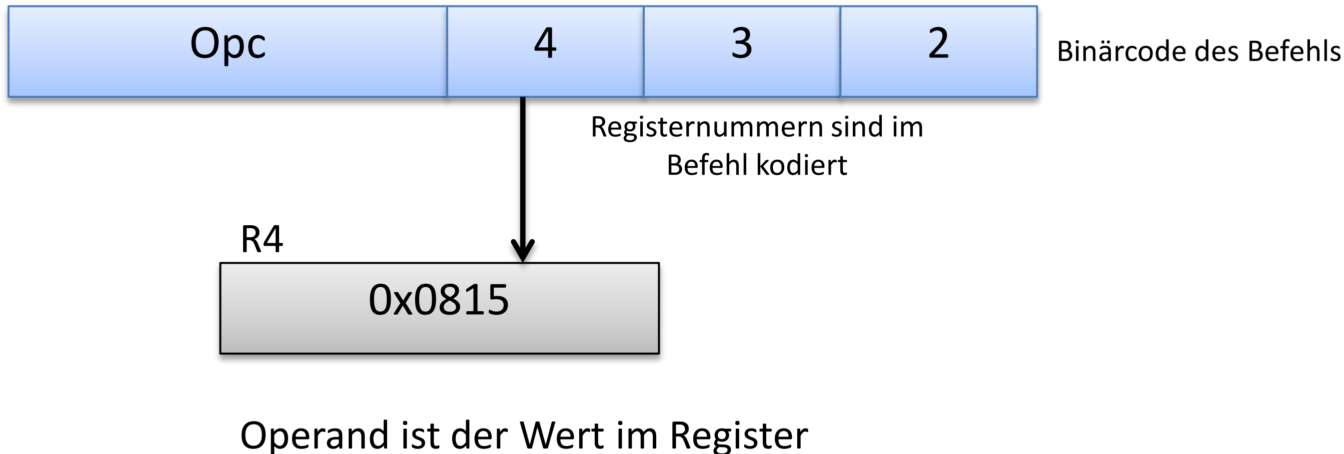


Operand ist die Konstante *imm*

Adressierungsarten

Registeradressierung

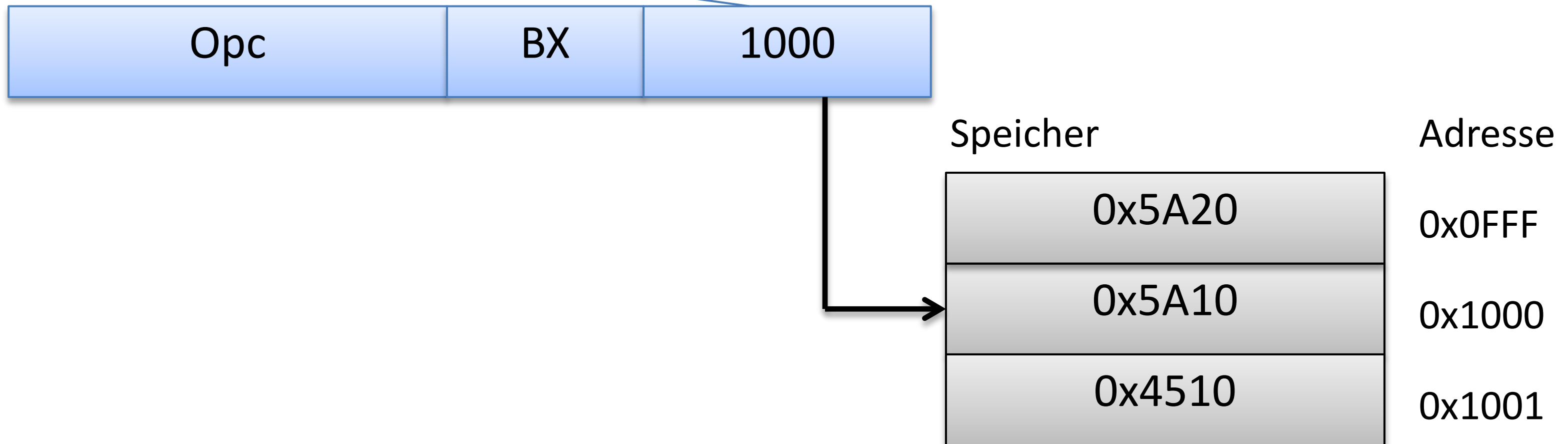
- Befehl kodiert die Registernummer
- Beispiel IA32:
  - ADD EBX,ECX berechnet EBX := EBX + ECX
- Beispiel ARM:
  - ADD R4,R3,R2 berechnet R4 := R3 + R2
- Beispiel Atmega
  - ADD R3,R5 berechent R3 := R3 + R5



Adressierungsarten

Absolute Speicheradressierung

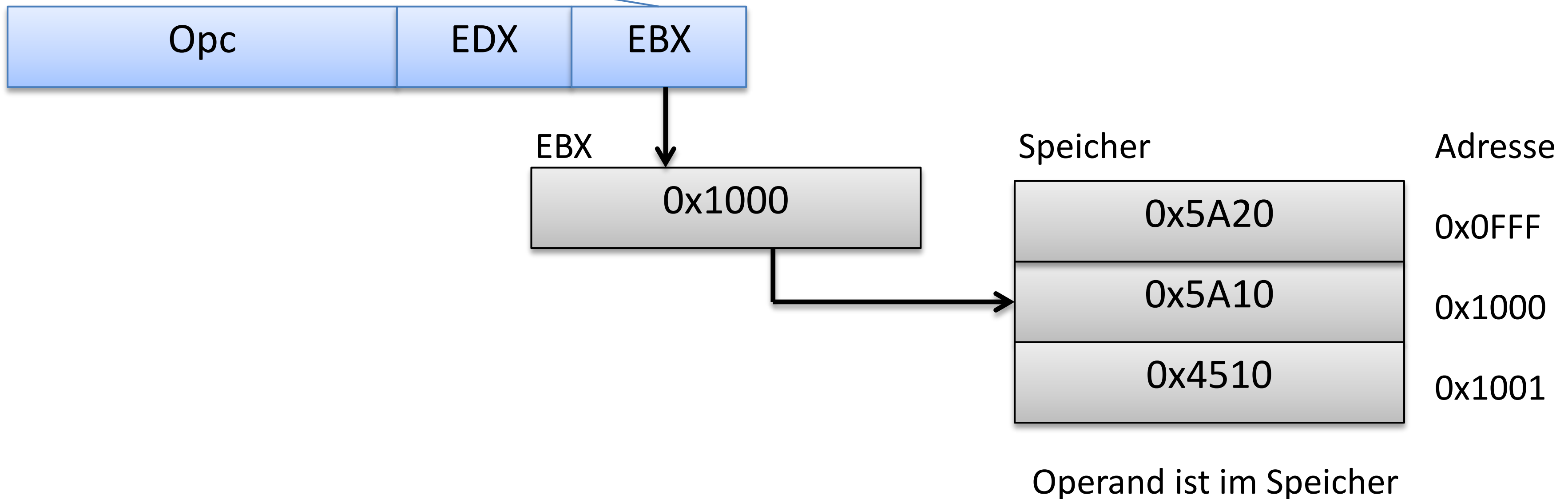
- Befehl kodiert eine Speicheradresse als Konstante
- Verwendung: Zugriff auf globale Variablen (haben konstante Adresse)
- Beispiel Atmega:
  - LDS R1, 0x4000 berechnet R1 := MEM[4000]
- Beispiel ARM: nicht verfügbar
- Beispiel IA-32:
  - ADD BX,[1000] berechnet BX := BX + MEM[1000]



Adressierungsarten

Registerindirekte Speicheradressierung

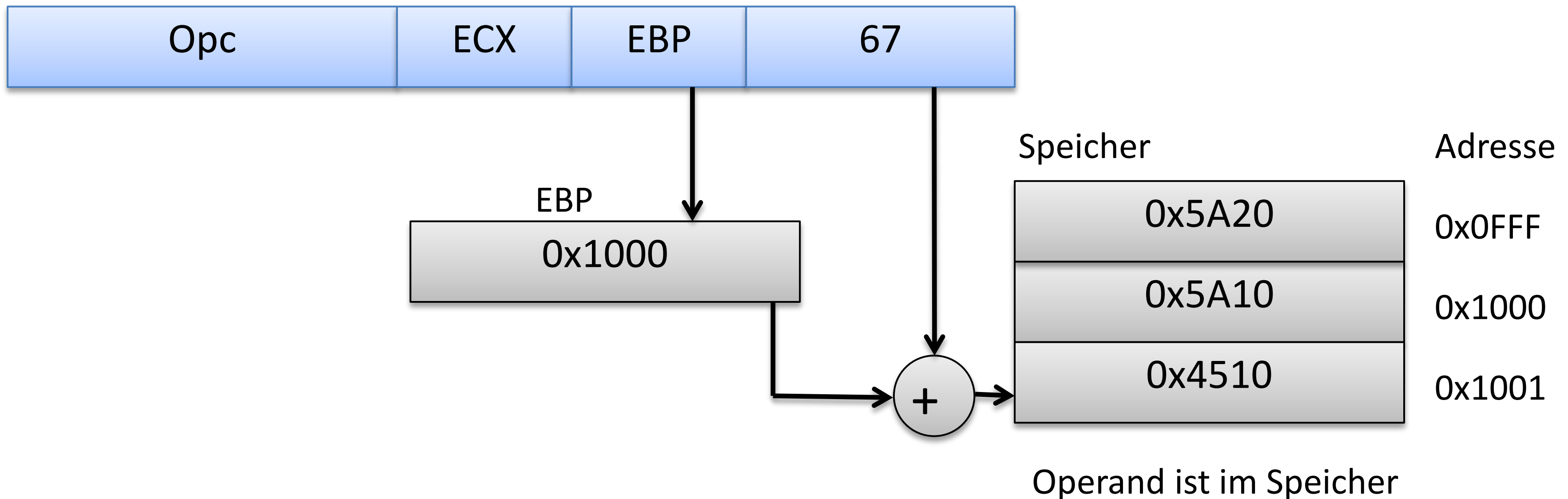
- Befehl kodiert ein Register
- Register enthält eine Speicheradresse
- Verwendung: Zeiger (Register enthält den Wert des Zeigers)
- Beispiel ARM:
  - LDR R1,[R4]   berechnet R1 := MEM[R4]
- Beispiel Atmega
  - LD R0,X   berechnet R0 := MEM[256 * R27 + R26]
- Beispiel IA-32:
  - ADD EDX, [EBX]   berechnet EDX := EDX + MEM[EBX]



Adressierungsarten

Registerindirekte Speicheradressierung mit Displacement

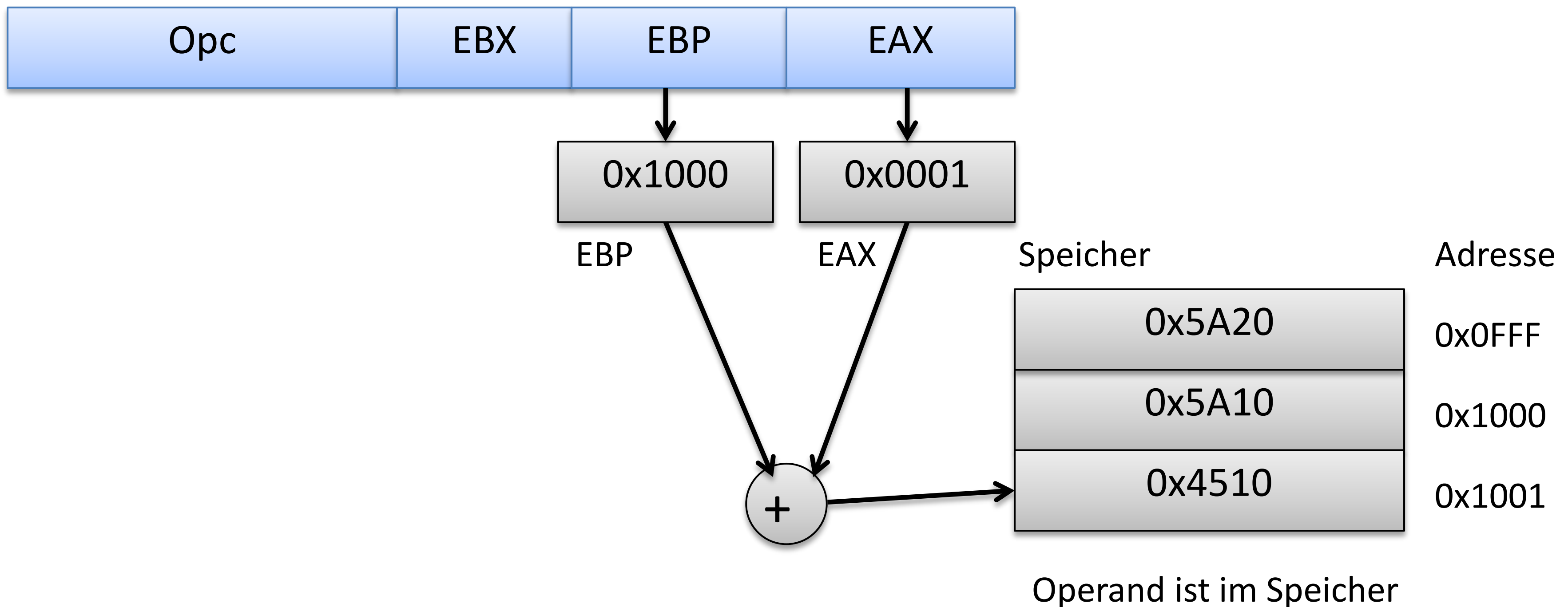
- Befehl kodiert ein Register und ein Displacement (Offset)
- Speicheradresse ergibt sich aus Registerwert + Displacement
- Verwendung: Zugriff auf lokale Variablen
- Beispiel ARM (Displacement ist auf 13 Bit beschränkt):
  - LDR R0, [R5, #1]   berechnet R0 := MEM[R5 + 1]
- Beispiel Atmega (nur für Y und Z möglich):
  - Ldd r0,Z+0x10   berechnet R0 := MEM[256*R31 + R30]
  - Std Y+20,r15
- Beispiel IA-32:
  - ADD ECX,[EBP+67]   berechnet ECX := ECX + MEM[EBP+67]



Adressierungsarten

Basisindizierte Speicheradressierung

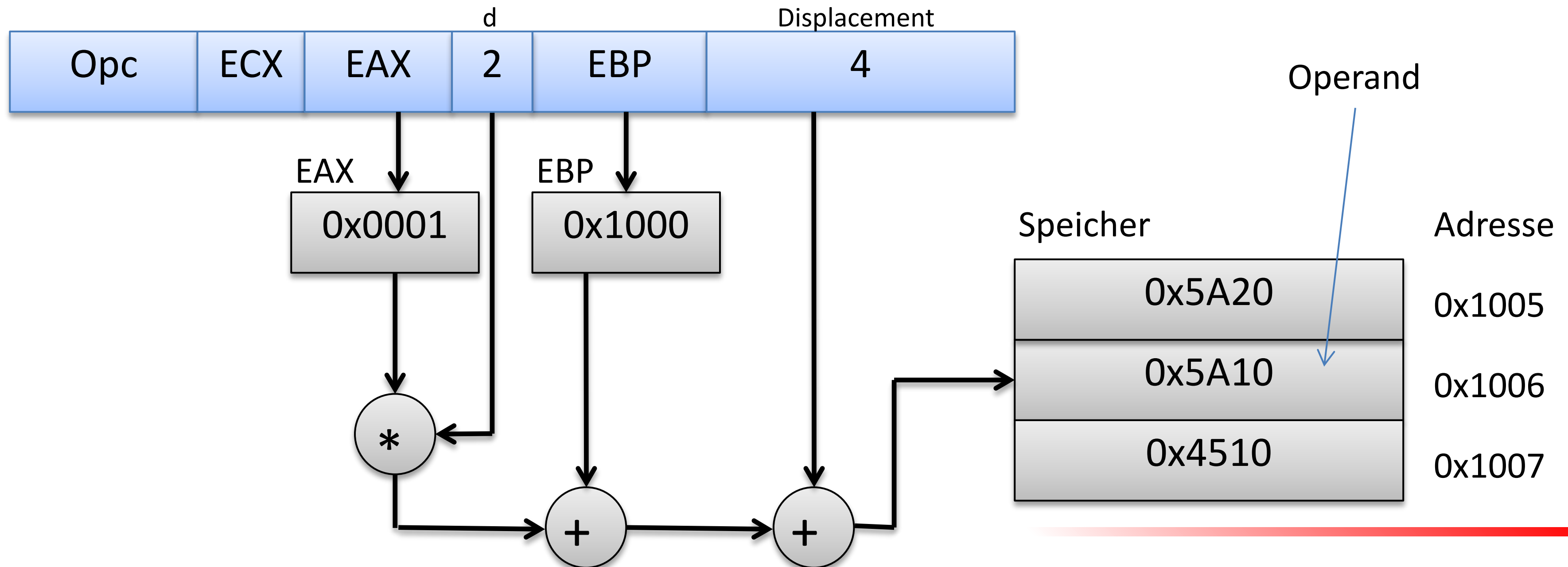
- Befehl kodiert ein Basisregister und ein Indexregister
- Speicheradresse ergibt sich aus Basisregisterwert + Indexregisterwert
- Verwendung: z.B. bei Feldzugriffen
- Beispiel IA-32:
  - ADD EBX, [EAX + EBP)   berechnet EBX := EBX + MEM[EAX + EBP]
- Beispiel ARM:
  - STR   R0, [R5, R1]   berechnet MEM[R5 + R1] := R0
- Atmega: nicht unterstützt



Adressierungsarten

Basisindiziert und skaliert mit Displacement

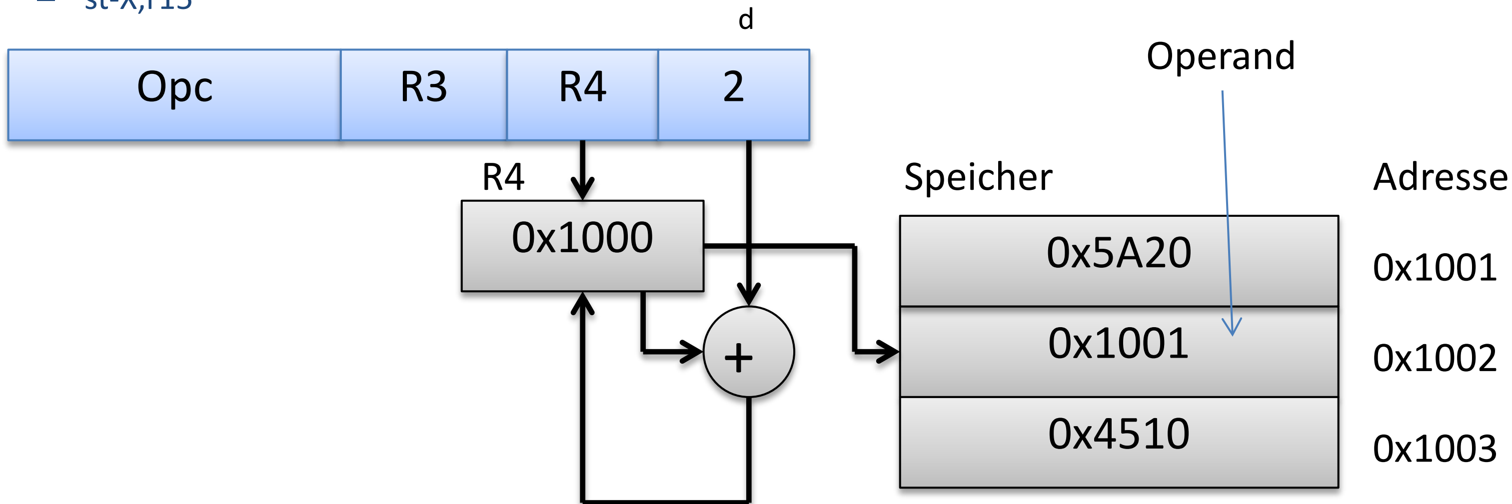
- Befehl kodiert ein Basis-, ein Indexregister, ein Displacement und einen Skalierungsfaktor
- Registerinhalt ist eine Speicheradresse, an der sich der Operand befindet
- Verwendung: Zugriff auf Feldelemente im Stack
- Beispiel IA-32:
  - ADD ECX, [d*EAX+EBP+4] berechnet ECX := ECX + MEM[d*EAX+EBP+4], wobei d = 1, 2, 4 oder 8
- Beispiel ARM (kein Displacement möglich)
  - STR R0, [R1, R2, LSL #2] berechnet MEM[R1 + 4*R2] := R0
- ATmega: nicht unterstützt



Adressierungsarten

Autoincrement/Autodecrement

- Befehl kodiert ein Register
- Register wird für Registerindirekte Adressierung verwendet und anschließend incrementiert/decrementiert
- Verwendung: Schleifen über Felder
- Beispiel ARM:
  - STR R3, [R4], #4 berechnet MEM[R4] := R3; R4 := R4 + 4
- Beispiel Atmega:
  - ld r0,X+
  - st X+,r15
  - ld r0,-X
  - st-X,r15



Gliederung Befehlssatzarchitektur

- Unterstützte Operationen
- Zulässige Datenformate für die Operanden
- Anzahl, Art und Verwendung der Operanden in einem Befehl
  - Registerarchitektur
  - Adressierungsmöglichkeiten des Speichers
- Kodierung der Befehle

Complex Instruction Set Computer (CISC)

Eigenschaften:

- Komplexe Operationen
- Unterstützung vieler verschiedener Operationen mit vielen verschiedenen Adressierungsmöglichkeiten
- Irreguläre Kombination von Operationen und Adressierungen
- Irreguläre Kodierung von Befehlen

Gründe für CISC:

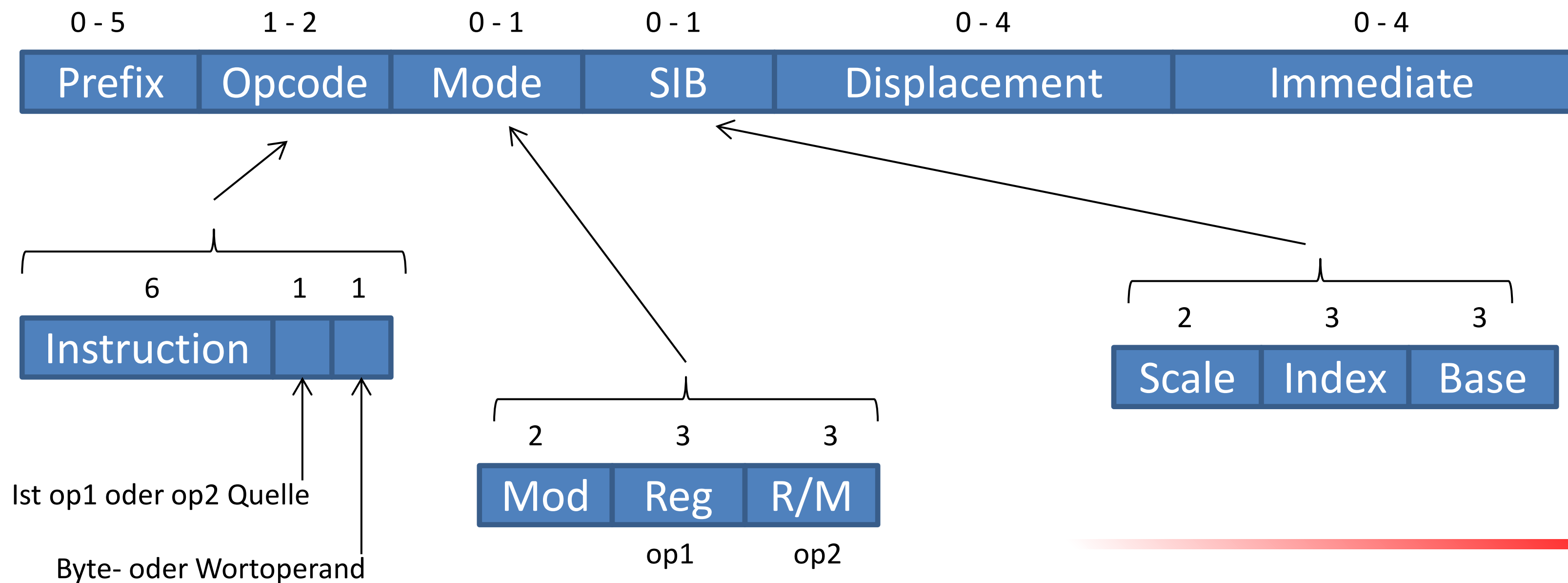
- Kompakte Kodierung von Befehlen wurde angestrebt
- Abwärtskompatibilität ermöglichen

Beispiele:

- Intel x86-Architektur
- IBM 370

Befehlskodierung der IA32

- Befehle werden in bis zu 6 Felder variabler Länge kodiert
- Davon sind 5 Felder optional
- Kompletter Befehl muss dekodiert werden, um Befehlslänge zu ermitteln
- Dadurch komplexe und langwierige Befehlsdekodierphase



Prefix-Bytes

- Standardnutzung von Segmentregistern überschreiben
  - 0x2E – CS-Override, 0x36 – SS-Override, …
- Operandengröße ändern:
  - 0x66 legt Operandengröße auf 32-Bit fest (im 16-Bit Modus)
- Adressgröße ändern
  - 0x67 ändert Größe des Adressoperanden auf 32 Bit (im 16-Bit Modus)
- Sprungvorhersage
  - 0x2E bedeutet: Folgender Sprung wird eher nicht ausgeführt
  - 0x3E bedeutet: Folgender Sprung wird eher genommen

Adressierungsarten im 32-Bit-Modus

MOD

R/M	MOD = 00	MOD = 01	MOD = 10	MOD = 11
000	M[EAX]	M[EAX+Offset8]	M[EAX+Offset32]	EAX oder AL
001	M[ECX]	M[ECX+Offset8]	M[ECX+Offset32]	ECX oder CL
010	M[EDX]	M[EDX+Offset8]	M[EDX+Offset32]	EDX oder DL
011	M[EBX]	M[EBX+Offset8]	M[EBX+Offset32]	EBX oder DH
100	SIB	SIB+Offset8	SIB+Offset32	ESP oder AH
101	Direkt	M[EBP+Offset8]	M[EBP+Offset32]	EBP oder CH
110	M[ESI]	M[ESI+Offset8]	M[ESI+Offset32]	ESI oder DH
111	M[EDI]	M[EDI+Offset8]	M[EDI+Offset32]	EDI oder BH

SIB

Scale	Index	Base
00: d = 1	000: EAX	000: EAX
01: d = 2	001: ECX	001: ECX
10: d = 4	010: EDX	010: EDX
11: d = 8	011: EBX	011: EBX
	100: ESP	100: ESP
	101: EBP	101: EBP
	110: ESI	110: ESI
	111: EDI	111: EDI

Reduced Instruction Set Computer (RISC)

Seit Anfang der 80er Jahre vermehrte Nutzung von Compilern mit folgender Beobachtung auf einer IBM 370 (ca. 200 Befehle):

- 80% des compilergenerierten Programmcodes nutzen nur 10 verschiedene Befehle
- 95% des compilergenerierten Programmcodes nutzen nur 21 verschiedene Befehle
- 99% des compilergenerierten Programmcodes nutzen nur 30 verschiedene Befehle

Konsequenz war die RISC-Philosophie

Gewünschte Eigenschaften für den Befehlssatz:

- Ausführung eines Maschinenbefehls pro Takt
- Lade-Speicher-Architektur: Speicherzugriff nur über load-/store-Befehle, keine Speicheroperanden in ALU-Befehlen zulässig
- Wünschenswert: Orthogonalität von Opcode und Adressierungsmodus:
  - Alle Opcodes sollten alle Adressierungsmodi zulassen
  - Alle Register sollten gleichwertig genutzt werden können

- Beispiel:

Befehlsformat für dyadische Operationen

Befehlsformat für monadische Operationen

Befehlsformat für Speicheroperationen

Anhand der Gruppe ist Befehlsformat bereits eindeutig festgelegt

Alle Befehle lassen die gleichen Register zu

Zulässige Speicheradressierung unabhängig vom Opcode

Befehlsformate ATmega168

- 6 einfache Befehlsformate; 2 oder 4 Byte lang

00cc	ccrd	dddd	rrrr					ALU-Befehle: Opcode (c), dstReg (d), srcReg (r)
1001	010d	dddd	cccc					ALU-Befehle: Opcode (c), dstReg (d)
01cc	kkkk	dddd	kkkk					ALU + Imm: Opcode (c), dstReg (d), Konstante (k)
10Q0	QQcd	dddd	cQQQ					Ld/st (c) X/Y/Z+Q, Rd
11cc	kkkk	kkkk	kkkk					Branch: br(c) PC + k
1001	010k	kkkk	11ck	kkkk	kkkk	kkkk	kkkk	Call/jmp (c) +k

Befehlsformate Arm-Cortex-A9

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

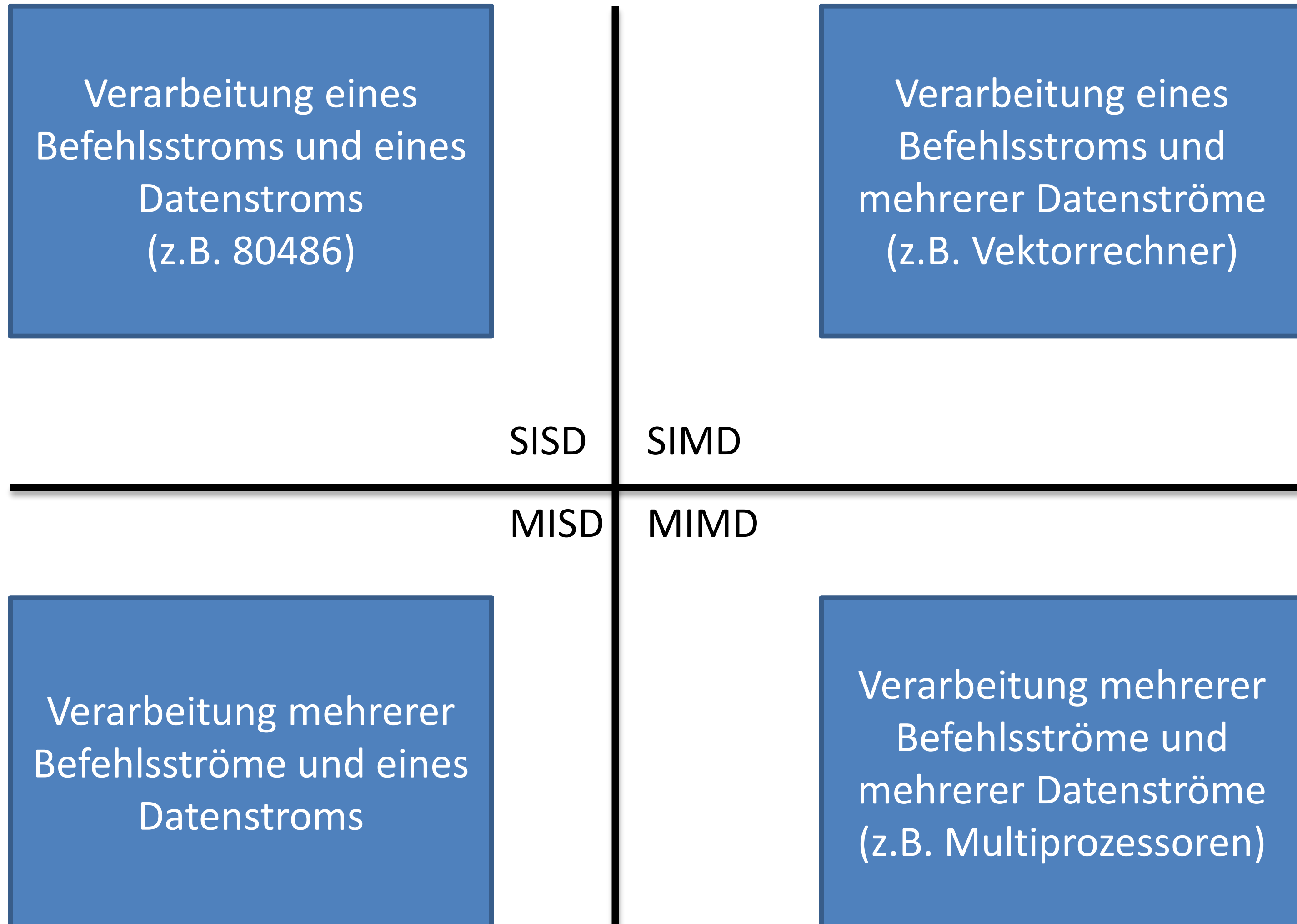
Instruction													
Data processing immediate shift	cond [1]	0 0 0	opcode	S	Rn	Rd	shift amount	shift	0	Rm			
Miscellaneous instructions: See Figure A3-4	cond [1]	0 0 0	1 0 x x	0	x x x x x x x x x x x x x x x	0	x x x x						
Data processing register shift [2]	cond [1]	0 0 0	opcode	S	Rn	Rd	Rs	0	shift	1	Rm		
Miscellaneous instructions: See Figure A3-4	cond [1]	0 0 0	1 0 x x	0	x x x x x x x x x x x x	0	x x	1	x x x x				
Multiplies: See Figure A3-3 Extra load/stores: See Figure A3-5	cond [1]	0 0 0	x x x x x	x x x x x x x x x x x x	1	x x	1	x x x x					
Data processing immediate [2]	cond [1]	0 0 1	opcode	S	Rn	Rd	rotate	immediate					
Undefined instruction	cond [1]	0 0 1	1 0	x	0 0	x x x x x x x x x x x x x x x x x x x x							
Move immediate to status register	cond [1]	0 0 1	1 0	R	1 0	Mask	SBO	rotate	immediate				
Load/store immediate offset	cond [1]	0 1 0	P	U	B	W	L	Rn	Rd	immediate			
Load/store register offset	cond [1]	0 1 1	P	U	B	W	L	Rn	Rd	shift amount	shift	0	Rm
Media instructions [4]: See Figure A3-2	cond [1]	0 1 1	x x x x x x x x x x x x x x x x x x x x	1	x x x x								
Architecturally undefined	cond [1]	0 1 1	1 1 1 1 1	x x x x x x x x x x x x	1 1 1 1	x x x x							
Load/store multiple	cond [1]	1 0 0	P	U	S	W	L	Rn	register list				
Branch and branch with link	cond [1]	1 0 1	L	24-bit offset									
Coprocessor load/store and double register transfers	cond [3]	1 1 0	P	U	N	W	L	Rn	CRd	cp_num	8-bit offset		
Coprocessor data processing	cond [3]	1 1 1 0	opcode1	CRn	CRd	cp_num	opcode2	0	CRm				
Coprocessor register transfers	cond [3]	1 1 1 0	opcode1	L	CRn	Rd	cp_num	opcode2	1	CRm			
Software interrupt	cond [1]	1 1 1 1	swi number										
Unconditional instructions: See Figure A3-6	1 1 1 1	x x x x x x x x x x x x x x x x x x x x x x x x x x x x											

ARM Instruction Set Summary, A3-1

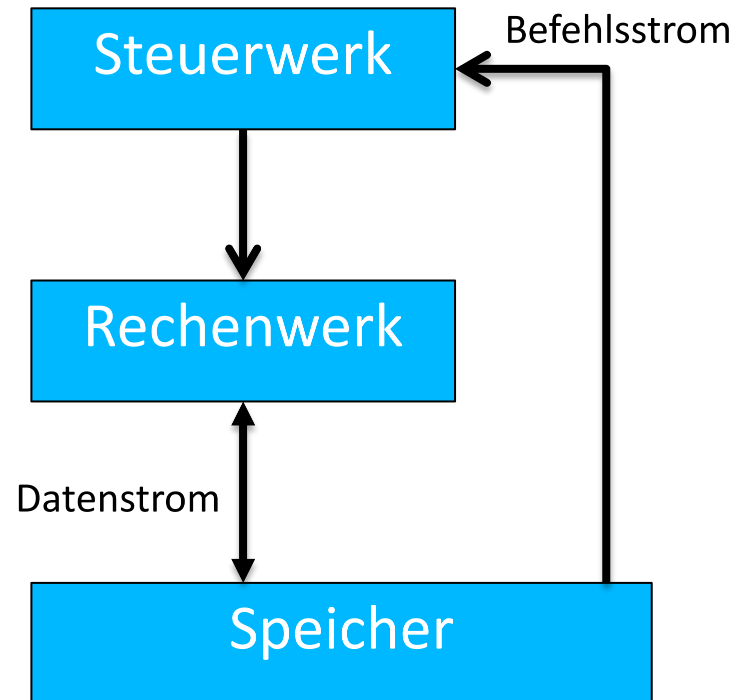
Gliederung

- Beispielprozessoren
- Klassifikation nach Befehlssatzarchitekturen
- Klassifikation nach Arten der Parallelität

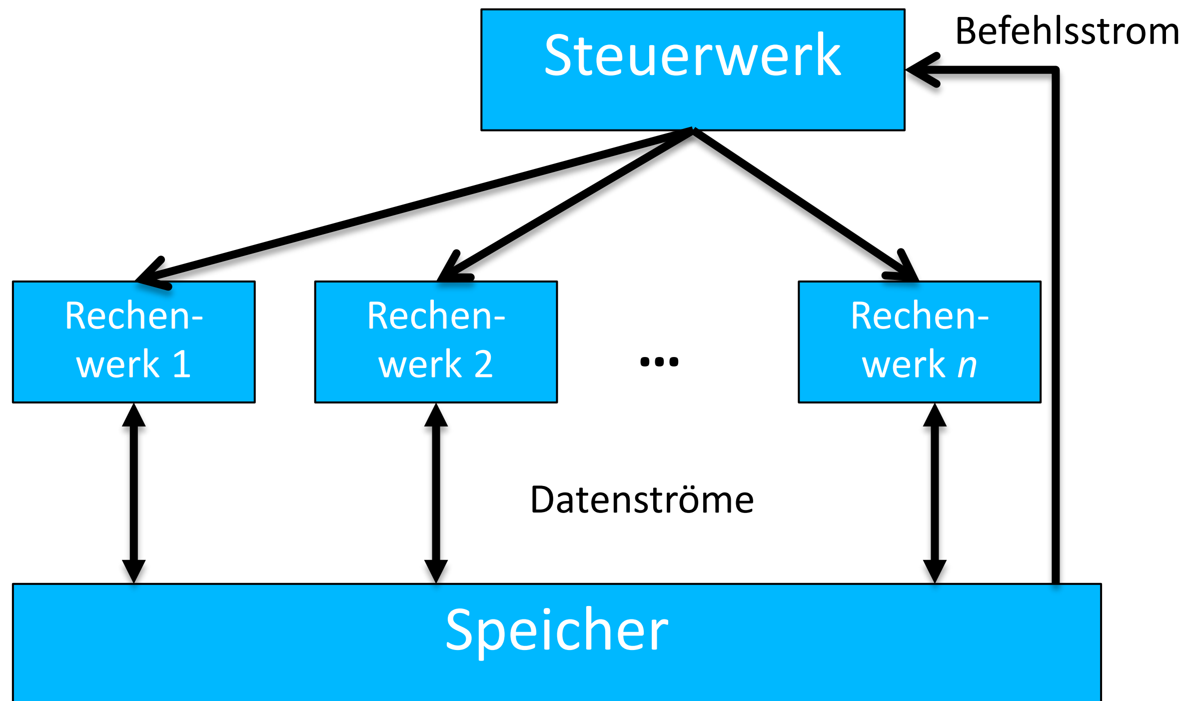
Klassifikation nach Flynn



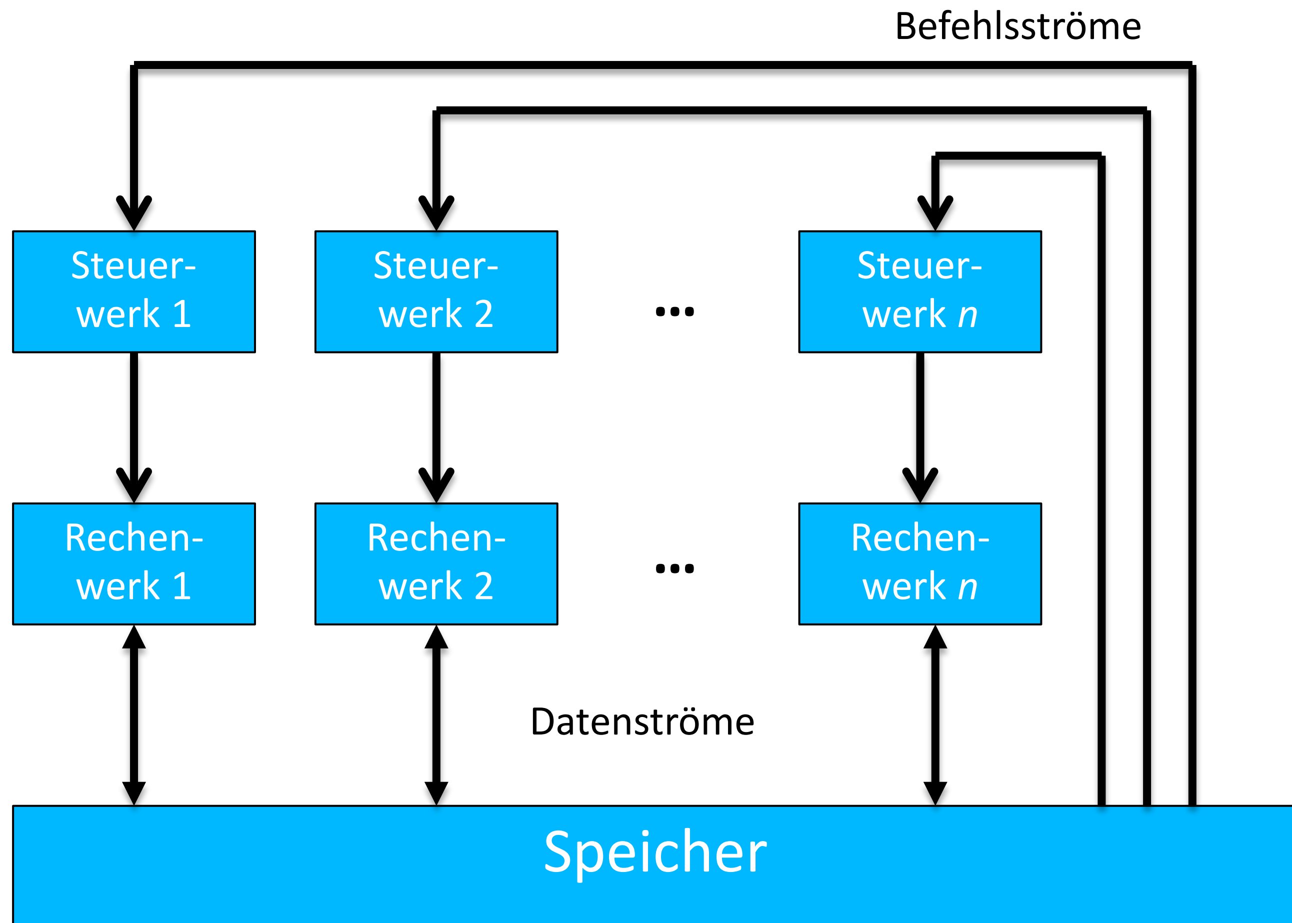
Informationsfluss beim SISD-Rechner



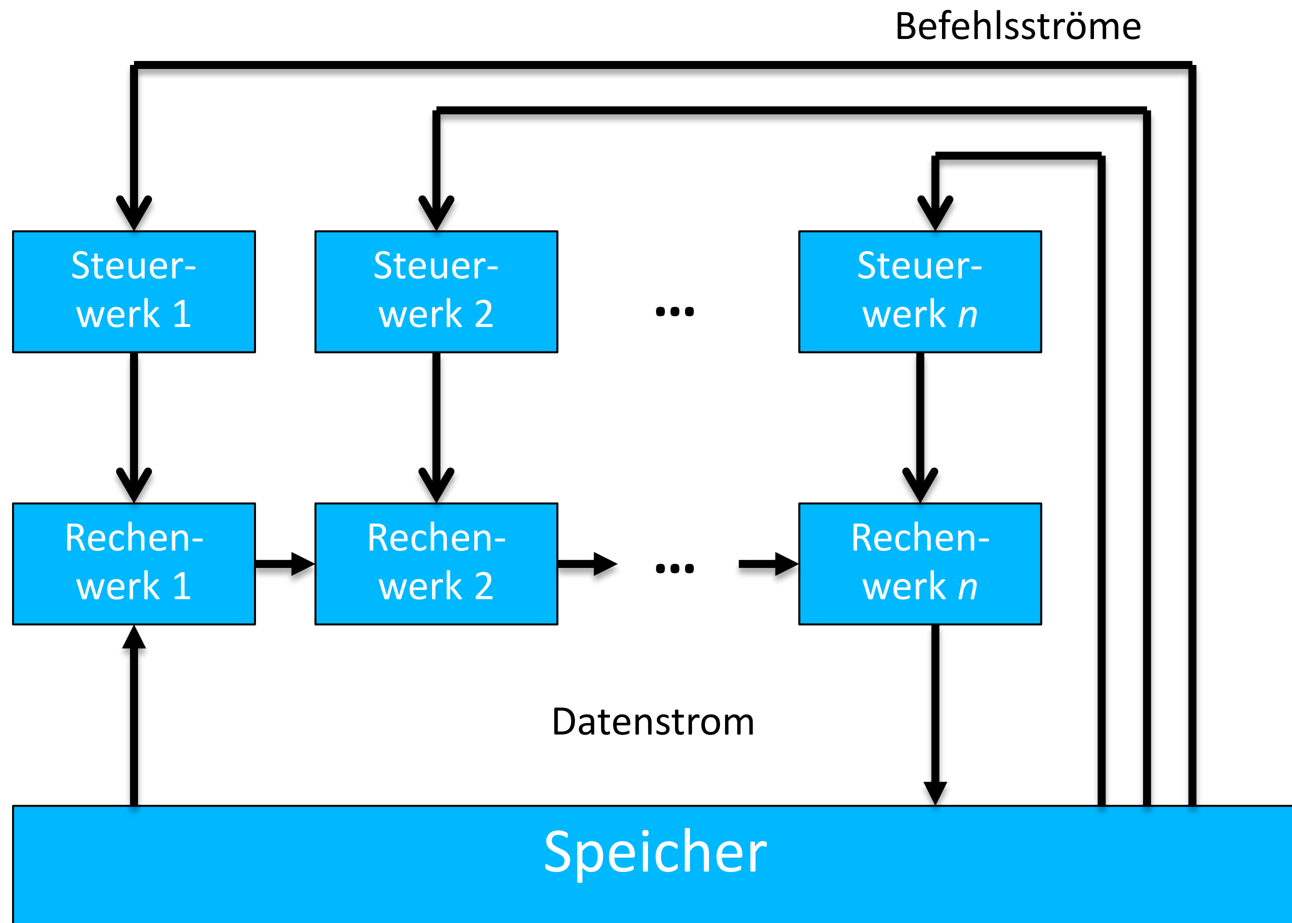
Informationsfluss im SIMD-Rechner



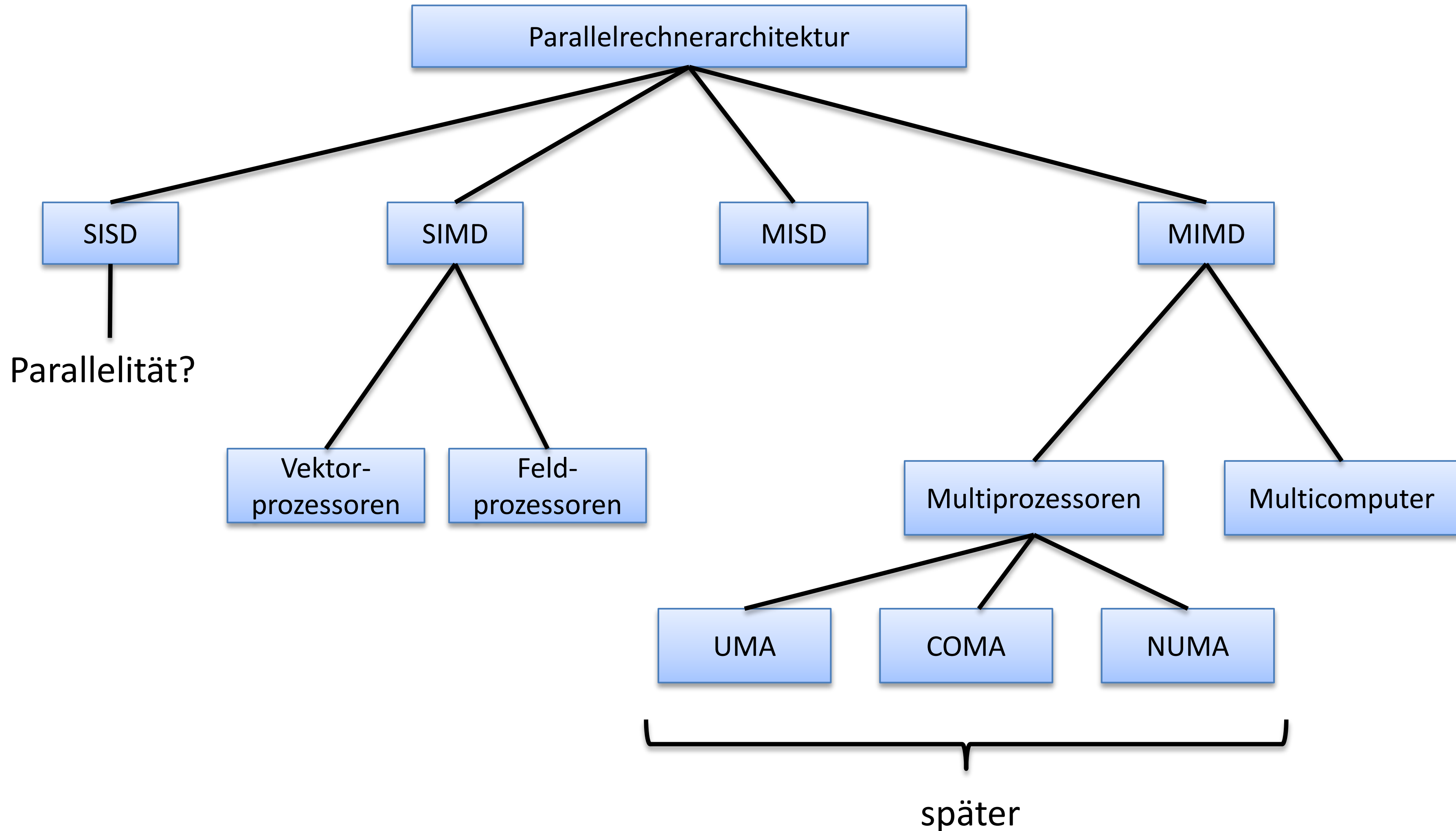
Informationsfluss im MIMD-Rechner



Informationsfluss im MISD-Rechner

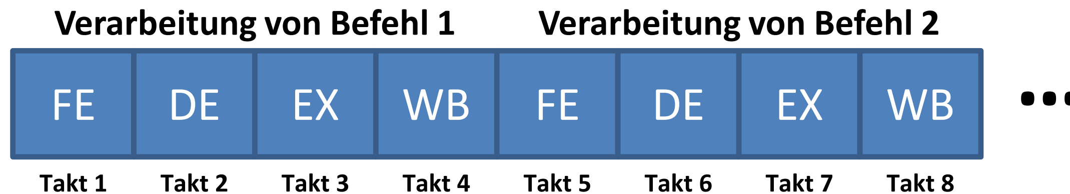


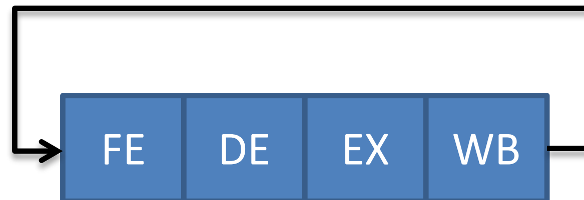
Parallele Prozessorarchitekturen



Befehlsverarbeitung in SISD Prozessoren

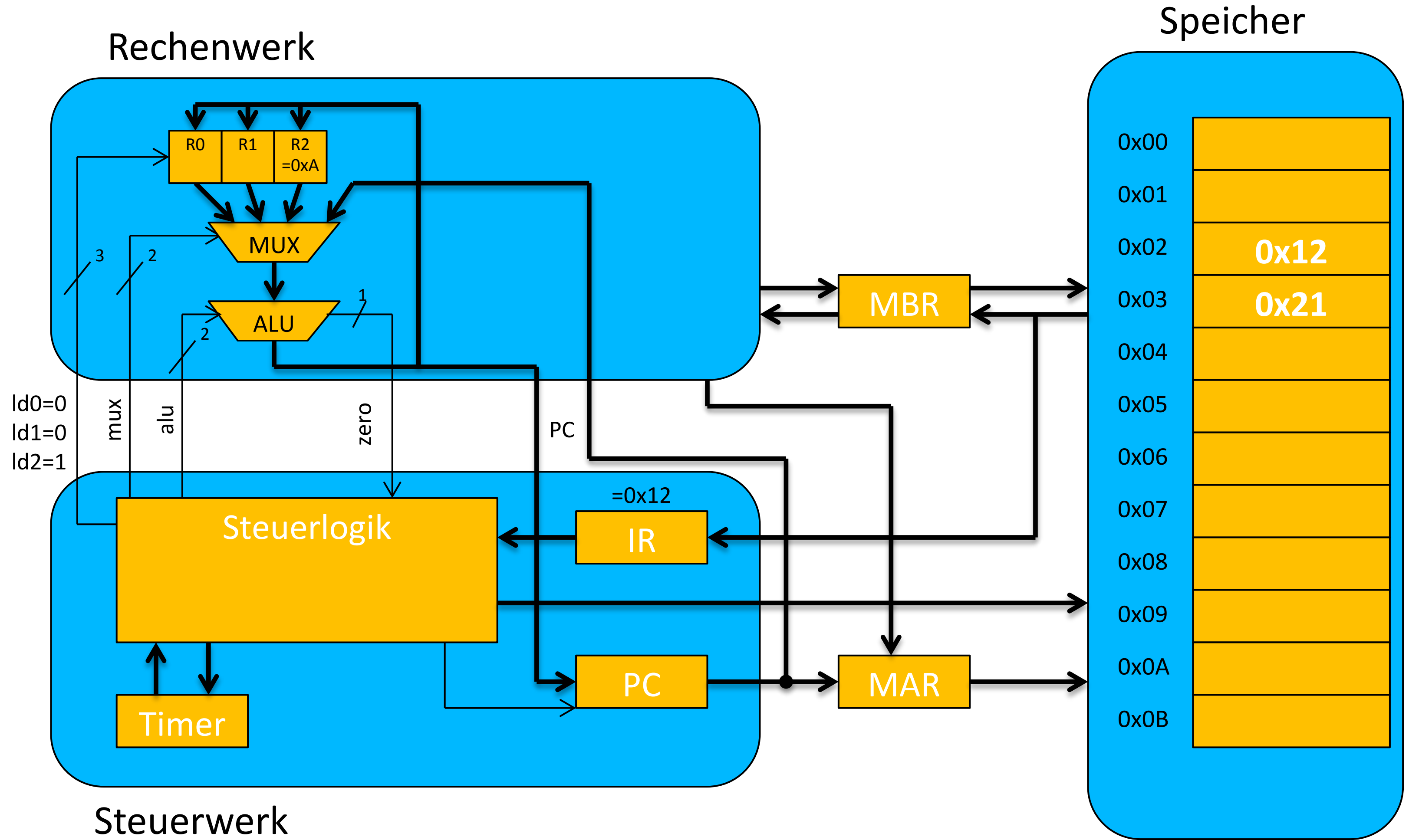
- Multi-Cycle-Prinzip:
  - Eine Instruktion wird in mehreren Takten ausgeführt
  - CPI = cycles per instruction
- Geringe kombinatorische Tiefe in jeder Verarbeitungsstufe
- Dadurch hoher Takt möglich
- Typische Befehlsverarbeitungsschleife:





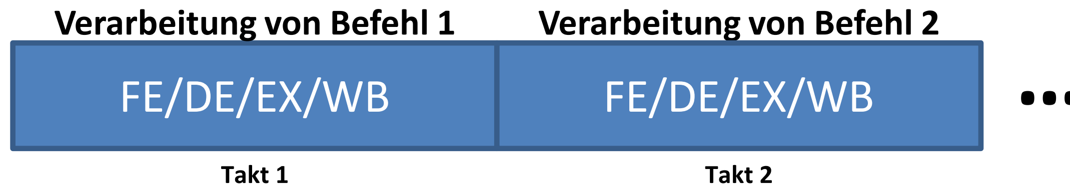
CPI = 4

Beispiel (bekannt)



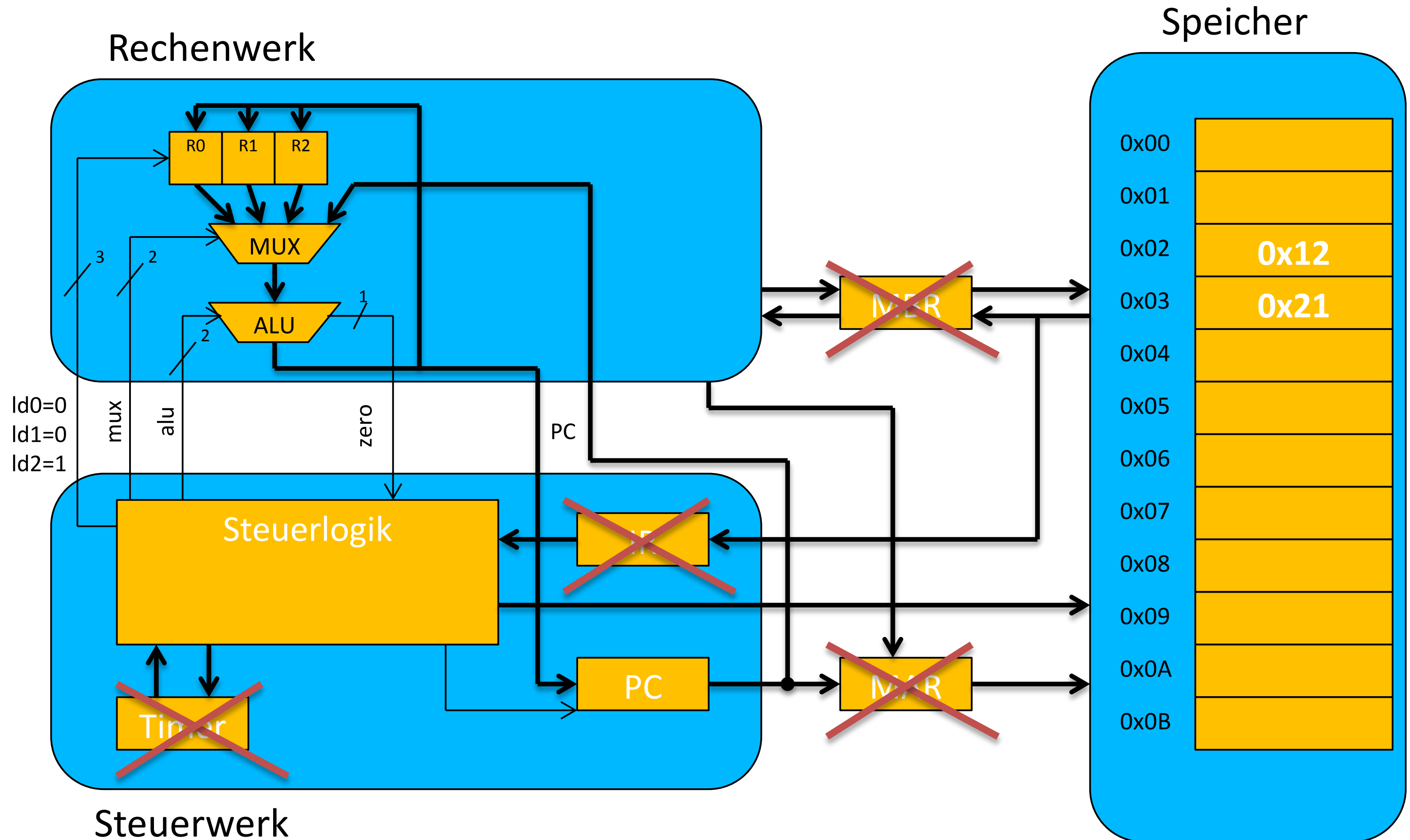
Befehlsverarbeitung in SISD Prozessoren (Single Cycle)

- Single-Cycle-Prinzip:
  - Eine Instruktion wird in einem Takt ausgeführt
- Hohe kombinatorische Tiefe in der Befehlsverarbeitungsschleife
- Dadurch sehr geringer Takt
- Typische Befehlsverarbeitungsschleife:



CPI = 1

Beispiel mit erforderlichen Modifikationen



Parallelität in SISD Architekturen (1)

Parallelität auf Bit-Ebene

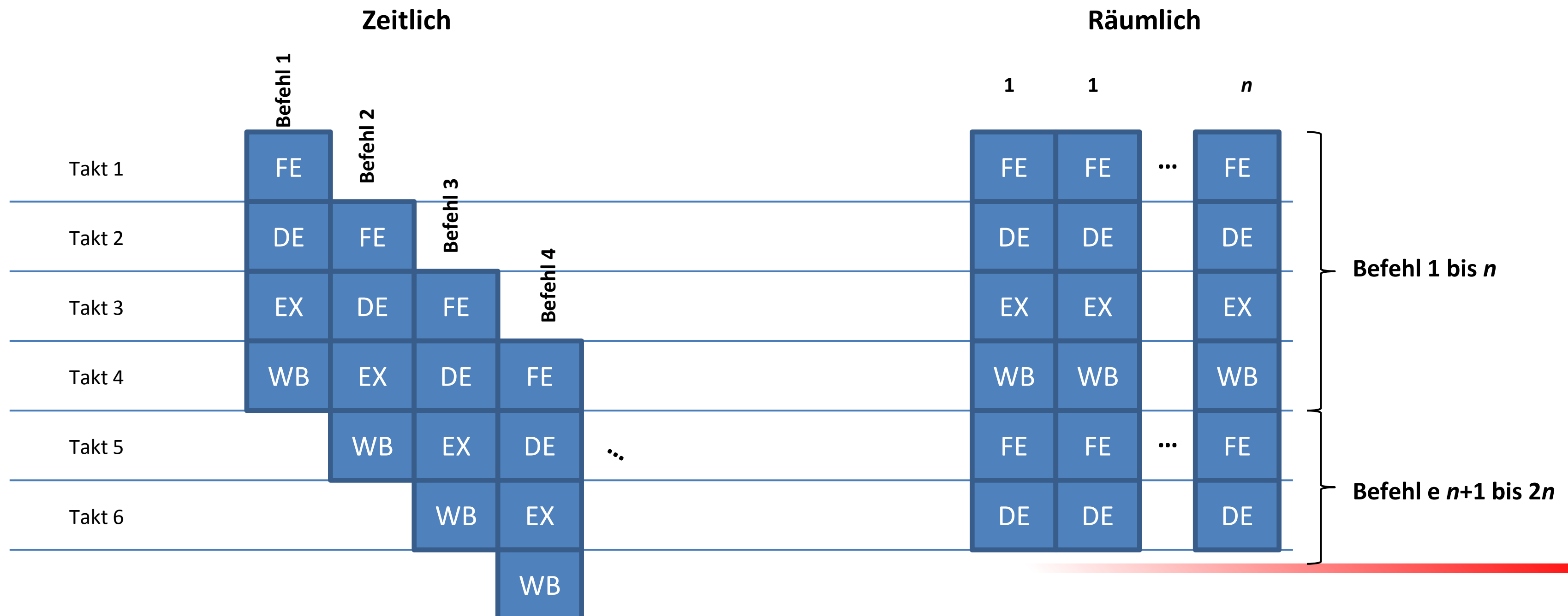
- Datenpfadparallelität wird erweitert:
  - 8-, 16-, 32-, 64-Bit-Architekturen

- Befehlsverarbeitung bleibt unverändert

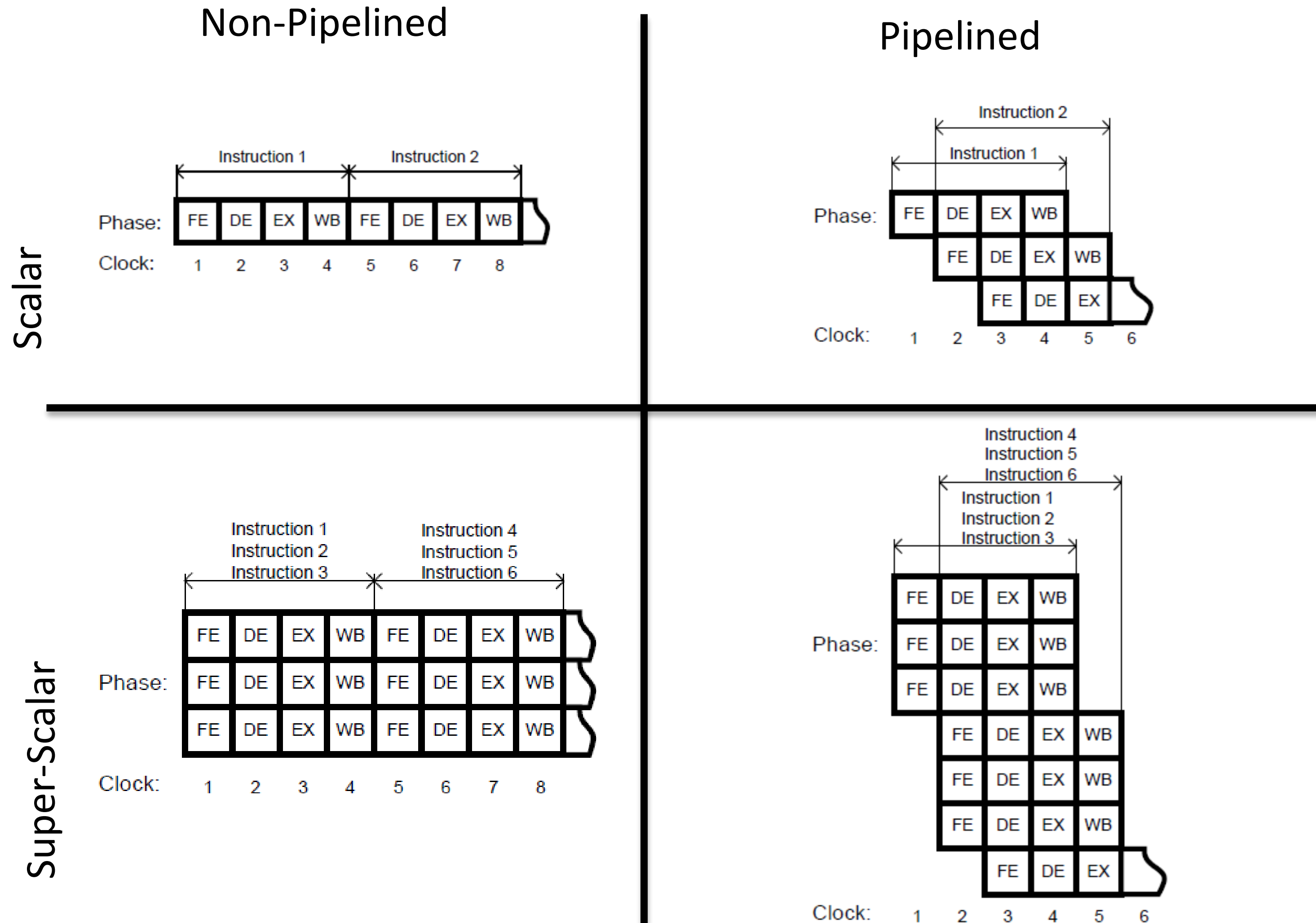
Parallelität in SISD Architekturen (2)

Parallelität auf Befehlsebene (Instruction Level Parallelism, ILP)

- Befehle aus demselben Befehlsstrom werden gleichzeitig ausgeführt
- Zwei Möglichkeiten:
  - Räumliche Parallelität:
    - Mehrere Befehle können sich in derselben Ausführungsphase befinden
  - Zeitliche Parallelität:
    - Alle Befehle befinden sich verschiedenen Ausführungsphasen



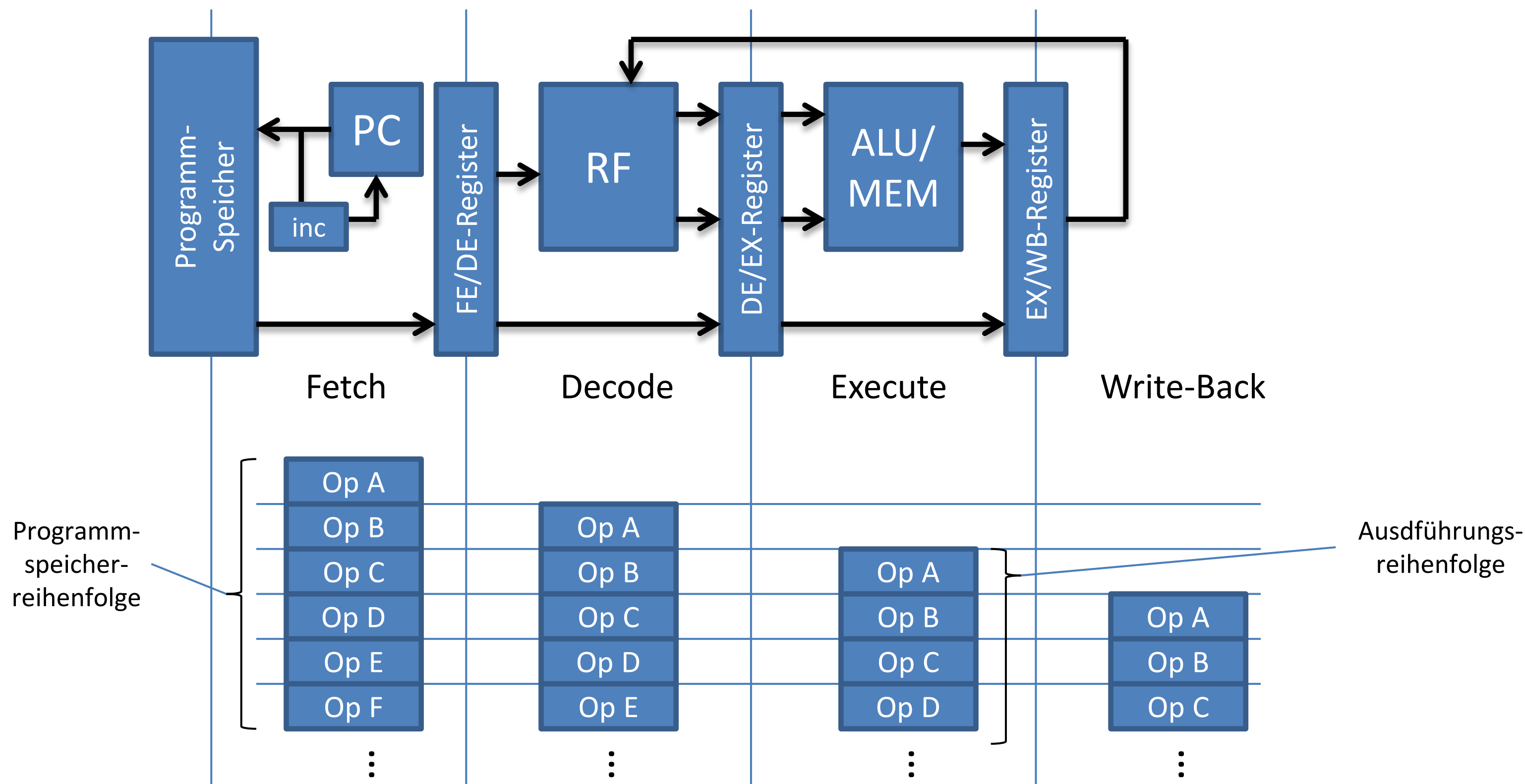
Klassifizierung



Zeitliche Parallelität

Struktur einer Befehlspipeline (4-stufig)

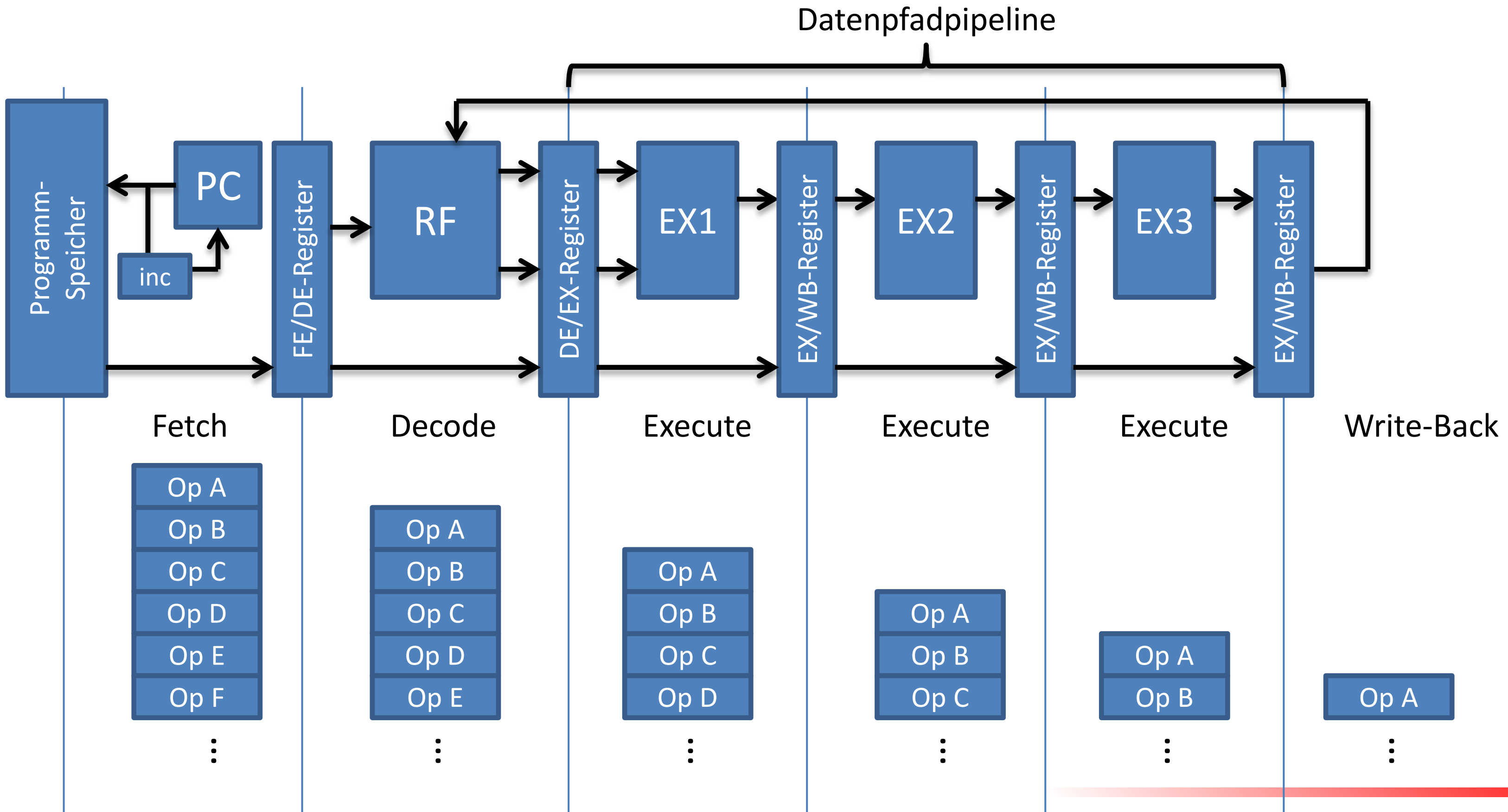
- Skalarer Datenpfad
- Ausführung der Operationen in der Reihenfolge, die im Programmspeicher vorliegt (in-order)



Zeitliche Parallelität

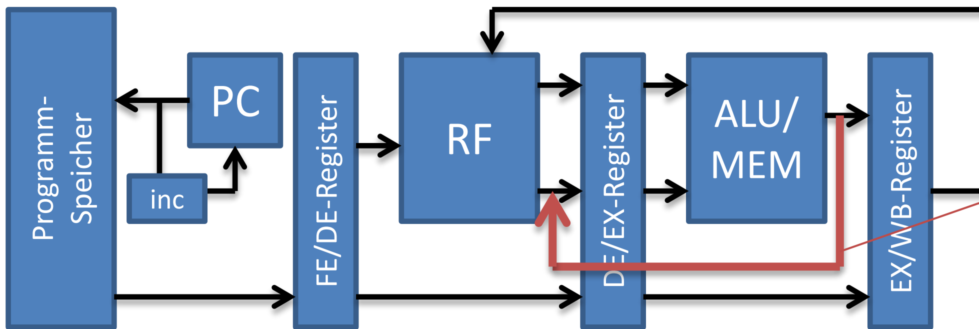
Struktur Befehls- und Datenpfadpipeline (6-stufig)

- Skalarer Datenpfad
- Ausführung der Operationen in-order
- Höherer Takt möglich, da Pipelinestufen in der Regel geringere kombinatorische Tiefe haben



Problem einer langen Execute-Phase

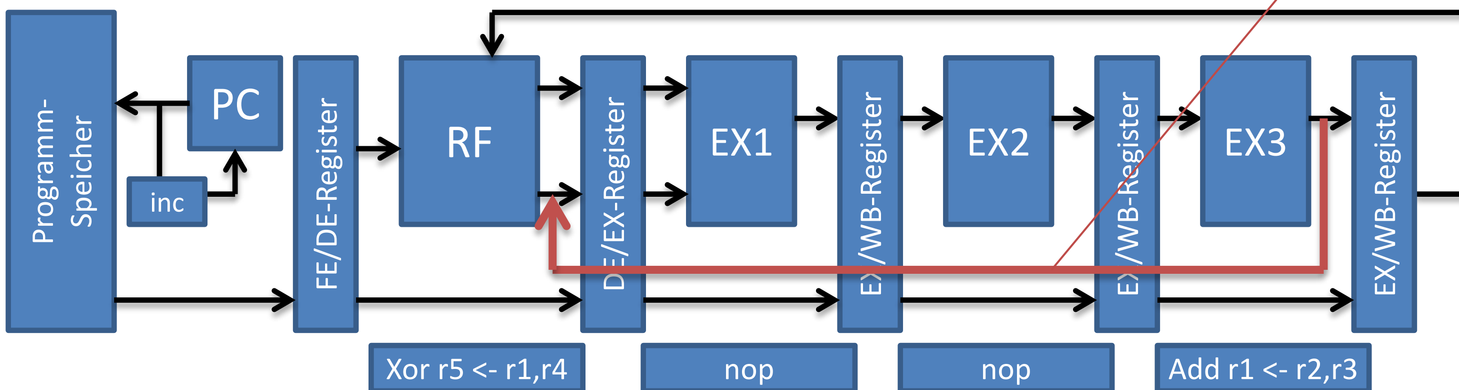
- Kurze Execute-Phase (1 Takt)



Auflösung von Daten-Hazards relativ einfach möglich durch Bypass, weil Ergebnis in einem Takt berechnet wird

- Lange Execute-Phase (3 Takte)

Hier müssen Operationen vor EX1 warten, bis Operand verfügbar (Pipeline-Stall erforderlich)



.
.
.
Add r1 <- r2,r3
Xor r5 <- r1,r4
.
.
.

Vermeidung von Pipeline-Stalls

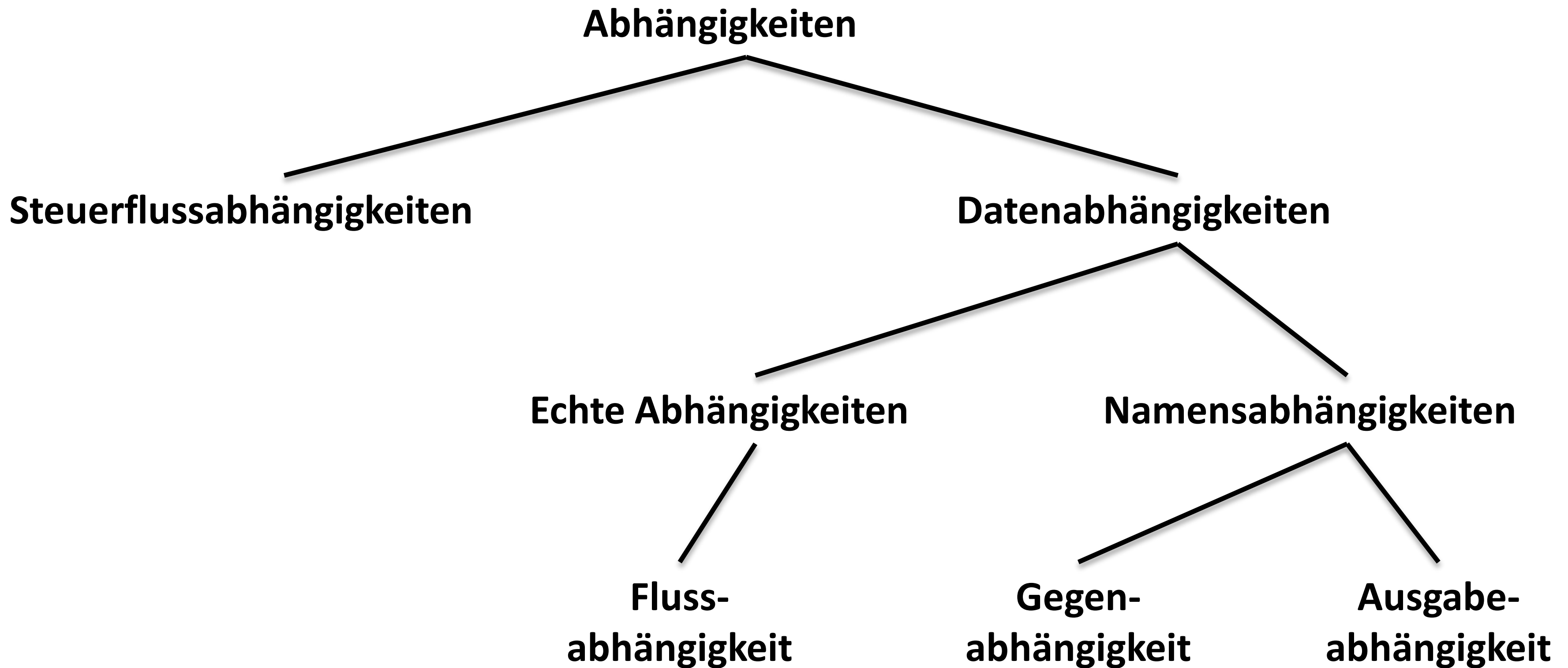
In Software

- Pipeline-Architektur ist dem Compiler (Assemblerprogrammierer) bekannt
- Compiler (Programmierer) muss Operationsreihenfolge finden, bei der die Anzahl der Stalls minimal ist
- Konsequenz: Programm ist optimiert für eine bestimmte Mikroarchitektur

In Hardware

- Pipeline-Architektur ist dem Compiler (Assemblerprogrammierer) nicht bekannt
- Prozessor selbst ändert die Ausführungsreihenfolge, falls möglich (dynamische Ablaufplanung)
- Beachtung von Abhängigkeiten erforderlich
- Prozessor optimiert Programm für seine Mikroarchitektur

Abhängigkeiten

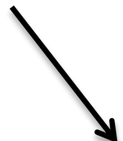
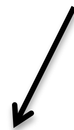


Datenabhängigkeiten

- Flussabhängigkeit: Es gibt eine Flussabhängigkeit von Anweisung *i* zu Anweisung *j*, falls
  - Anweisung *i* einen Wert erzeugt, der von Anweisung *j* genutzt werden kann, oder
  - Es gibt eine Flussabhängigkeit von *i* nach *k* und von *k* nach *j*
- Gegenabhängigkeit: Es gibt eine Gegenabhängigkeit von Anweisung *i* zu Anweisung *j*, falls
  - Anweisung *j* schreibt ein Register oder eine Speicherposition, die von Anweisung *i* gelesen werden kann
- Ausgabeabhängigkeit: Es gibt eine Ausgabeabhängigkeit von Anweisung *i* zu Anweisung *j*, falls
  - *i* und *j* in dieselbe Speicherposition oder in dasselbe Register schreiben können

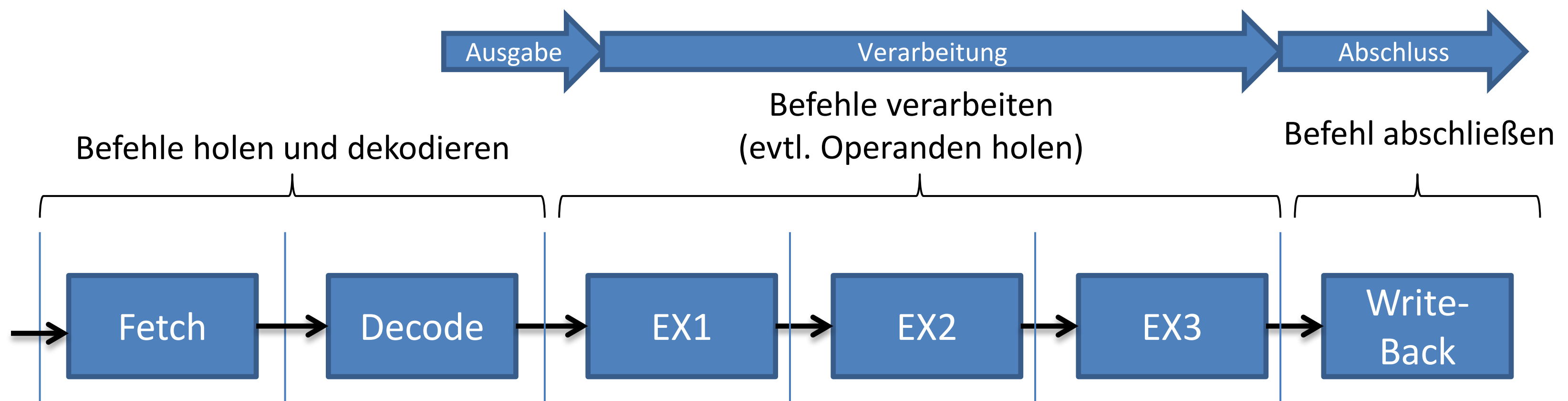
Umordnung von Operationen

Datenabhängigkeit zwischen zwei Operationen erzwingt die Einhaltung der bestehenden Ausführungsreihenfolge

Flussabhängigkeit	**Gegenabhängigkeit**	**Ausgabeabhängigkeit**
add r1, r2, r3	add r1, r2, r3	add r1, r2, r3
		
sub r4, r2, r1	sub r2, r4, r0	sub r1, r4, r0
sub r4, r2, r1	sub r2, r4, r0	sub r1, r4, r0
add r1, r2, r3	add r1, r2, r3	add r1, r2, r3
sub r4, r2, r1 || add r1, r2, r3	sub r2, r4, r0 || add r1, r2, r3	sub r1, r4, r0 || add r1, r2, r3

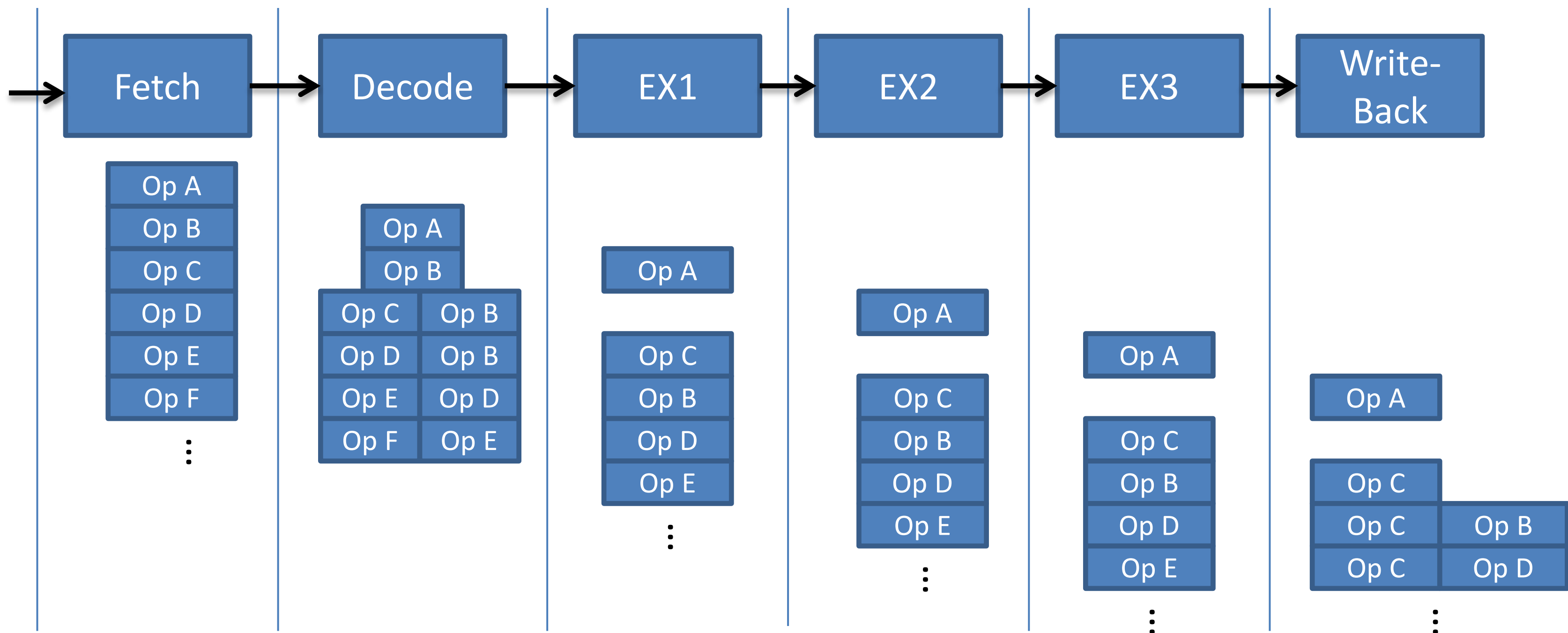
Funktionen einer komplexeren Pipeline

- Ausgabe: Übergabe des Befehls in den Datenpfad (EX-Phasen)
- Verarbeitung: Berechnen des Ergebnisses im Datenpfad
- Abschluss: Ergebnis in die Architekturregister zurückschreiben
- Architekturregister: Für den Programmierer sichtbare Register



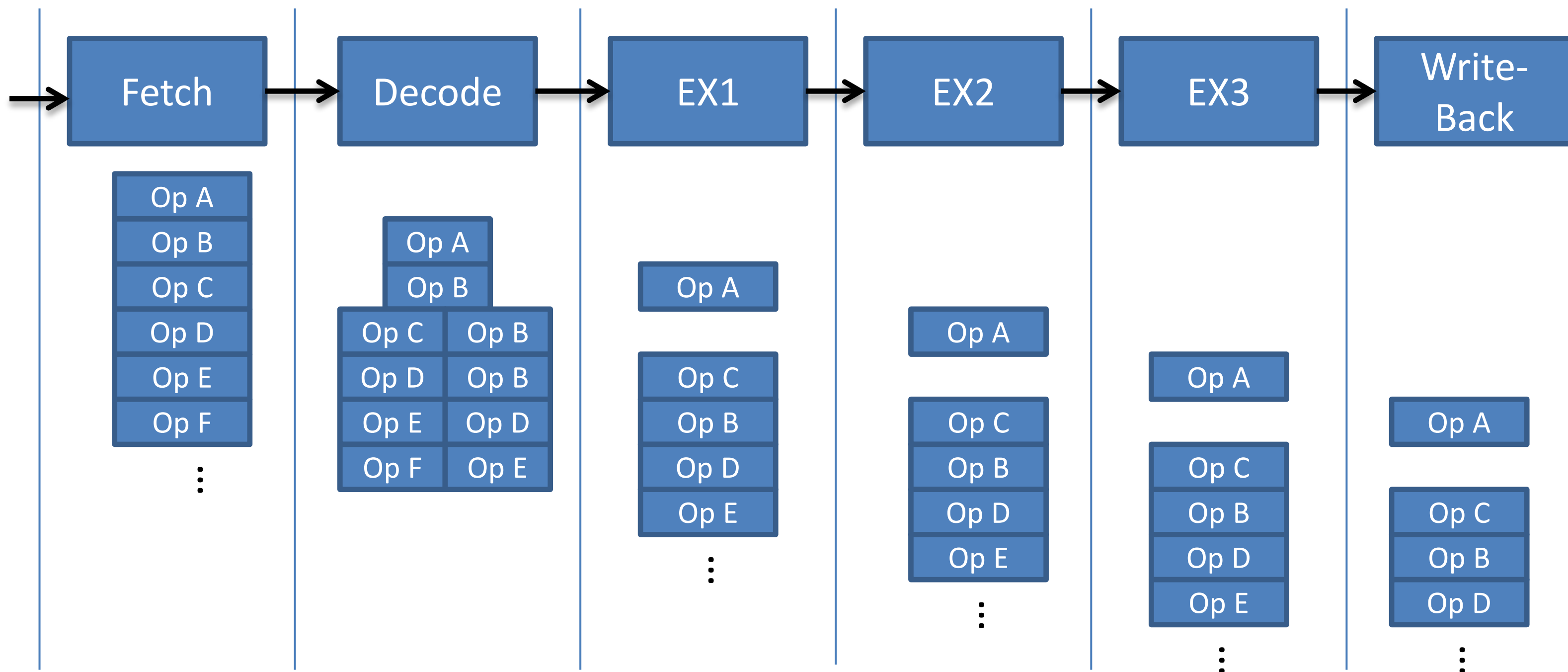
Dynamische Ablaufplanung (1)

- Skalarer Datenpfad mit Datenpfadpipeline
  - out-of-order Ausgabe mit max. einer Instruktion pro Takt
  - in-order Abschluss mit max. einer Instruktion pro Takt
- CPI < 1 so nicht möglich, da pro Takt höchstens eine Operation geholt und fertiggestellt wird



Dynamische Ablaufplanung (2)

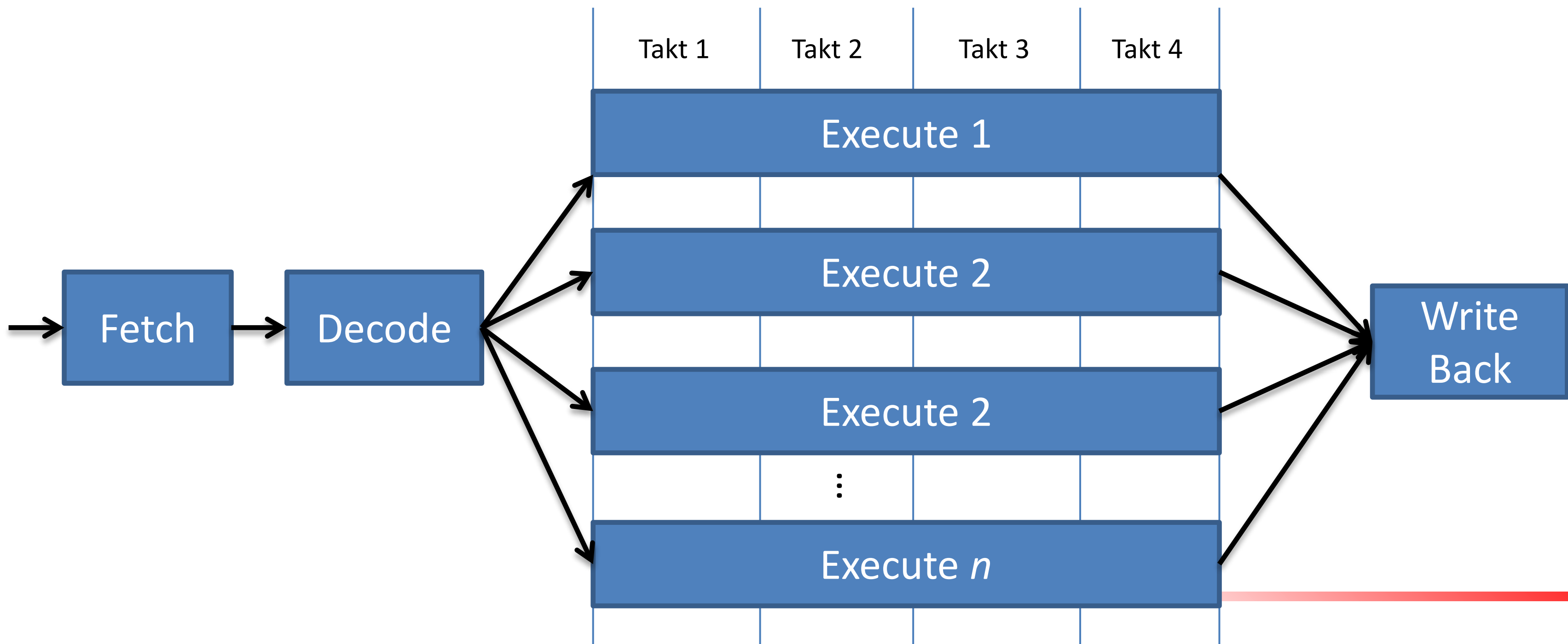
- Skalarer Datenpfad mit Datenpfadpipeline
  - out-of-order Ausgabe mit max. einer Instruktion pro Takt
  - out-of-order Abschluss mit max. einer Instruktion pro Takt
- CPI < 1 so nicht möglich, da pro Takt höchstens eine Operation geholt und fertiggestellt wird



Superskalarer Datenpfad

Mehrere Ausführungseinheiten im Datenpfad arbeiten parallel

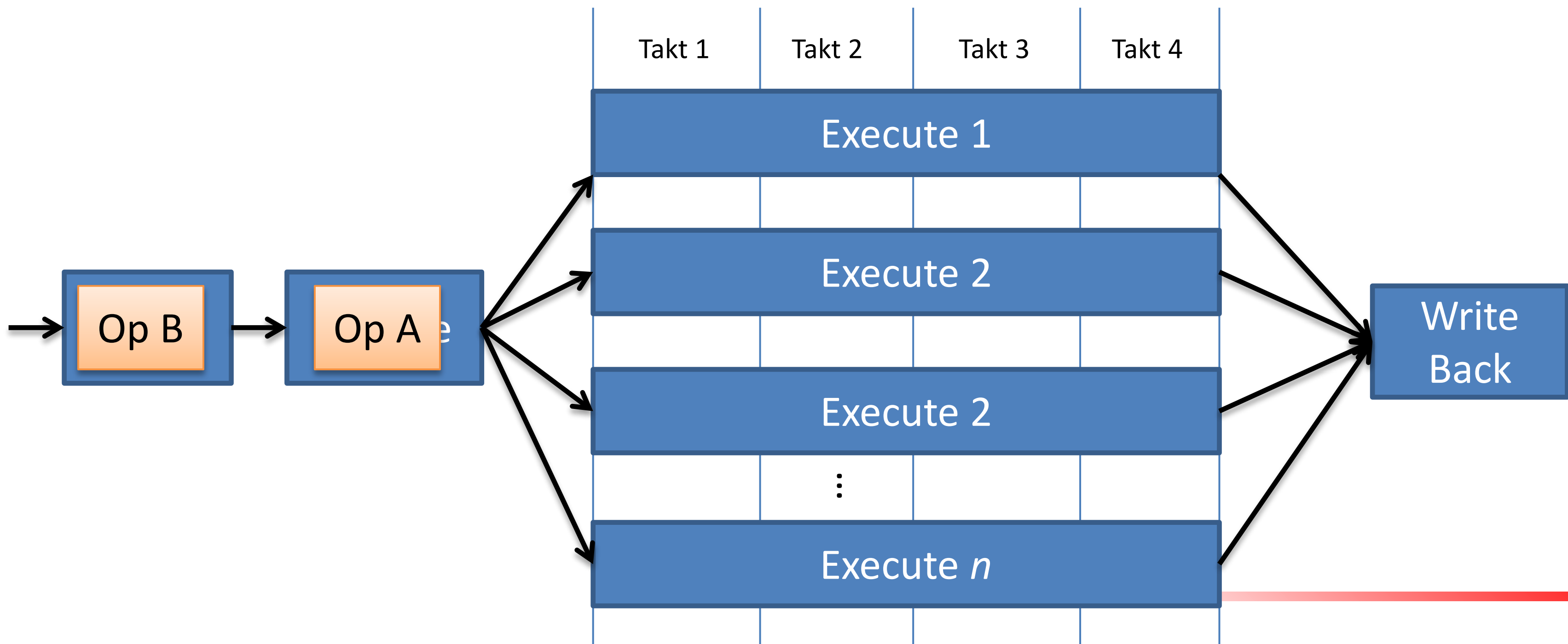
- Ausführungseinheiten sind nicht gepipelined
- Gleich lange Ausführungsdauer in allen Ausführungseinheiten
  - bei in-order Ausgabe in-order Ausführung und Abschluss
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben
- Konzeptionell identisch mit skalarem Datenpfad mit n-stufiger Datenpfad-Pipeline



Superskalarer Datenpfad

Mehrere Ausführungseinheiten im Datenpfad arbeiten parallel

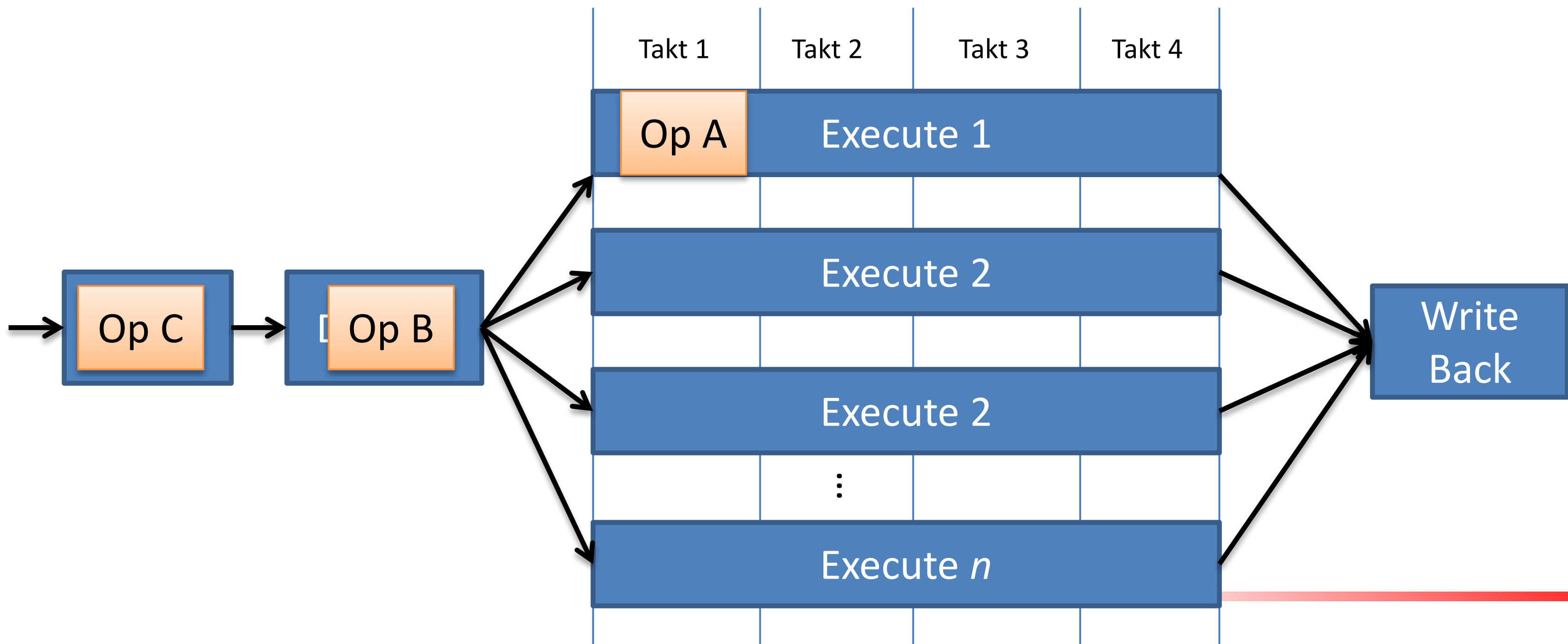
- Ausführungseinheiten sind nicht gepipelined
- Gleich lange Ausführungsdauer in allen Ausführungseinheiten
  - bei in-order Ausgabe in-order Ausführung und Abschluss
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben
- Konzeptionell identisch mit skalarem Datenpfad mit n-stufiger Datenpfad-Pipeline



Superskalarer Datenpfad

Mehrere Ausführungseinheiten im Datenpfad arbeiten parallel

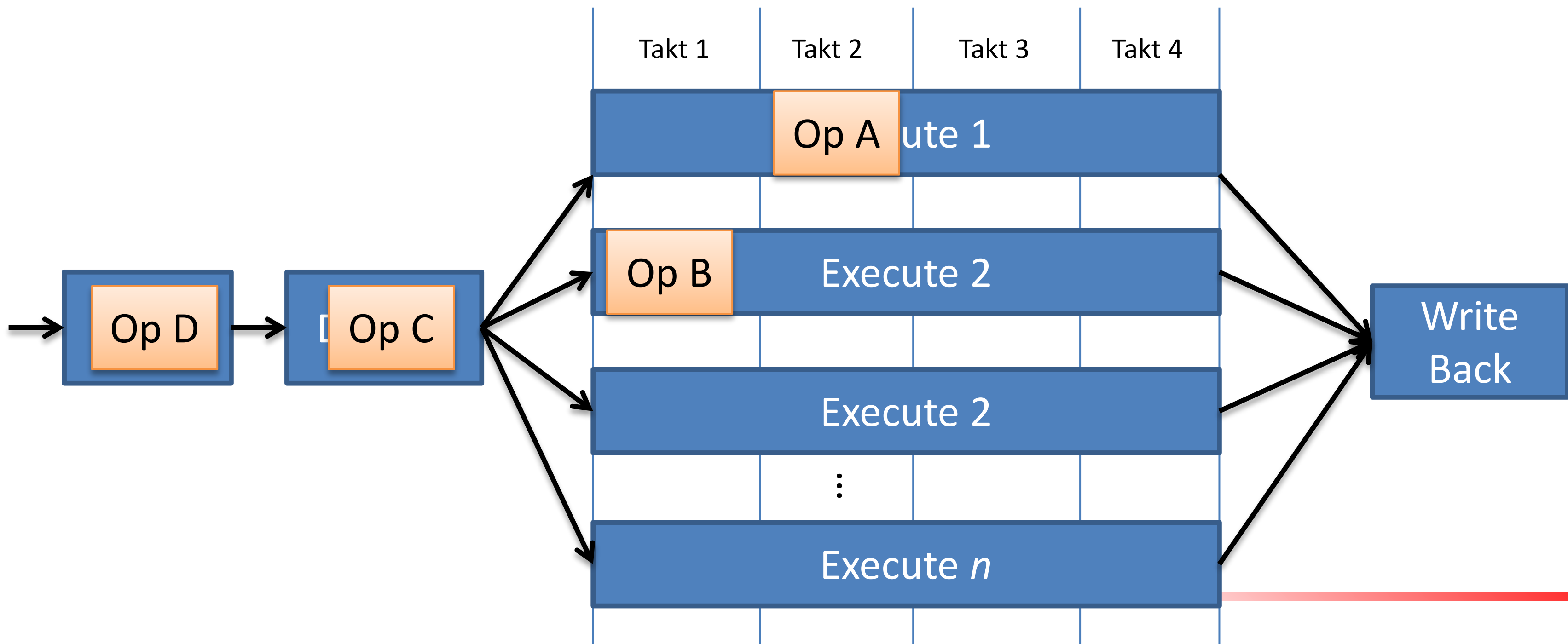
- Ausführungseinheiten sind nicht gepipelined
- Gleich lange Ausführungsdauer in allen Ausführungseinheiten
  - bei in-order Ausgabe in-order Ausführung und Abschluss
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben
- Konzeptionell identisch mit skalarem Datenpfad mit n-stufiger Datenpfad-Pipeline



Superskalarer Datenpfad

Mehrere Ausführungseinheiten im Datenpfad arbeiten parallel

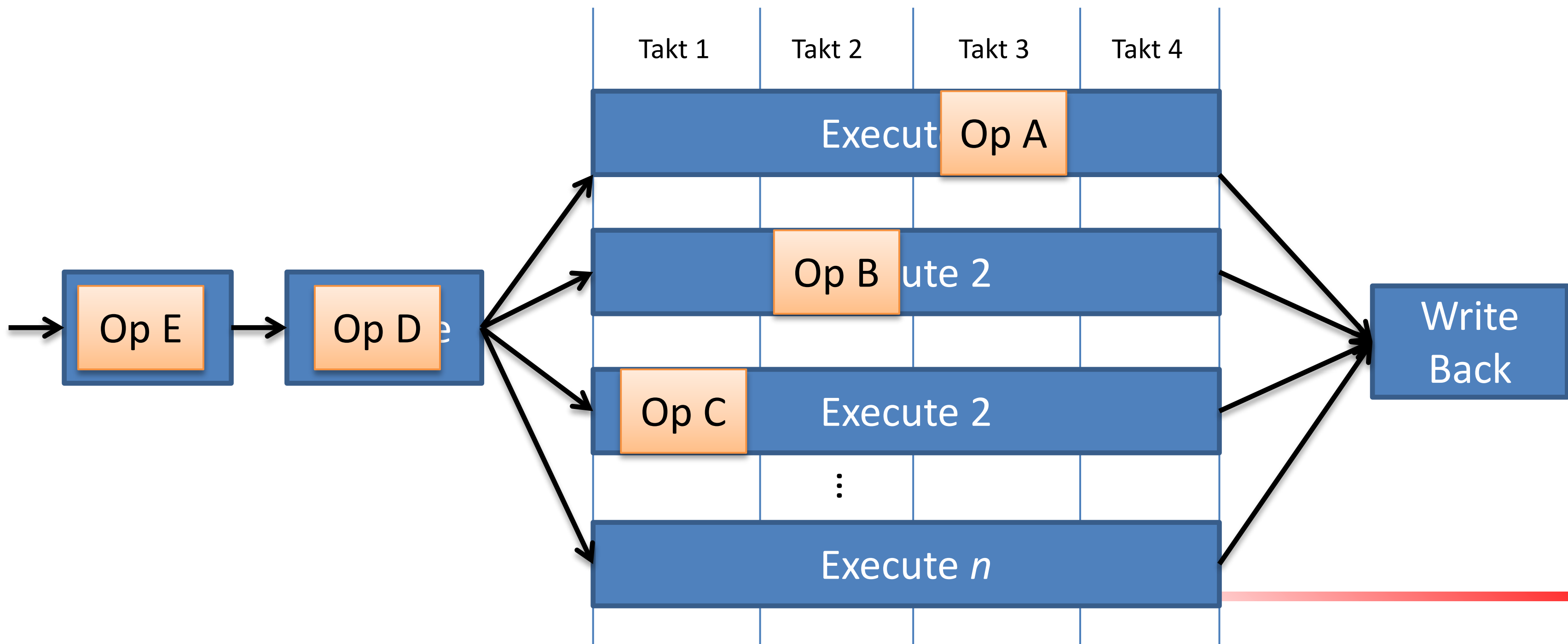
- Ausführungseinheiten sind nicht gepipelined
- Gleich lange Ausführungsdauer in allen Ausführungseinheiten
  - bei in-order Ausgabe in-order Ausführung und Abschluss
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben
- Konzeptionell identisch mit skalarem Datenpfad mit n-stufiger Datenpfad-Pipeline



Superskalarer Datenpfad

Mehrere Ausführungseinheiten im Datenpfad arbeiten parallel

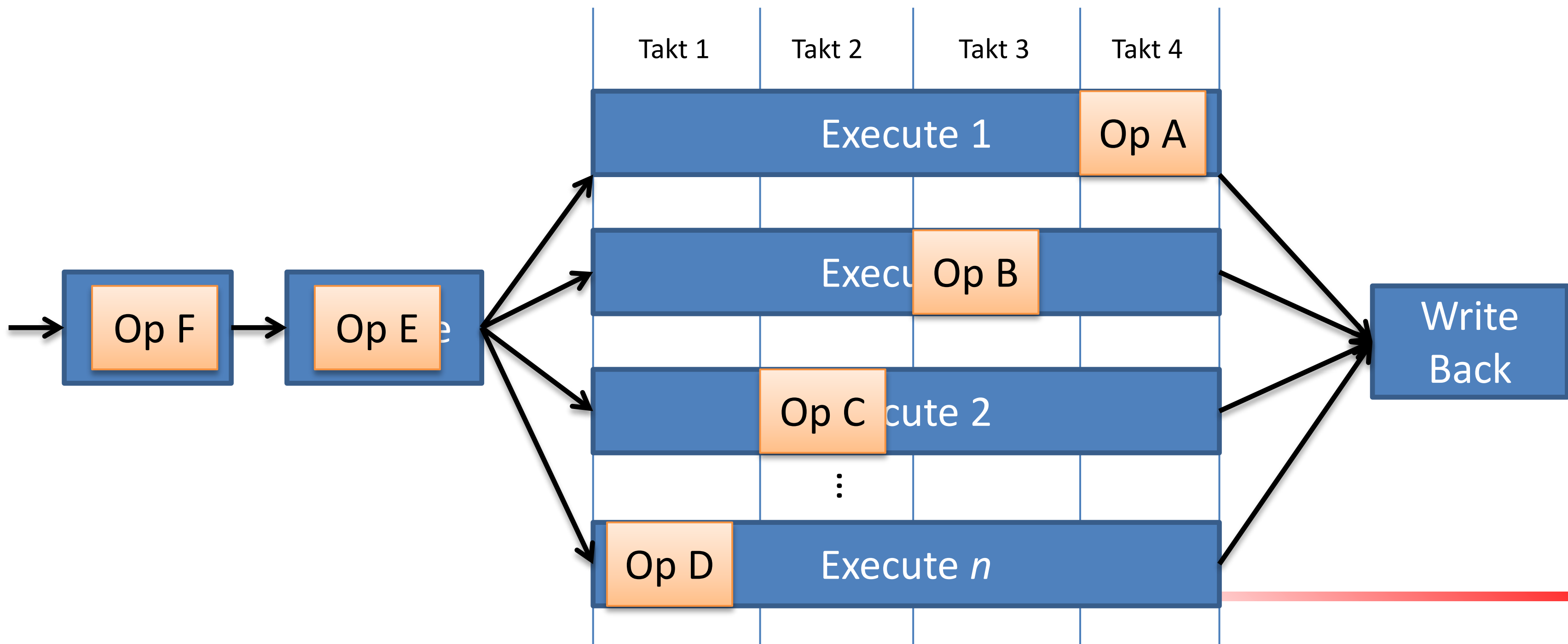
- Ausführungseinheiten sind nicht gepipelined
- Gleich lange Ausführungsdauer in allen Ausführungseinheiten
  - bei in-order Ausgabe in-order Ausführung und Abschluss
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben
- Konzeptionell identisch mit skalarem Datenpfad mit n-stufiger Datenpfad-Pipeline



Superskalarer Datenpfad

Mehrere Ausführungseinheiten im Datenpfad arbeiten parallel

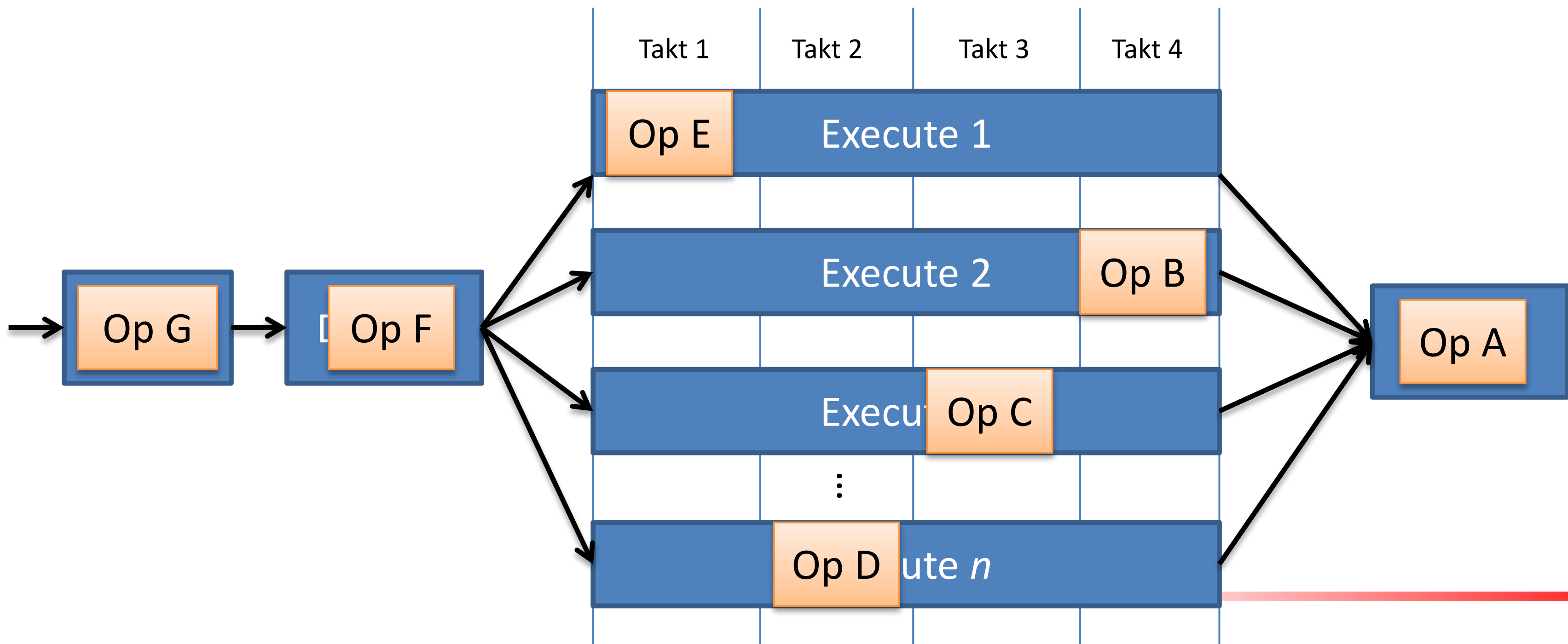
- Ausführungseinheiten sind nicht gepipelined
- Gleich lange Ausführungsdauer in allen Ausführungseinheiten
  - bei in-order Ausgabe in-order Ausführung und Abschluss
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben
- Konzeptionell identisch mit skalarem Datenpfad mit n-stufiger Datenpfad-Pipeline



Superskalarer Datenpfad

Mehrere Ausführungseinheiten im Datenpfad arbeiten parallel

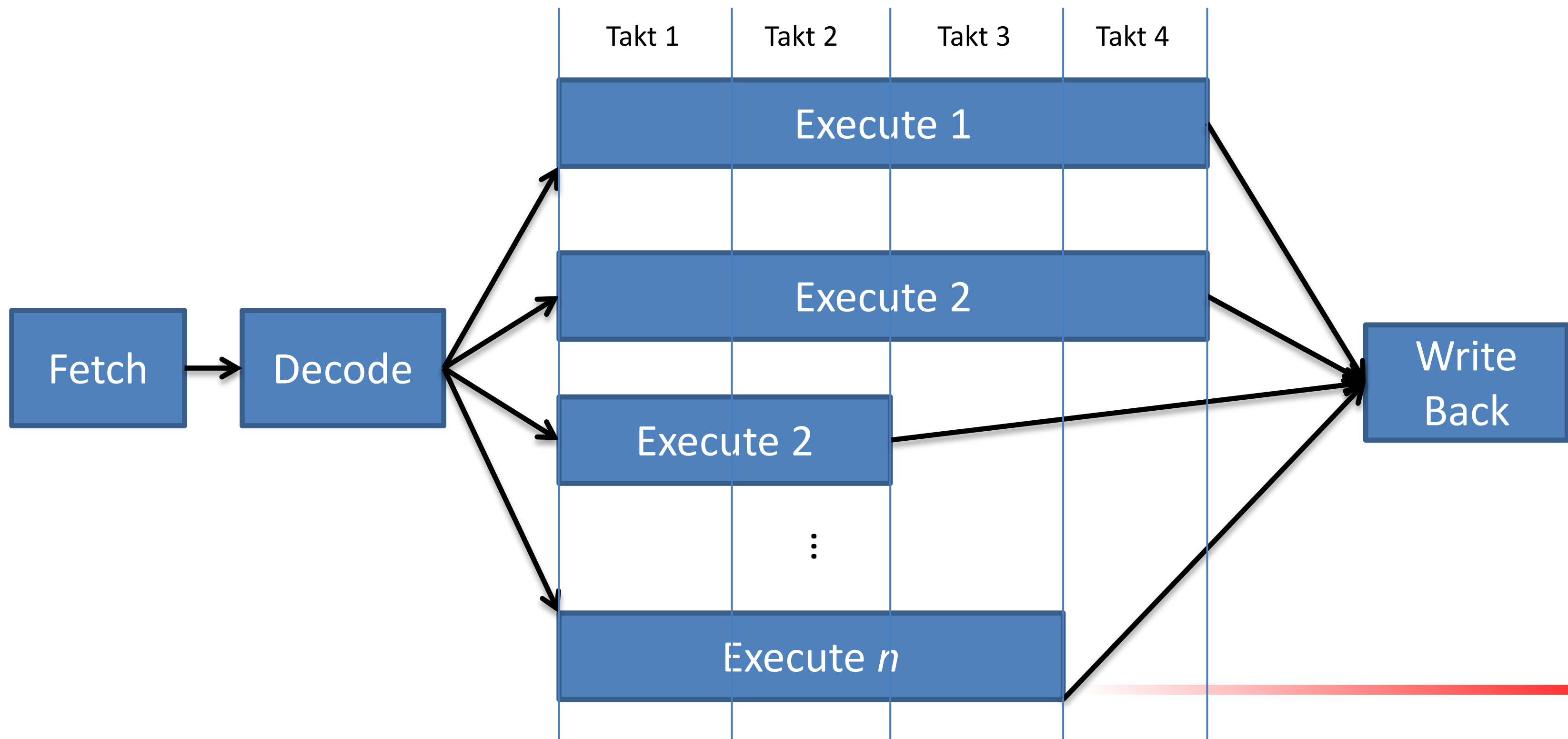
- Ausführungseinheiten sind nicht gepipelined
- Gleich lange Ausführungsdauer in allen Ausführungseinheiten
  - bei in-order Ausgabe in-order Ausführung und Abschluss
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben
- Konzeptionell identisch mit skalarem Datenpfad mit n-stufiger Datenpfad-Pipeline



Superskalarer Datenpfad

Mehrere Ausführungseinheiten, die parallel arbeiten können

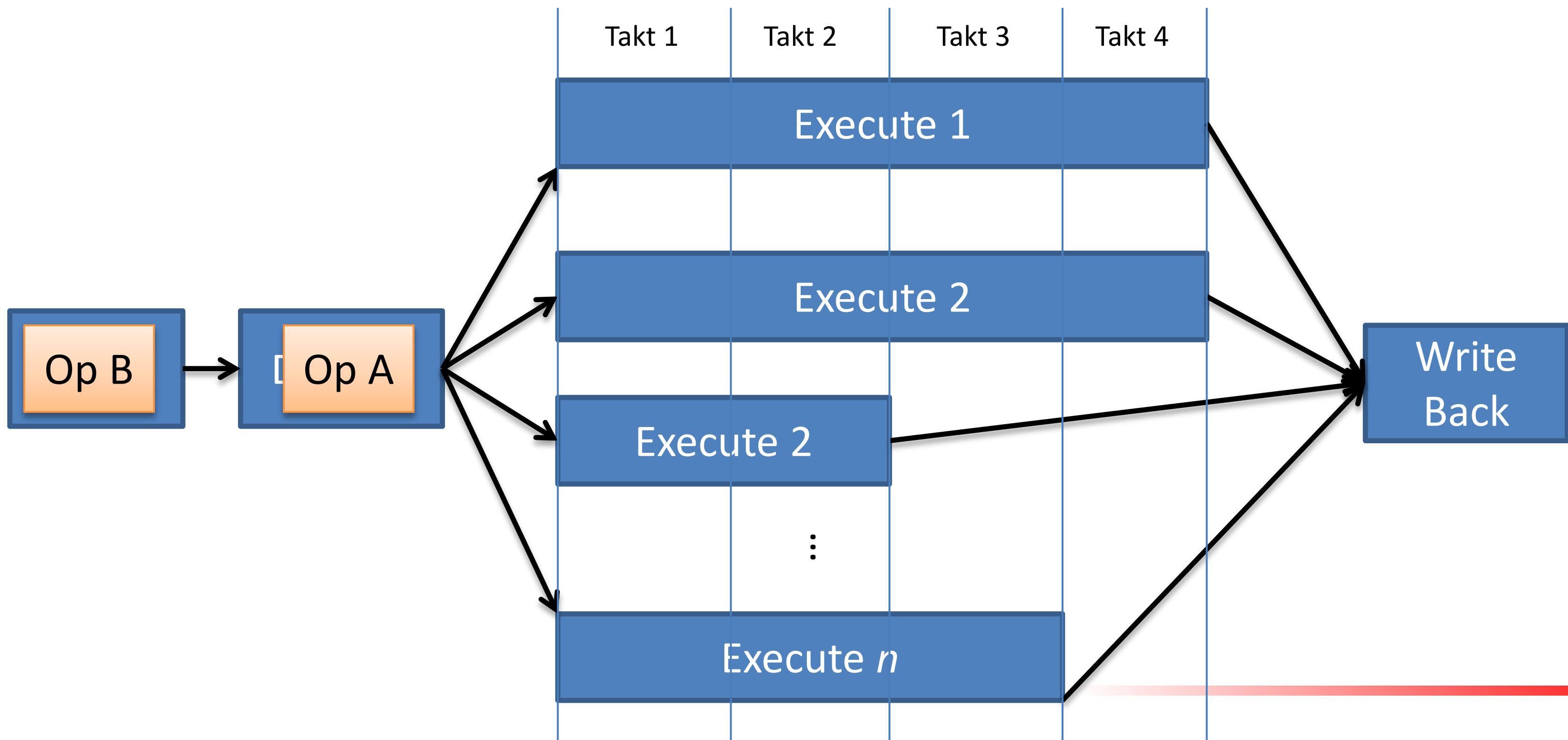
- Ausführungseinheiten nicht gepipelined
- Unterschiedlich lange Ausführungsdauer möglich
  - bei in-order-Ausgabe ist out-of-order Verarbeitung möglich
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben



Superskalarer Datenpfad

Mehrere Ausführungseinheiten, die parallel arbeiten können

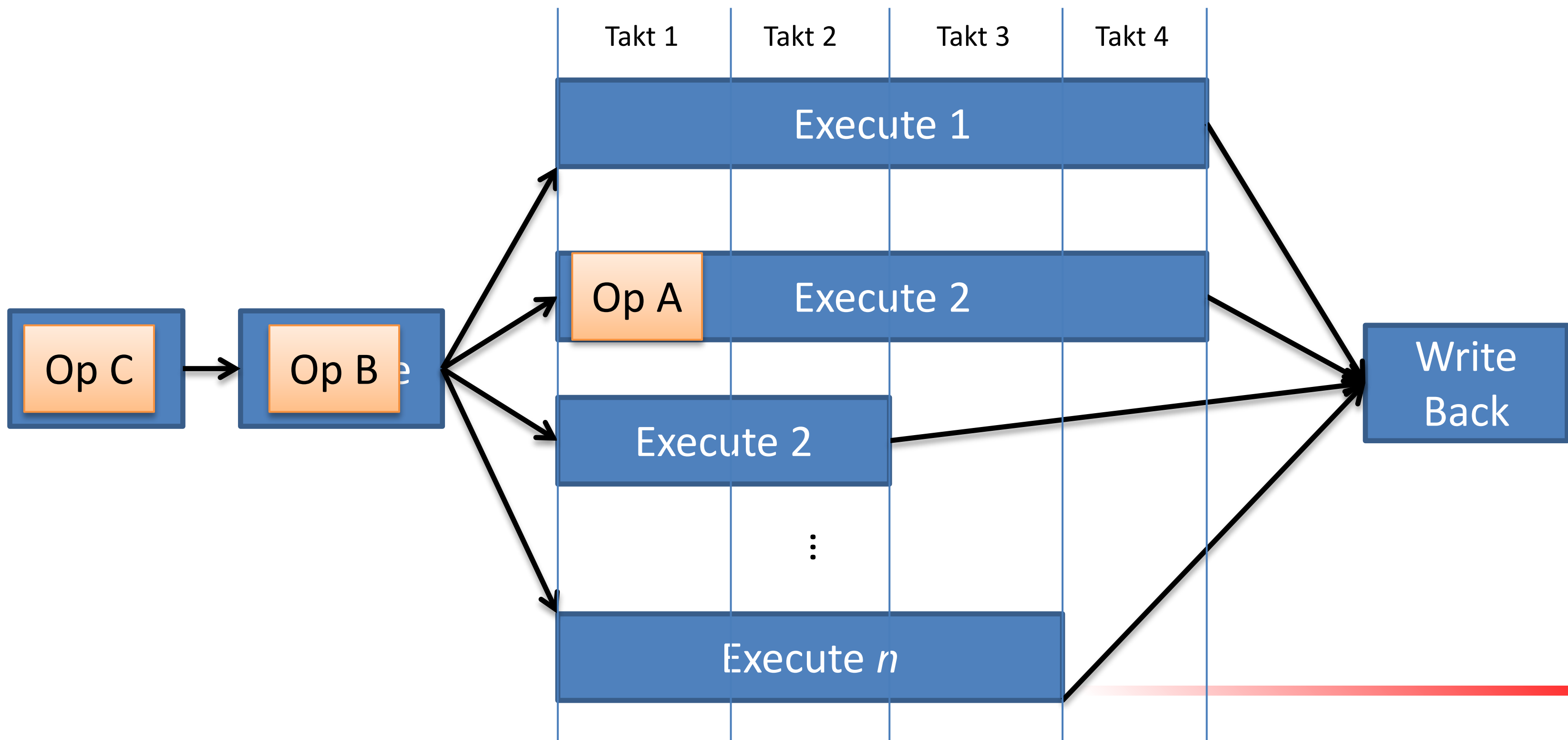
- Ausführungseinheiten nicht gepipelined
- Unterschiedlich lange Ausführungsdauer möglich
  - bei in-order-Ausgabe ist out-of-order Verarbeitung möglich
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben



Superskalarer Datenpfad

Mehrere Ausführungseinheiten, die parallel arbeiten können

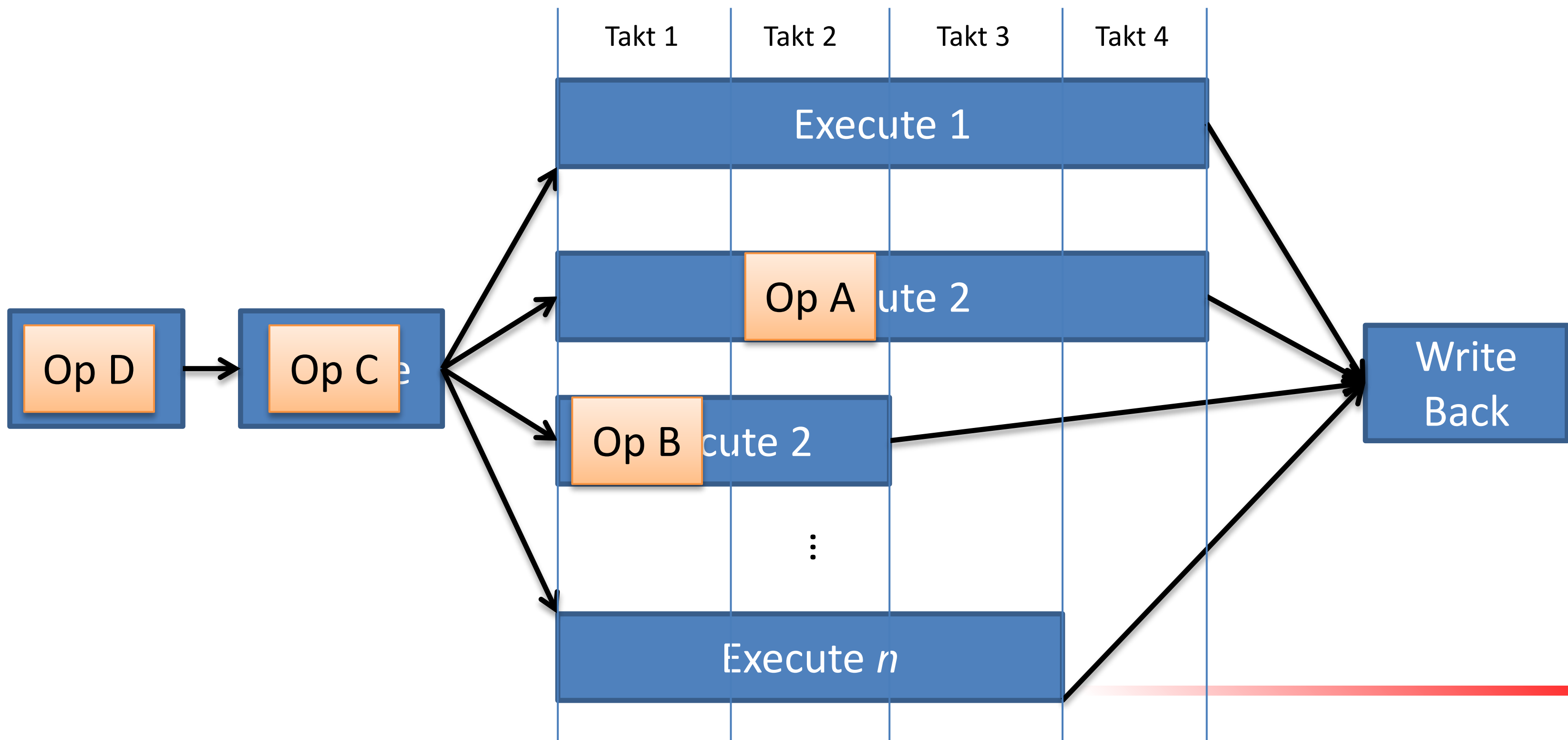
- Ausführungseinheiten nicht gepipelined
- Unterschiedlich lange Ausführungsdauer möglich
  - bei in-order-Ausgabe ist out-of-order Verarbeitung möglich
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben



Superskalarer Datenpfad

Mehrere Ausführungseinheiten, die parallel arbeiten können

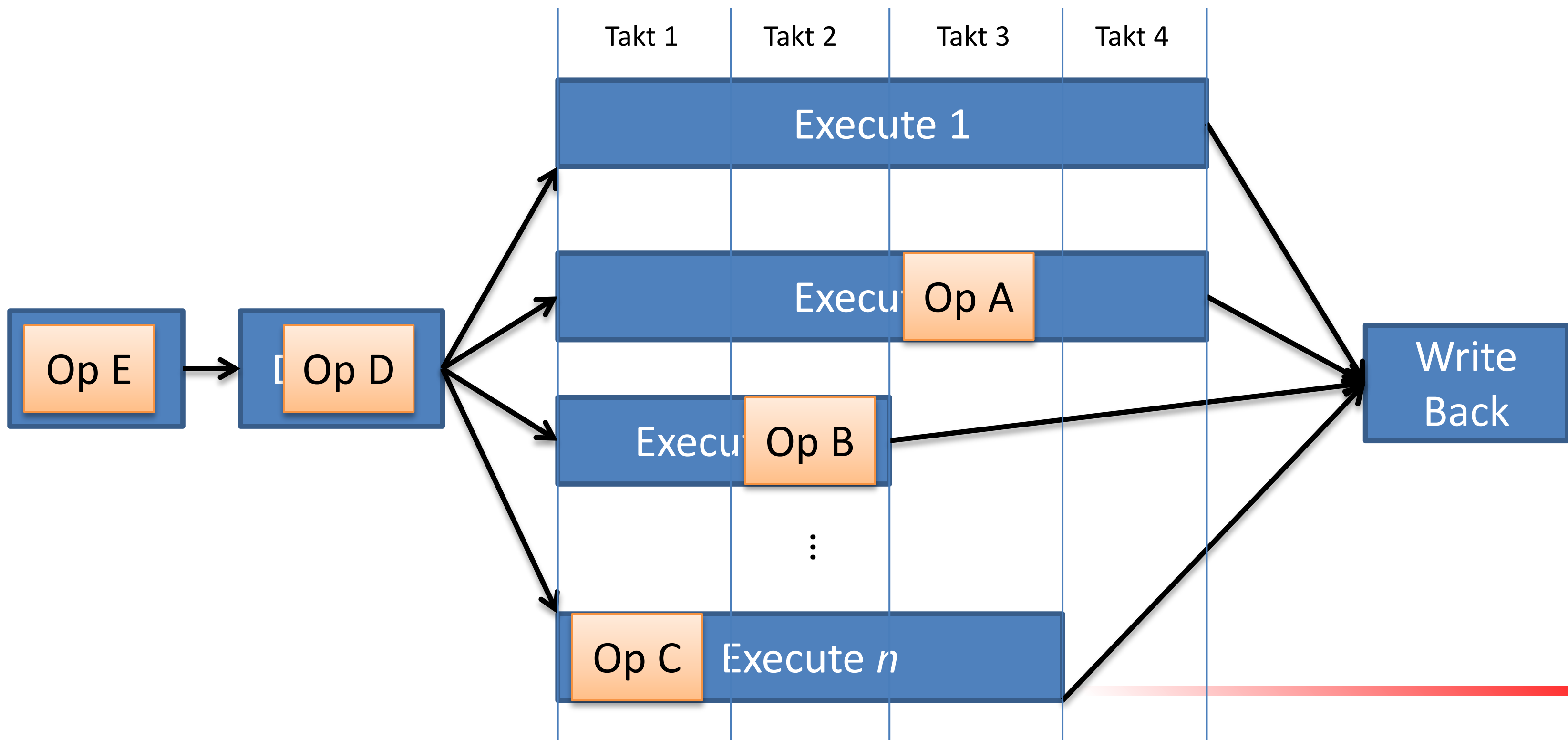
- Ausführungseinheiten nicht gepipelined
- Unterschiedlich lange Ausführungsdauer möglich
  - bei in-order-Ausgabe ist out-of-order Verarbeitung möglich
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben



Superskalarer Datenpfad

Mehrere Ausführungseinheiten, die parallel arbeiten können

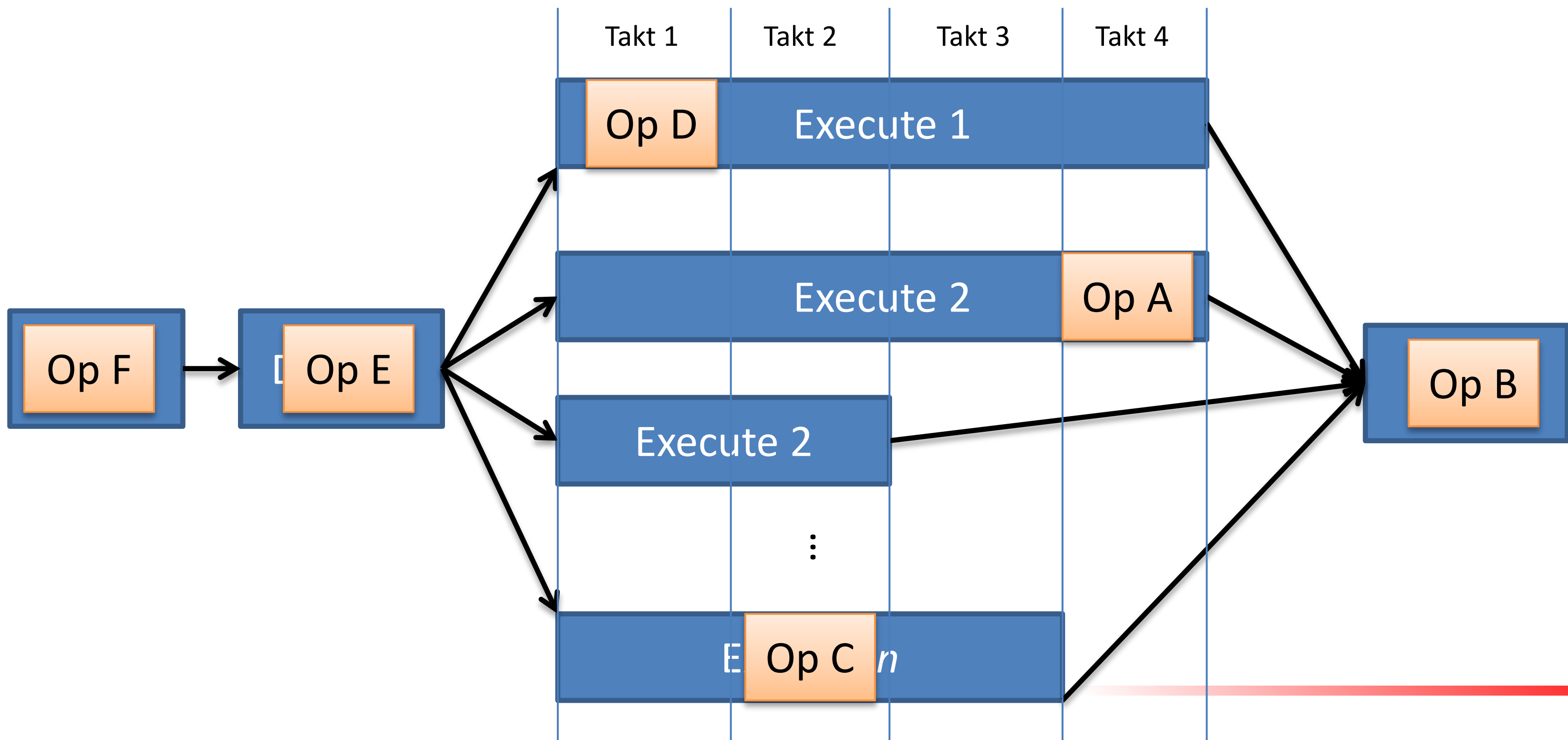
- Ausführungseinheiten nicht gepipelined
- Unterschiedlich lange Ausführungsdauer möglich
  - bei in-order-Ausgabe ist out-of-order Verarbeitung möglich
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben



Superskalarer Datenpfad

Mehrere Ausführungseinheiten, die parallel arbeiten können

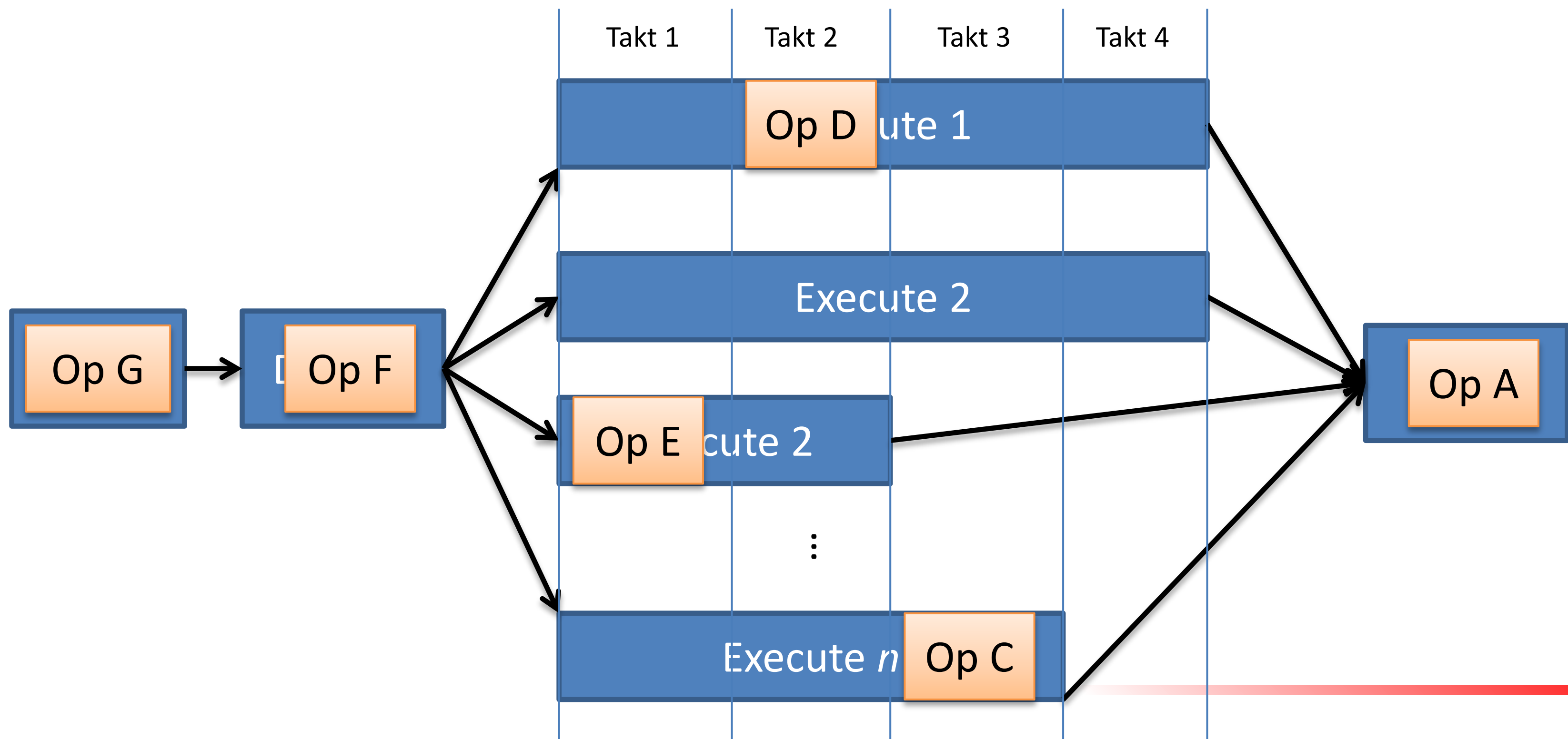
- Ausführungseinheiten nicht gepipelined
- Unterschiedlich lange Ausführungsdauer möglich
  - bei in-order-Ausgabe ist out-of-order Verarbeitung möglich
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben



Superskalarer Datenpfad

Mehrere Ausführungseinheiten, die parallel arbeiten können

- Ausführungseinheiten nicht gepipelined
- Unterschiedlich lange Ausführungsdauer möglich
  - bei in-order-Ausgabe ist out-of-order Verarbeitung möglich
- CPI < 1 nicht möglich, da Ausgabe und Abschluss nur einen Durchsatz von einem Befehl pro Takt haben

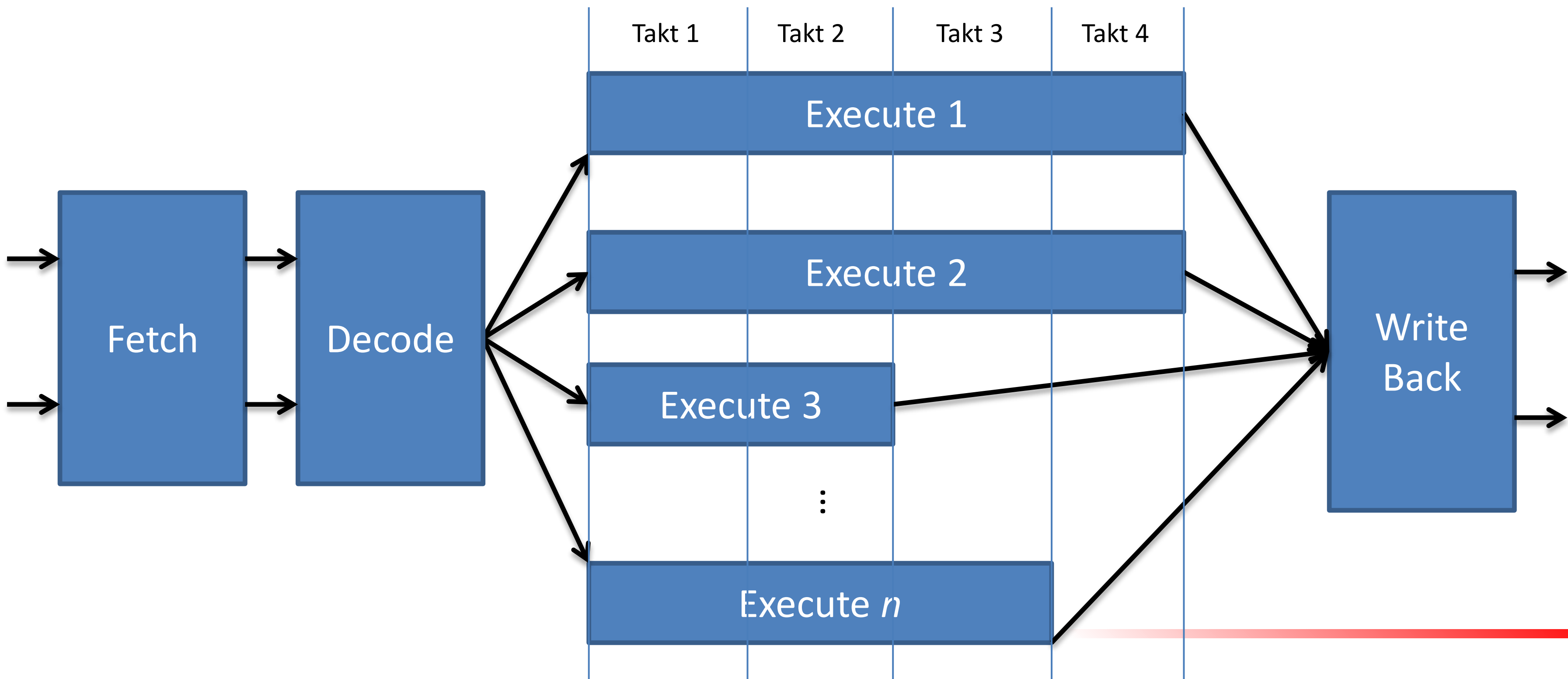


n-fach Superskalar Prozessor

- Jede Stufe der Befehlspipeline (insbesondere Fetch, Ausgabe und Abschluss) können bis zu *n* Befehle pro Takt verarbeiten
- CPI < 1 möglich

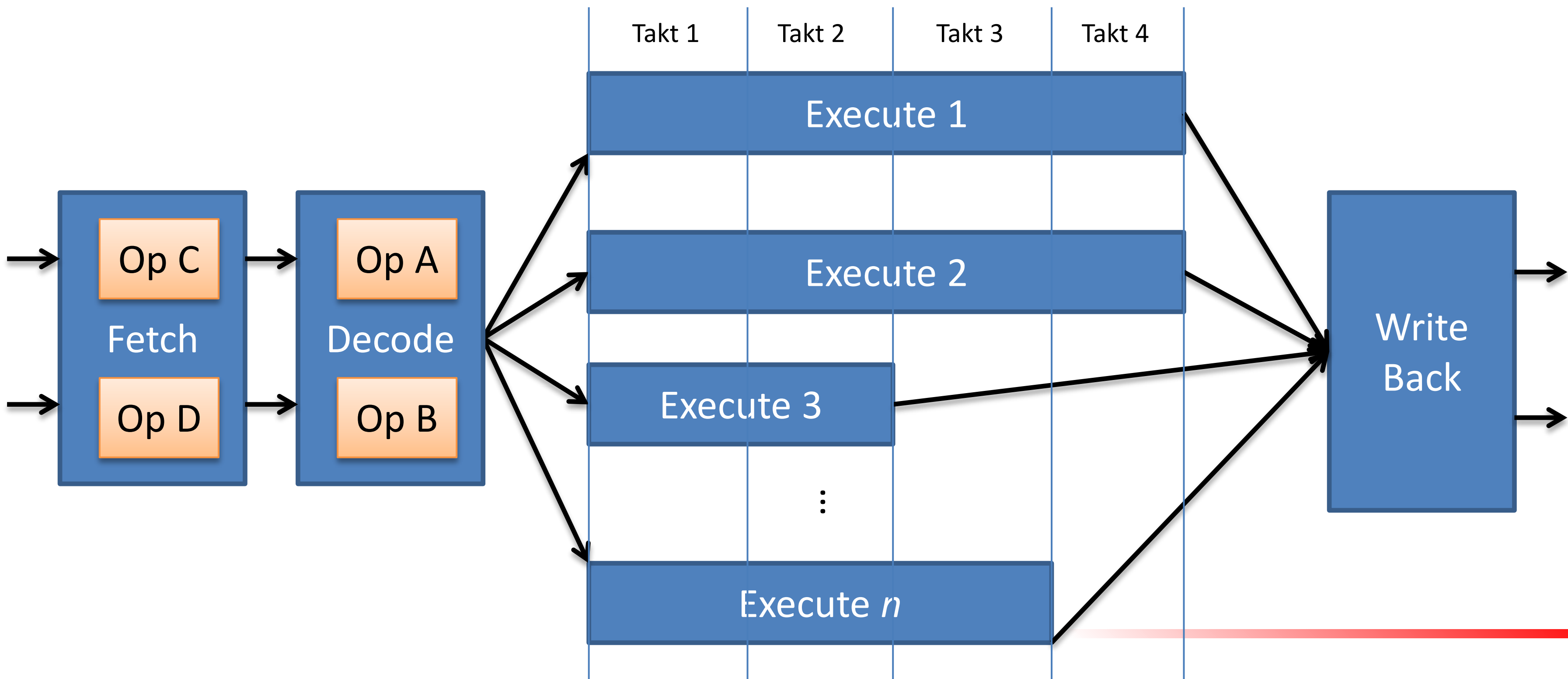
n-fach Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung: Befehle müssen aufeinander warten; Ausführungseinheiten werden blockiert
  - out-of-order Befehlsausführung



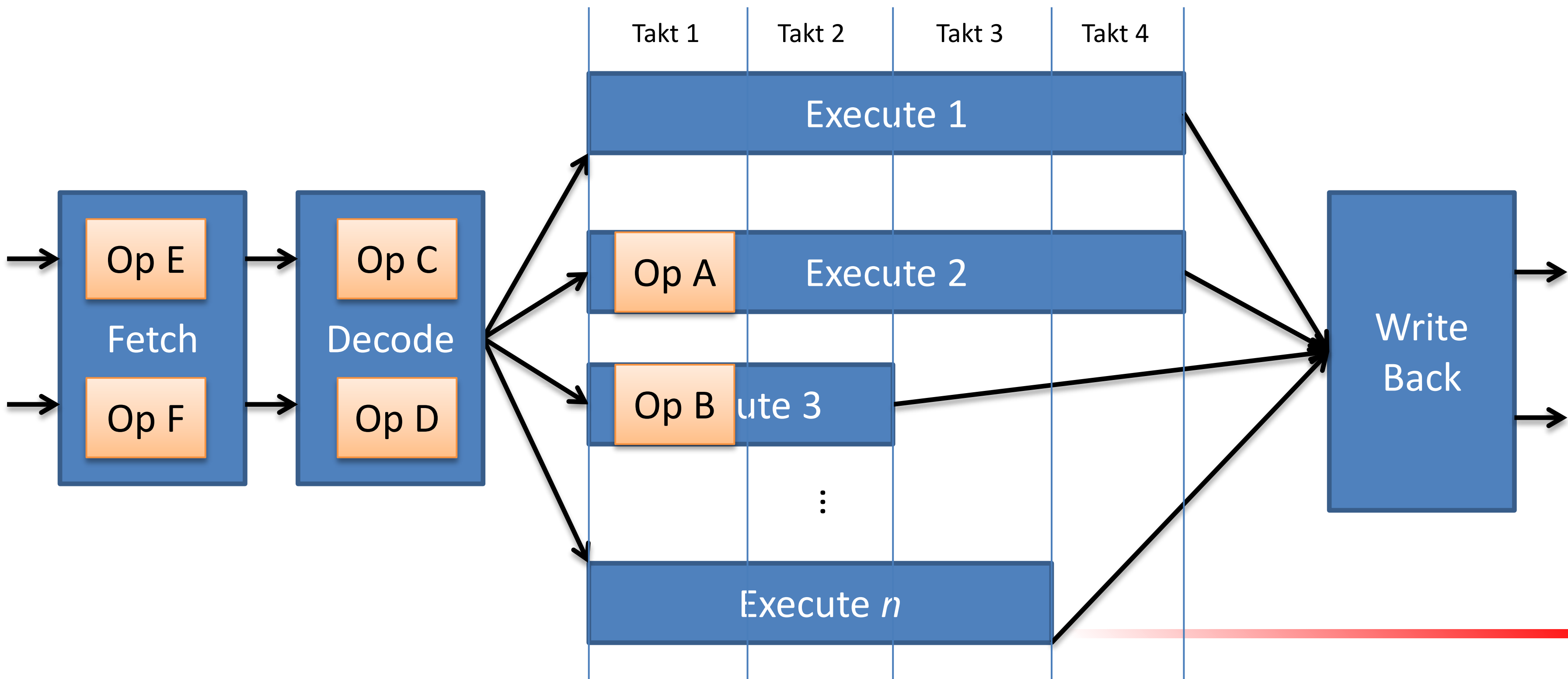
n-fach Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung: Befehle müssen aufeinander warten; Ausführungseinheiten werden blockiert
  - out-of-order Befehlsausführung



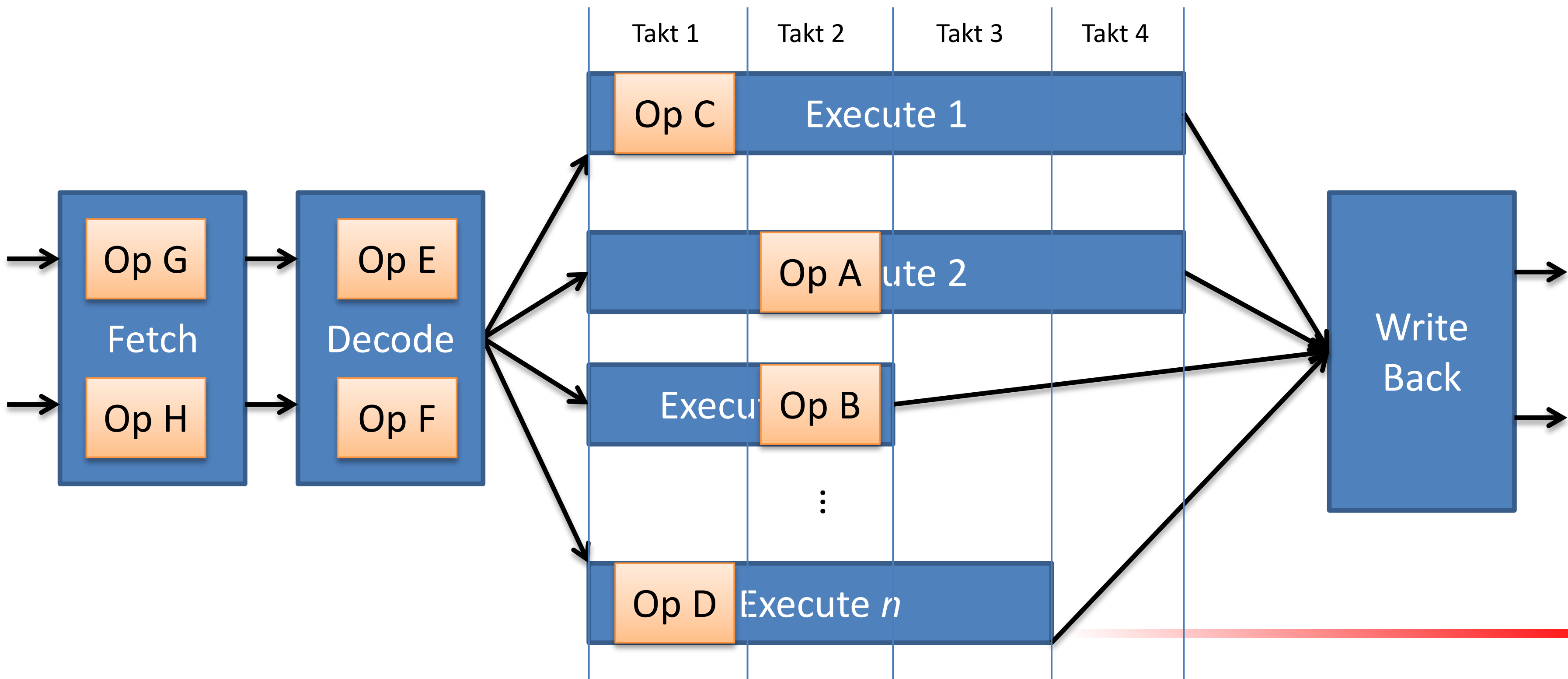
n-fach Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung: Befehle müssen aufeinander warten; Ausführungseinheiten werden blockiert
  - out-of-order Befehlsausführung



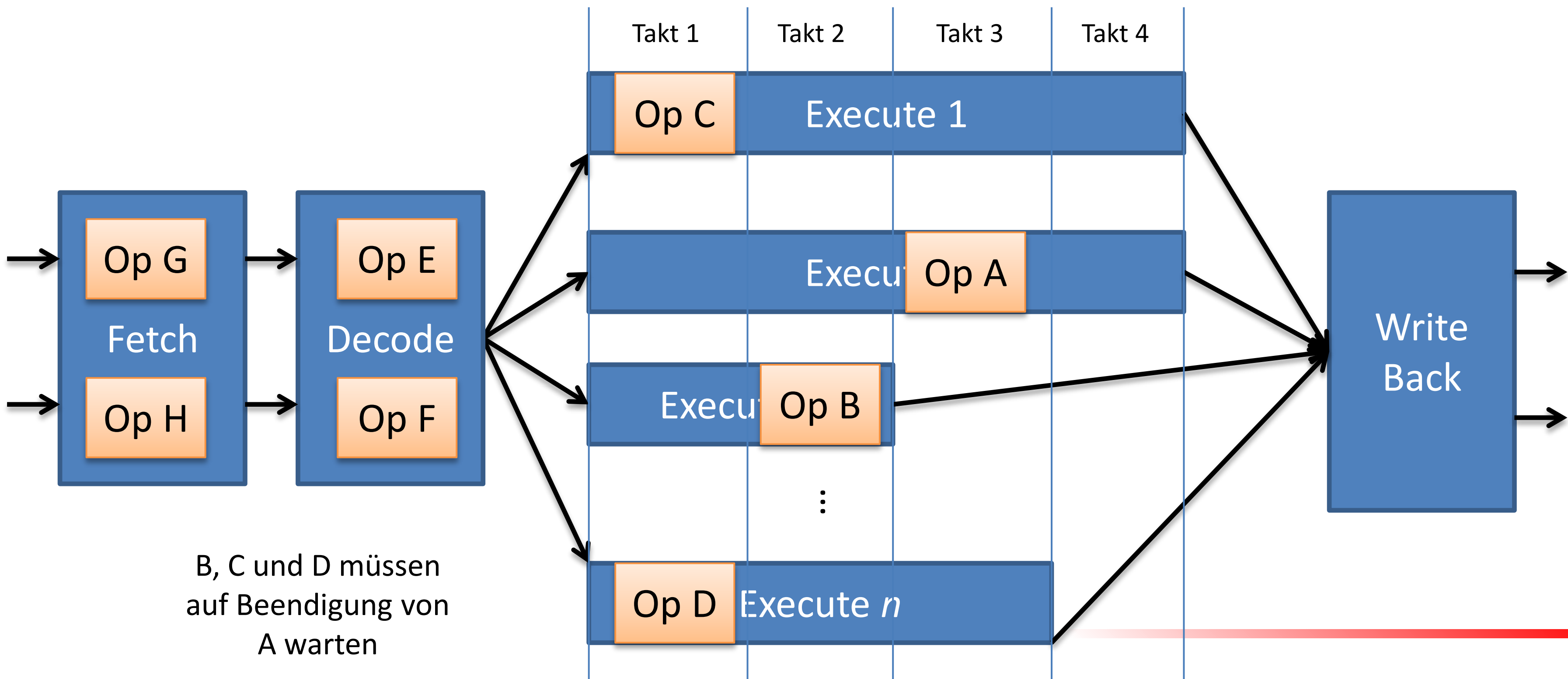
n-fach Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung: Befehle müssen aufeinander warten; Ausführungseinheiten werden blockiert
  - out-of-order Befehlsausführung



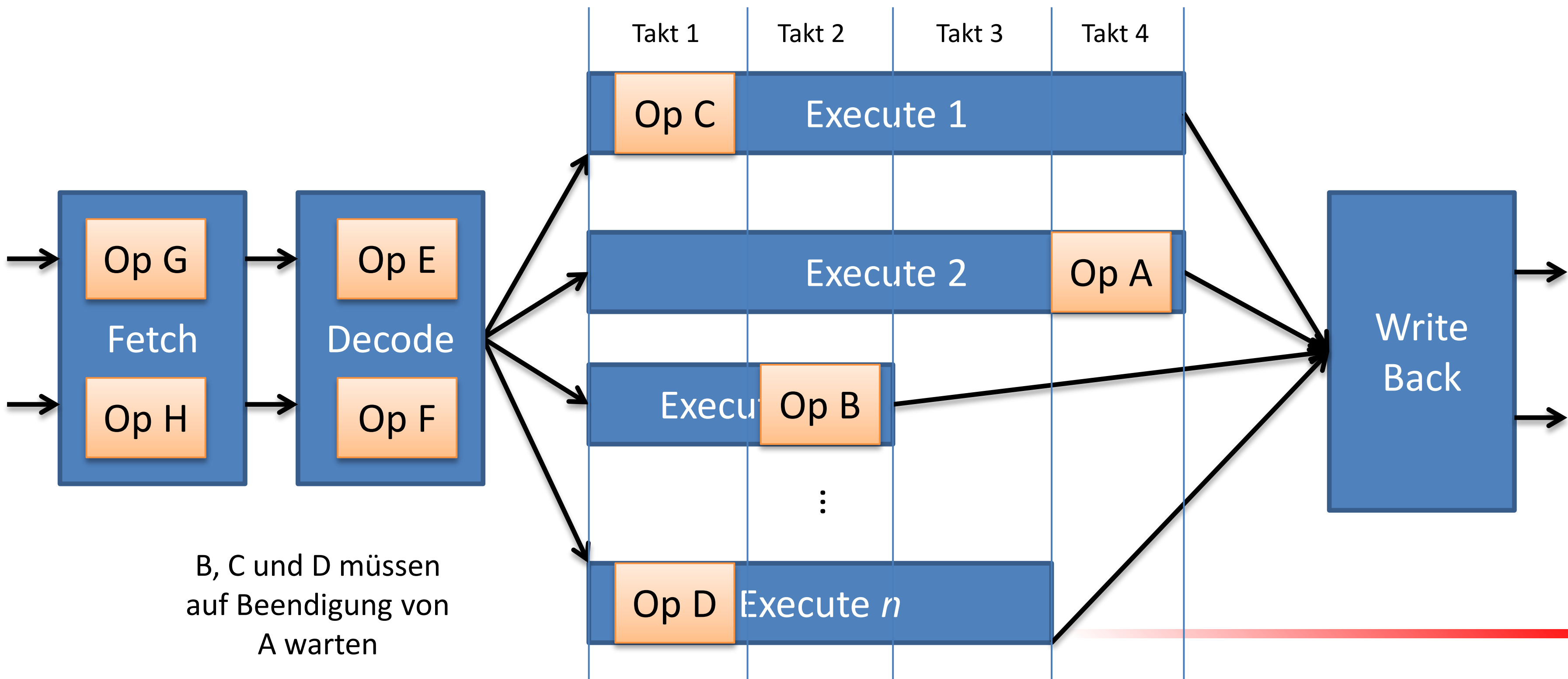
n-fach Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung: Befehle müssen aufeinander warten; Ausführungseinheiten werden blockiert
  - out-of-order Befehlsausführung



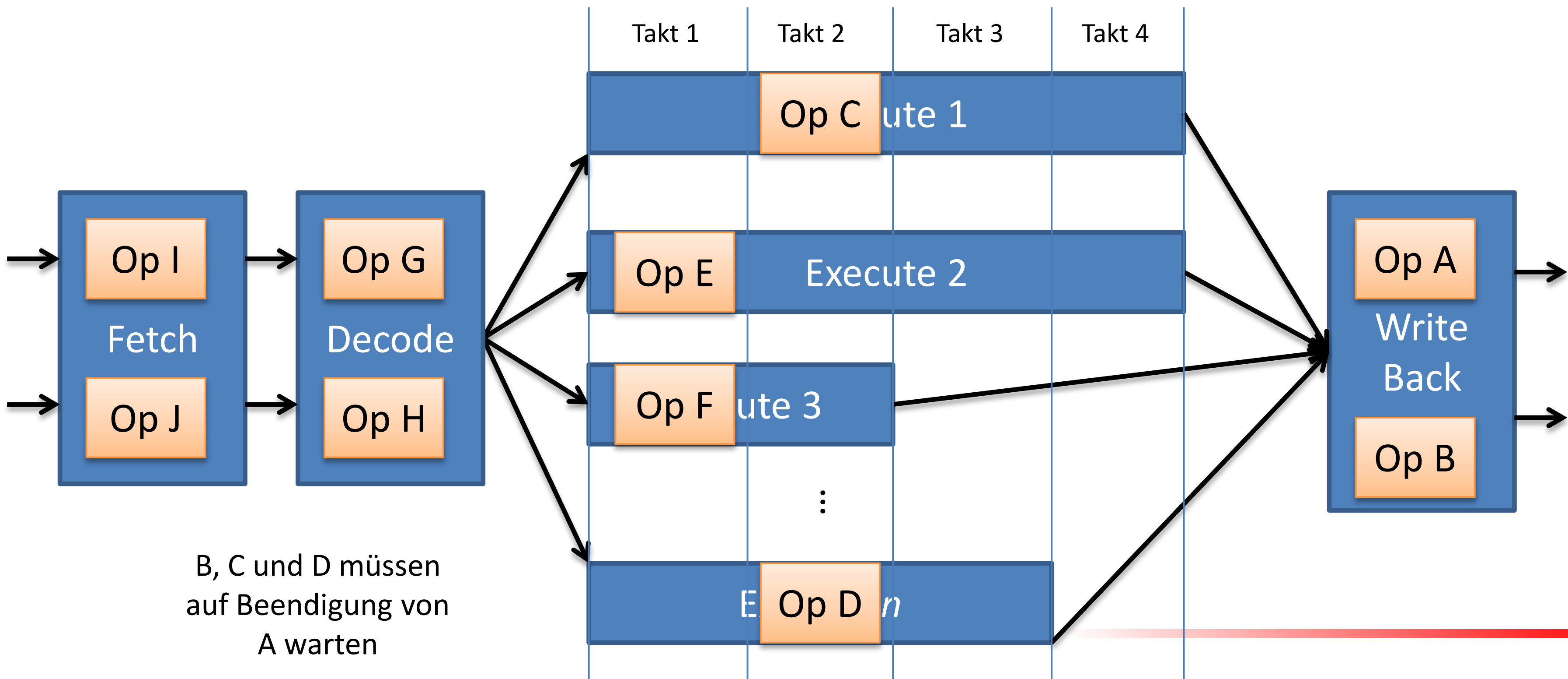
n-fach Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung: Befehle müssen aufeinander warten; Ausführungseinheiten werden blockiert
  - out-of-order Befehlsausführung



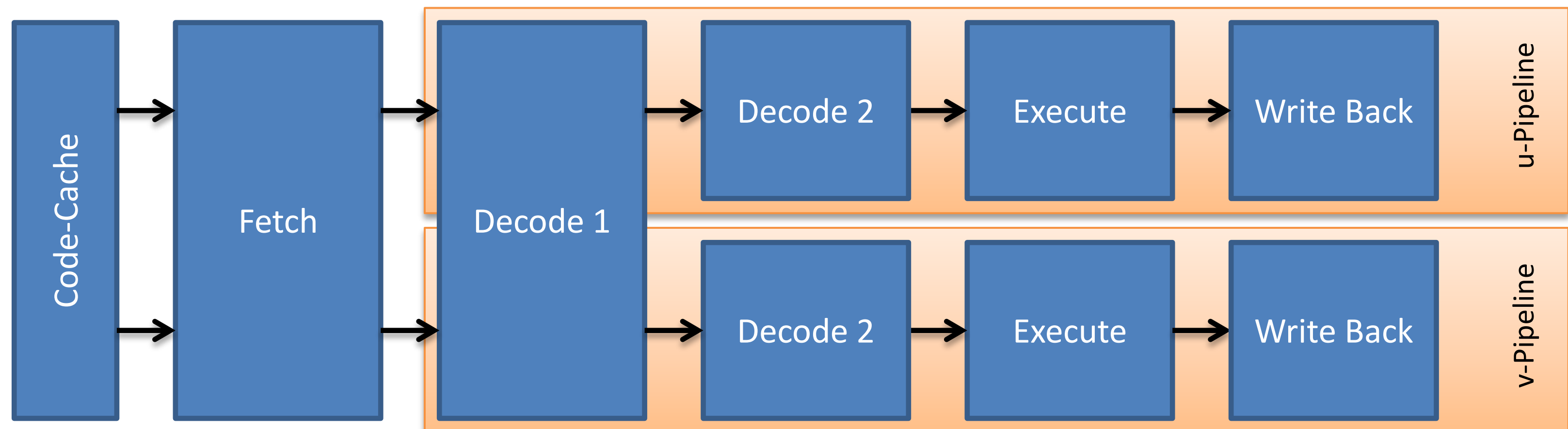
n-fach Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung: Befehle müssen aufeinander warten; Ausführungseinheiten werden blockiert
  - out-of-order Befehlsausführung



Beispiel Intel Pentium-Prozessor

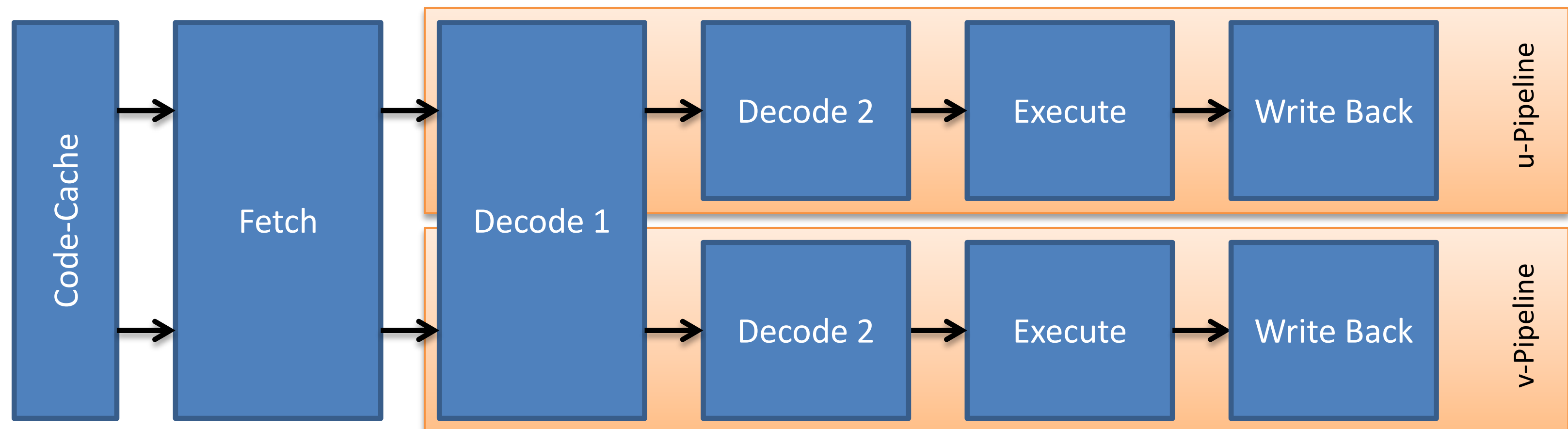
- Zwei parallele Integer-Pipelines
- Befehle werden – wenn möglich – paarweise ausgeführt
- Durchlaufen gemeinsam jede Pipelinestufe
- Schnellere Befehle müssen auf langsamere Befehle warten



5-stufige Integer-Pipeline des Pentium-Prozessors

Beispiel Intel Pentium-Pipeline

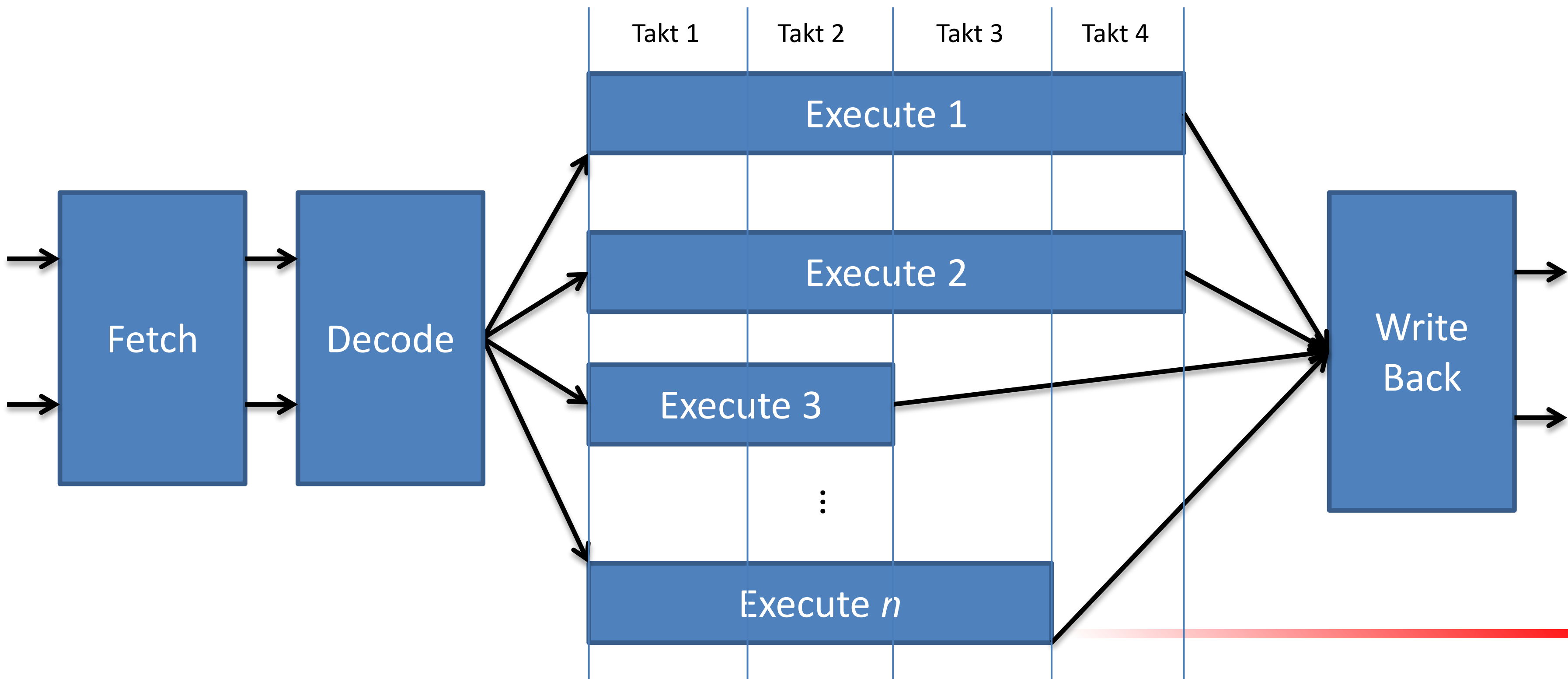
- Prefetch: Holt bis zu zwei Befehle aus dem Code Cache
- Decode 1: Prüft ob Befehlspaar parallel abgearbeitet werden kann
  - Ja: Erste Befehl in u-Pipeline und zweite Befehl in v-Pipeline
  - Nein: Erste Befehl in u-Pipeline und zweiter Befehl wird mit nächstem geholten Befehl kombiniert
- Decode 2: Berechnet Adressen für Speicheroperanden
- Execute: Führt ALU-Operationen und erforderliche Speicherzugriffe aus
  - holt Operanden bei Bedarf aus Speicher
  - schnellere Operation in u-Pipeline kann diese Stufe vor Beendigung der anderen Operation verlassen
- Write-Back: Zurückschreiben der Ergebnisse in Prozessorregister (in-order)



5-stufige Integer-Pipeline des Pentium-Prozessors

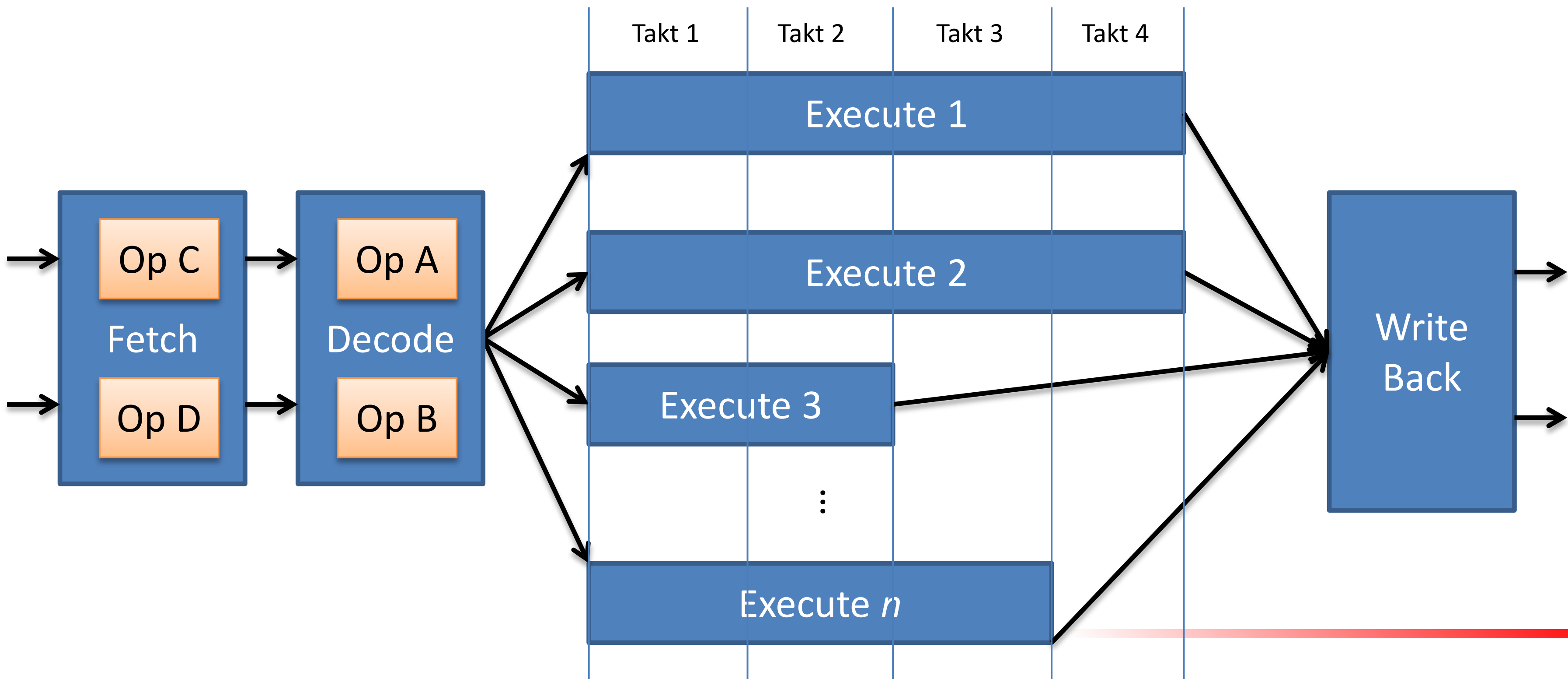
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung und/oder out-of-order Befehlsausgabe
- Befehle können sich überholen; Ausführungseinheit wird dadurch früher frei
- Write-Back-Phase bringt Befehle in richtige Reihenfolge



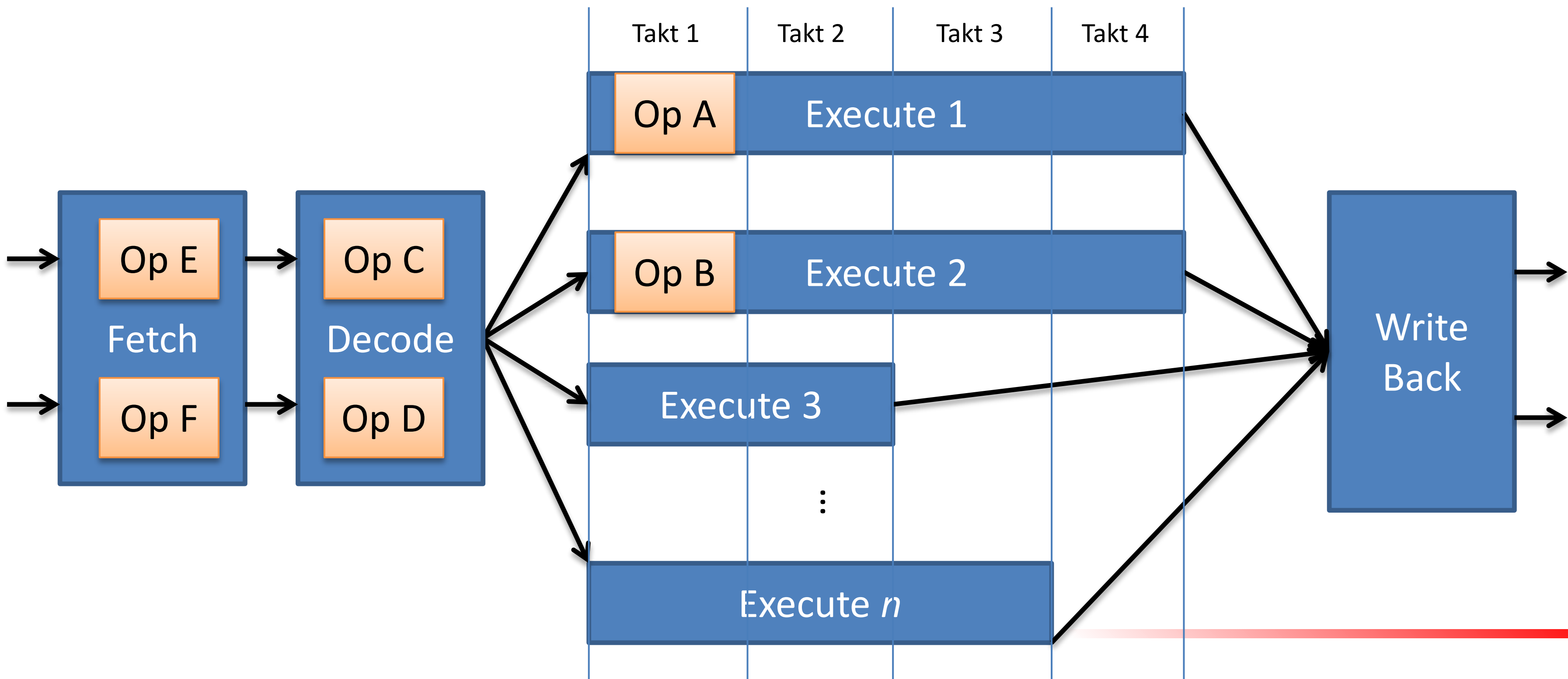
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung und/oder out-of-order Befehlsausgabe
- Befehle können sich überholen; Ausführungseinheit wird dadurch früher frei
- Write-Back-Phase bringt Befehle in richtige Reihenfolge (in-order Abschluss)



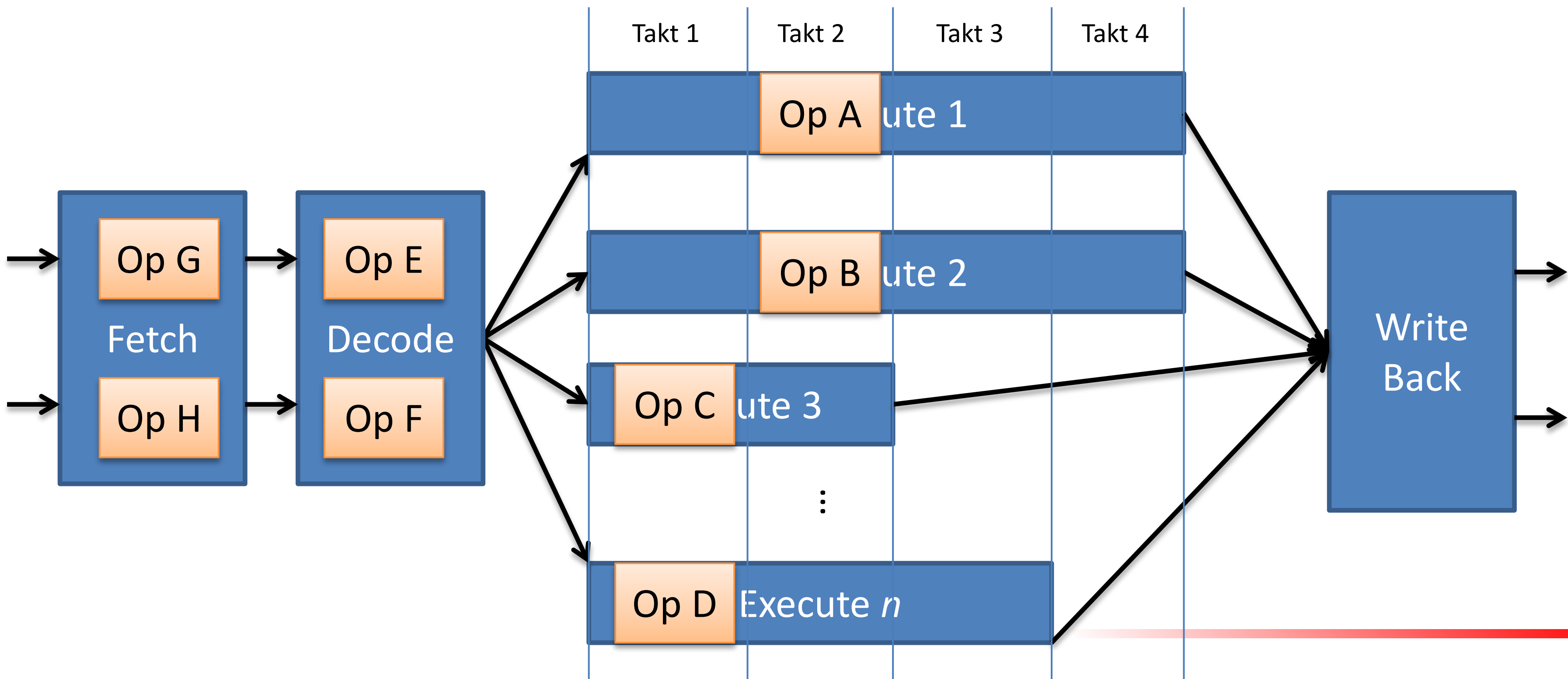
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung und/oder out-of-order Befehlsausgabe
- Befehle können sich überholen; Ausführungseinheit wird dadurch früher frei
- Write-Back-Phase bringt Befehle in richtige Reihenfolge (in-order Abschluss)



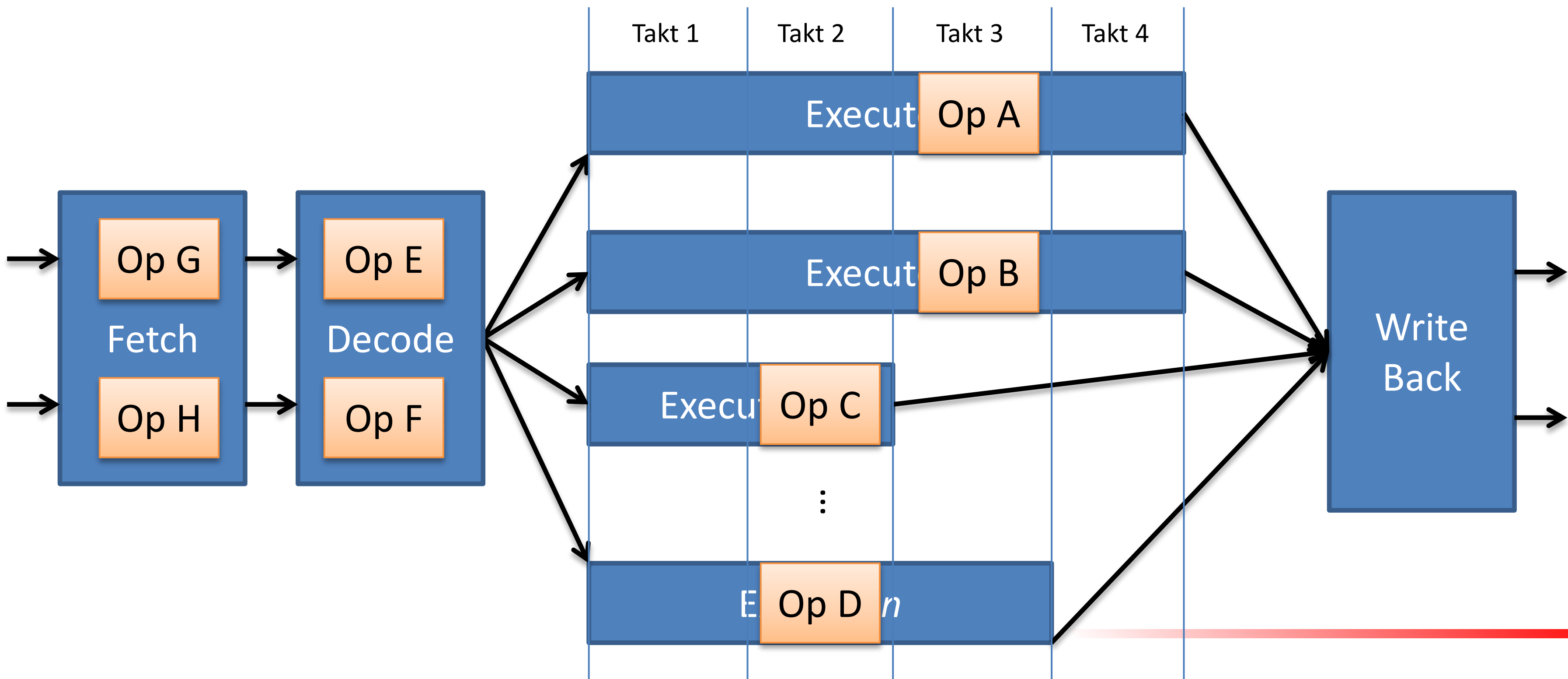
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung und/oder out-of-order Befehlsausgabe
- Befehle können sich überholen; Ausführungseinheit wird dadurch früher frei
- Write-Back-Phase bringt Befehle in richtige Reihenfolge (in-order Abschluss)



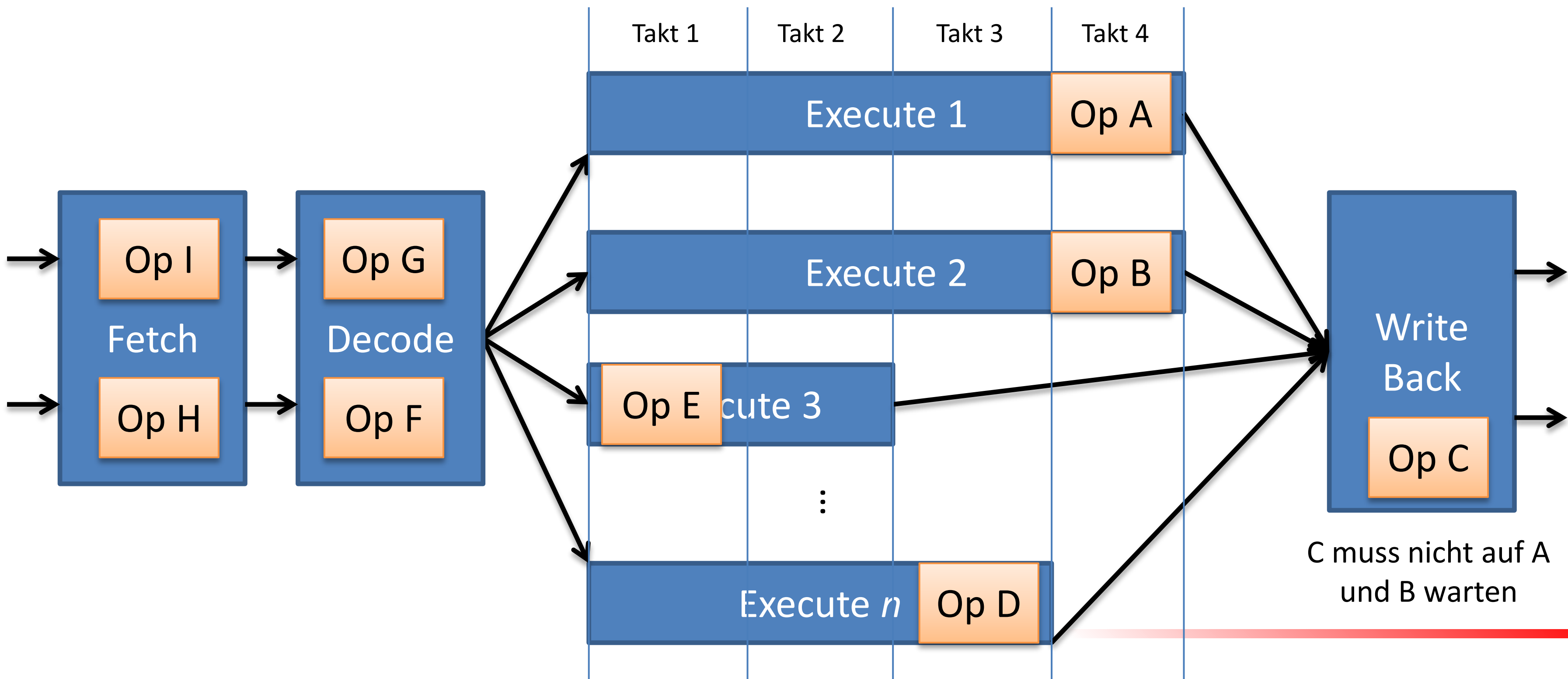
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung und/oder out-of-order Befehlsausgabe
- Befehle können sich überholen; Ausführungseinheit wird dadurch früher frei
- Write-Back-Phase bringt Befehle in richtige Reihenfolge (in-order Abschluss)



Superskalar Prozessor

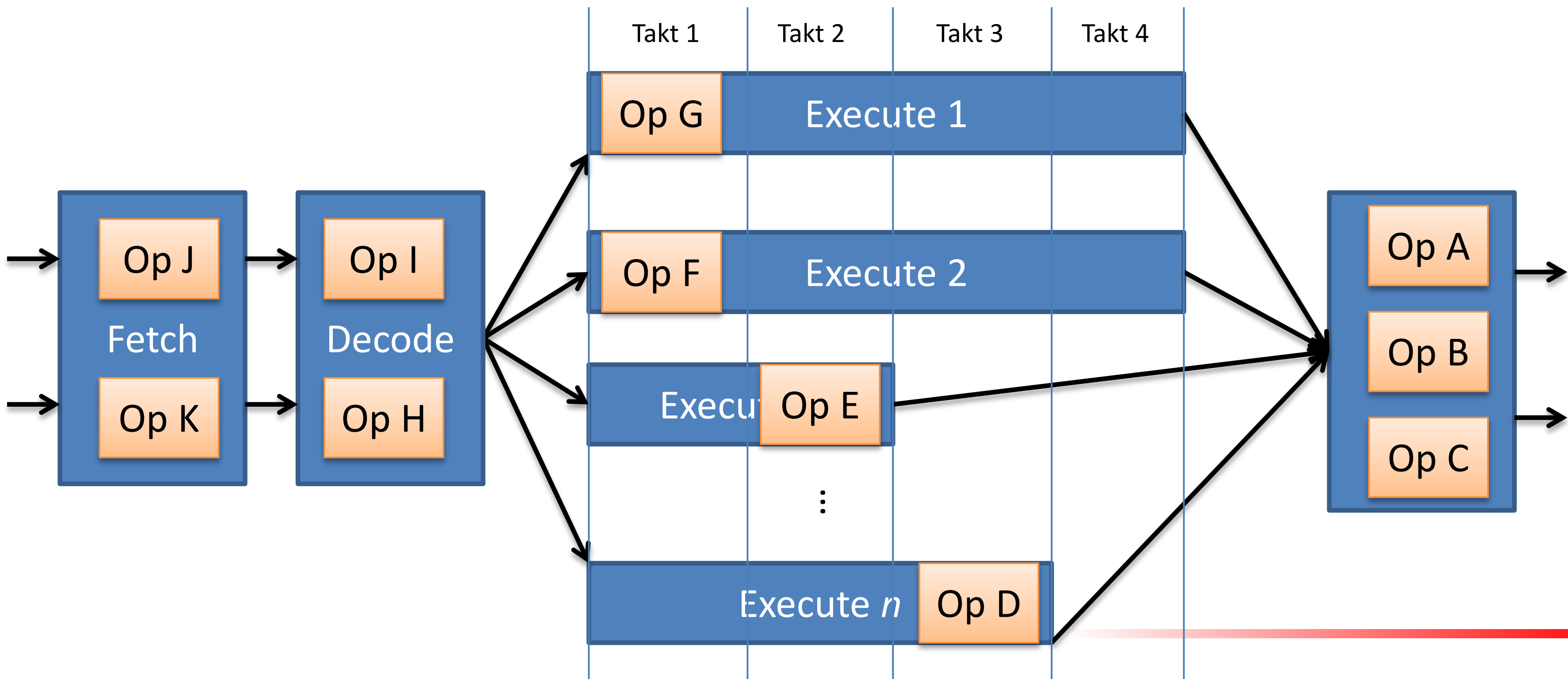
- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung und/oder out-of-order Befehlsausgabe
- Befehle können sich überholen; Ausführungseinheit wird dadurch früher frei
- Write-Back-Phase bringt Befehle in richtige Reihenfolge (in-order Abschluss)



C muss nicht auf A und B warten

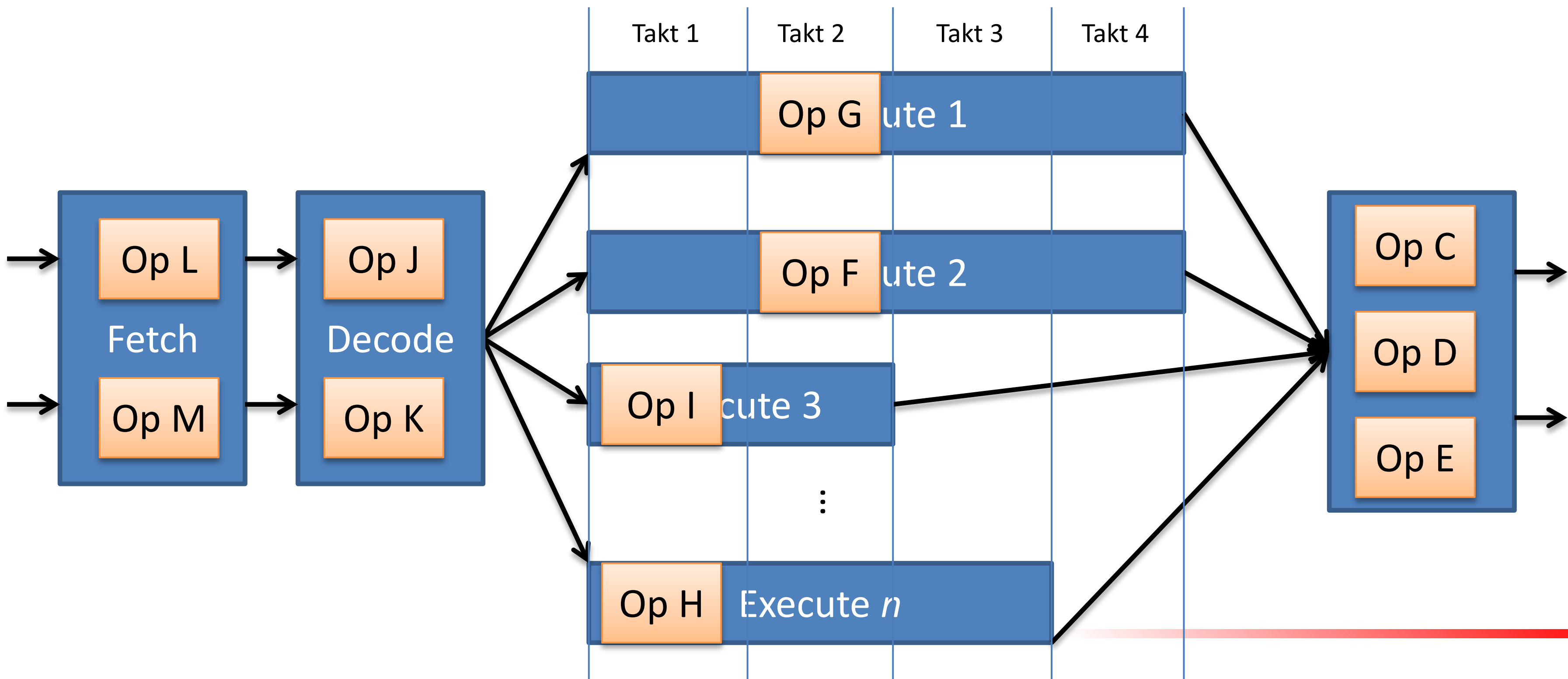
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung und/oder out-of-order Befehlsausgabe
- Befehle können sich überholen; Ausführungseinheit wird dadurch früher frei
- Write-Back-Phase bringt Befehle in richtige Reihenfolge (in-order Abschluss)

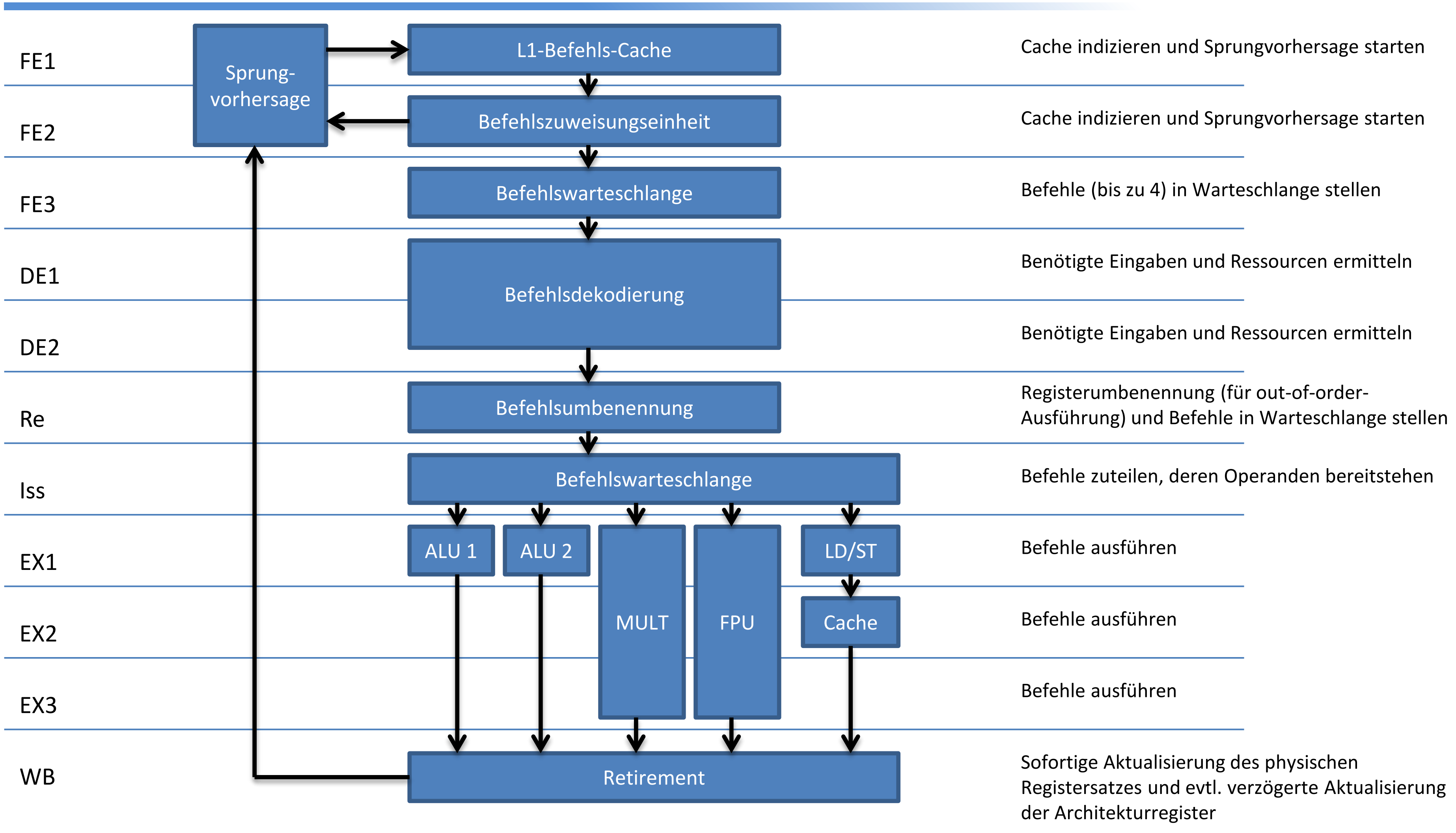


Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung und/oder out-of-order Befehlsausgabe
- Befehle können sich überholen; Ausführungseinheit wird dadurch früher frei
- Write-Back-Phase bringt Befehle in richtige Reihenfolge (in-order Abschluss)

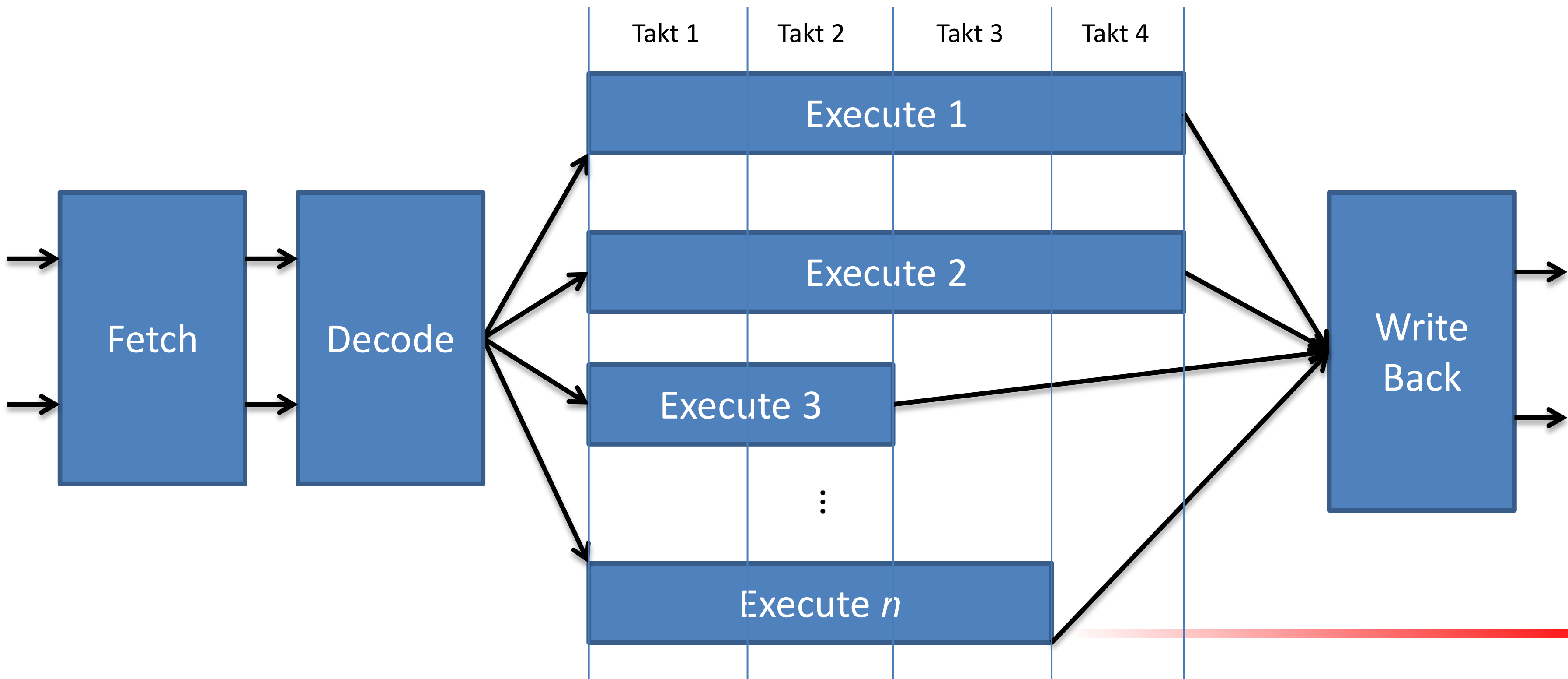


Beispiel Cortex-A9-Pipeline



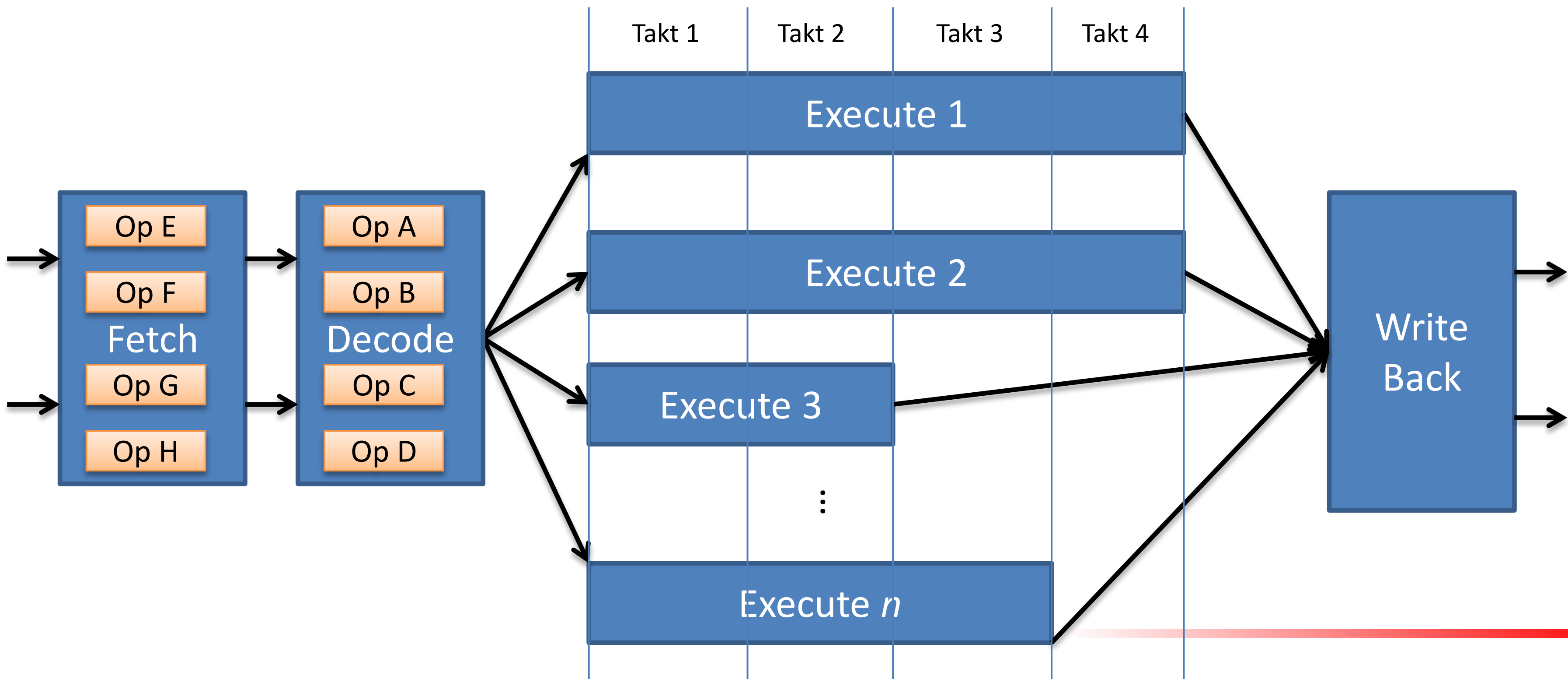
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung: Befehle können sich überholen
- Write-Back-Phase bringt Befehle in richtige Reihenfolge



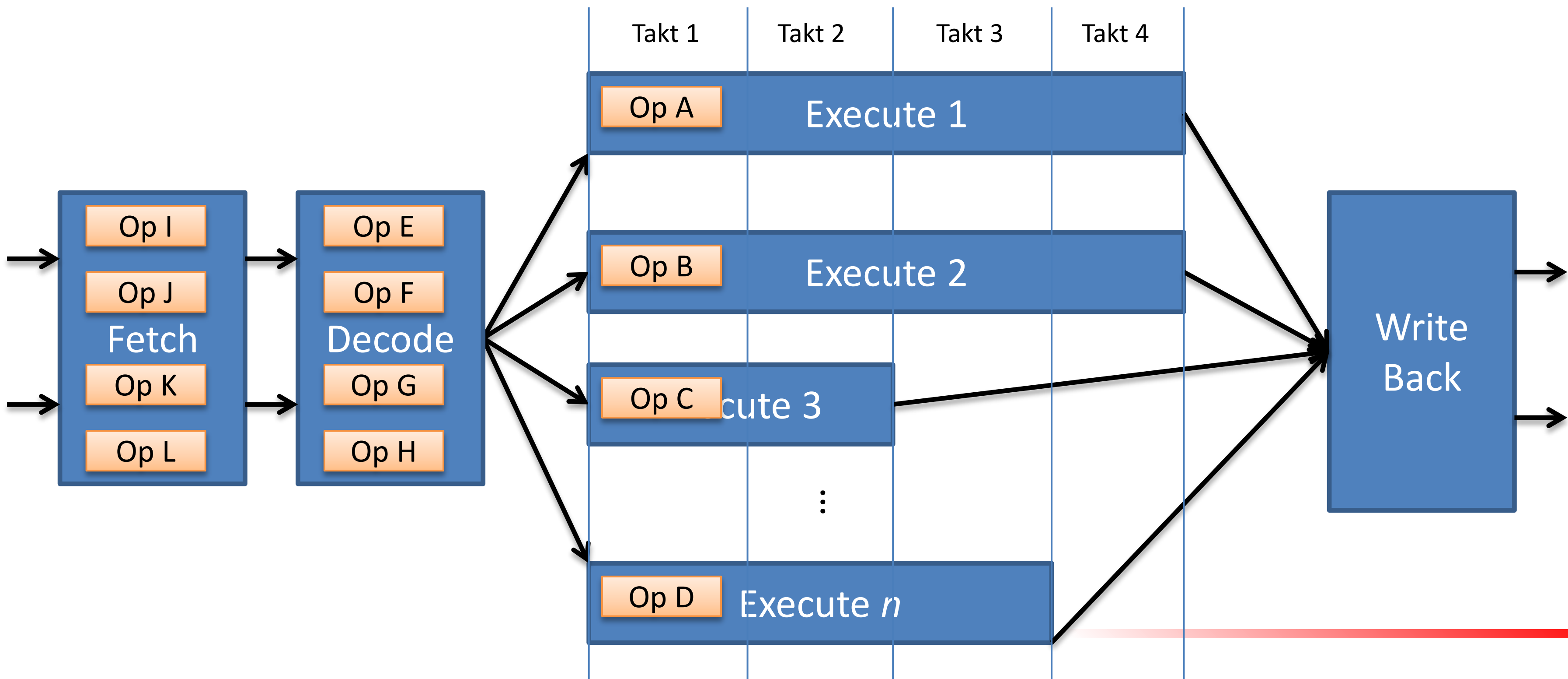
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung: Befehle können sich überholen
- Write-Back-Phase bringt Befehle in richtige Reihenfolge



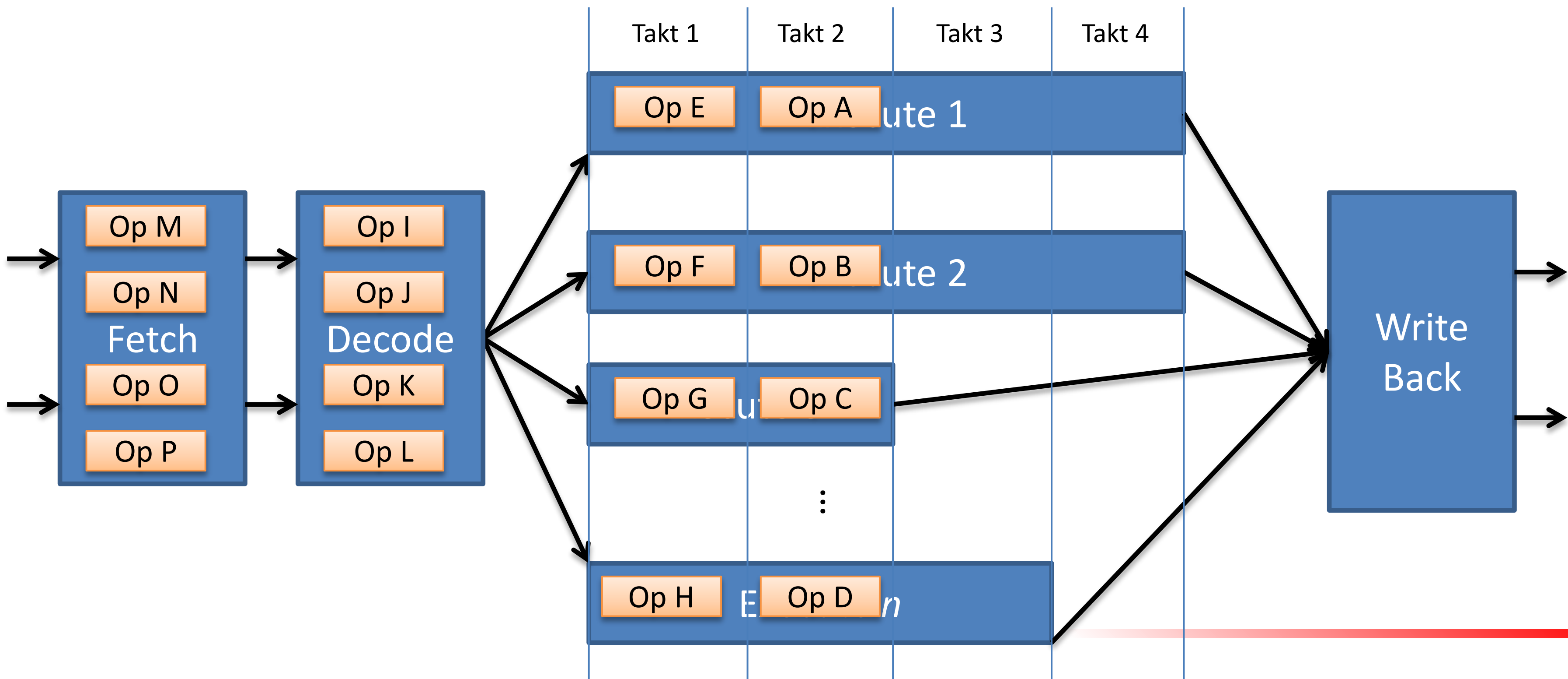
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung: Befehle können sich überholen
- Write-Back-Phase bringt Befehle in richtige Reihenfolge



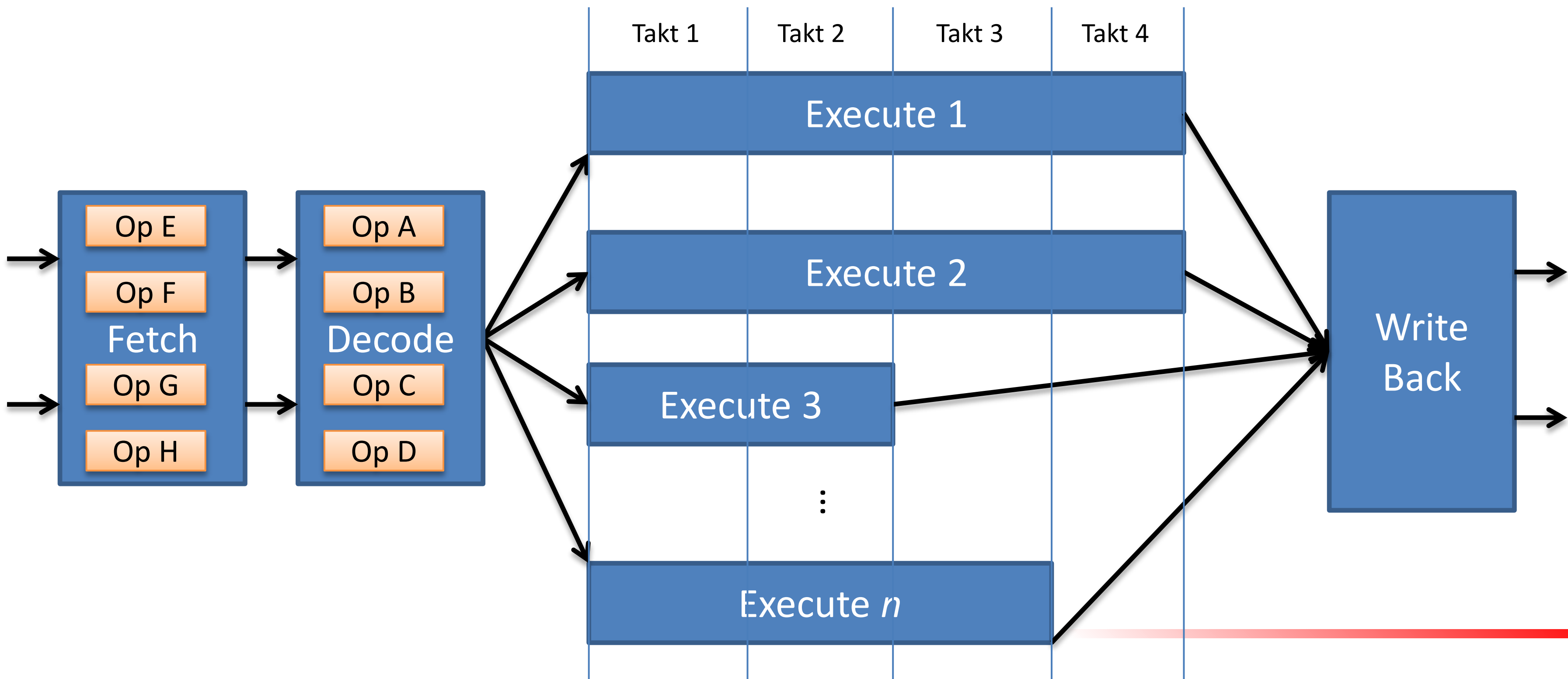
Superskalar Prozessor

- Ausführungseinheiten im Datenpfad
  - ohne Pipeline: Zuteilung eines Befehls erst, wenn voriger Befehl verarbeitet
  - mit Pipeline: Zuteilung eines Befehls in jedem Takt möglich
- In der Regel unterschiedliche Ausführungszeiten im Datenpfad möglich
  - in-order Befehlsausführung
  - out-of-order Befehlsausführung: Befehle können sich überholen
- Write-Back-Phase bringt Befehle in richtige Reihenfolge



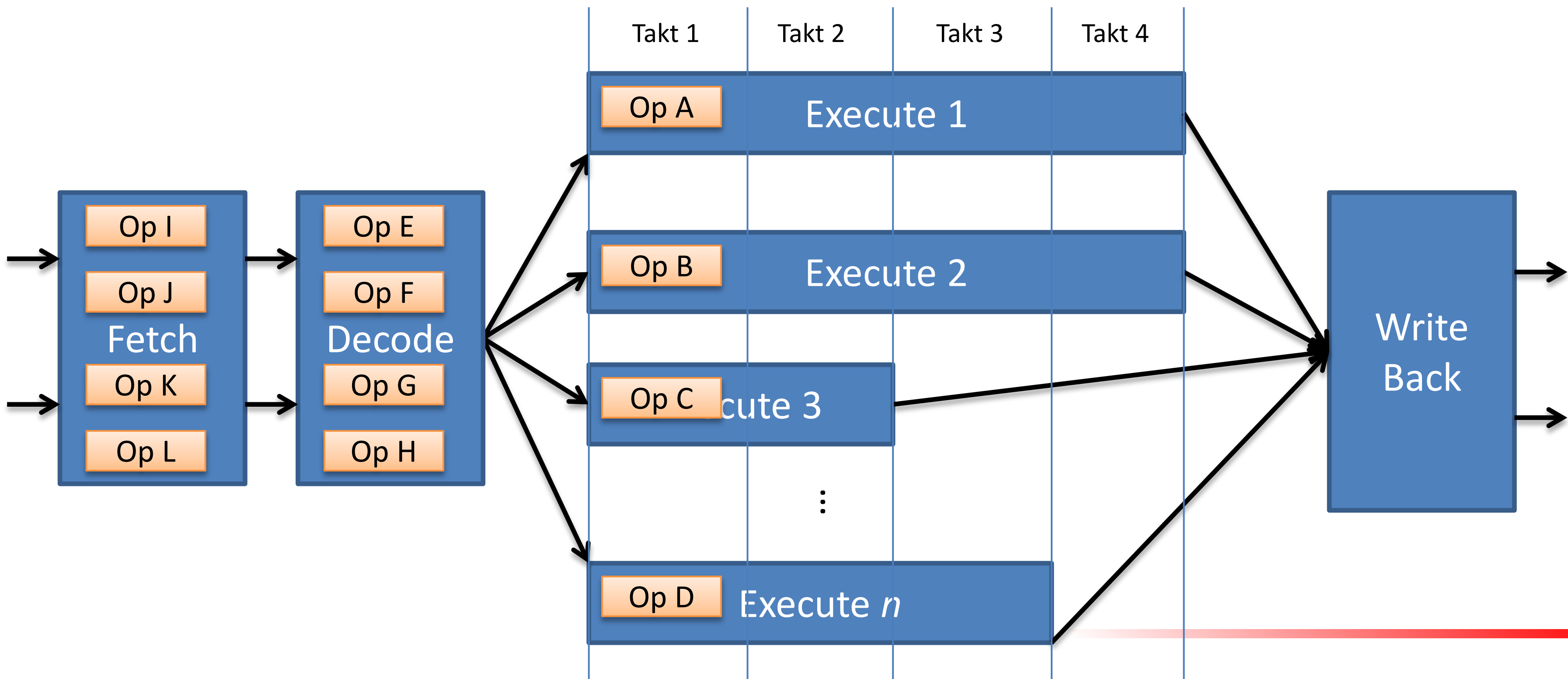
Effekt von Datenabhängigkeiten

- Datenabhängigkeiten müssen beachtet werden
- Verarbeitung einer Operation beginnt erst, wenn Daten vorhanden sind
- Out-of-order Ausgabe verhindert Pipeline-Stalls
- Annahme im Beispiel:
  - E benötigt Ergebnis von C



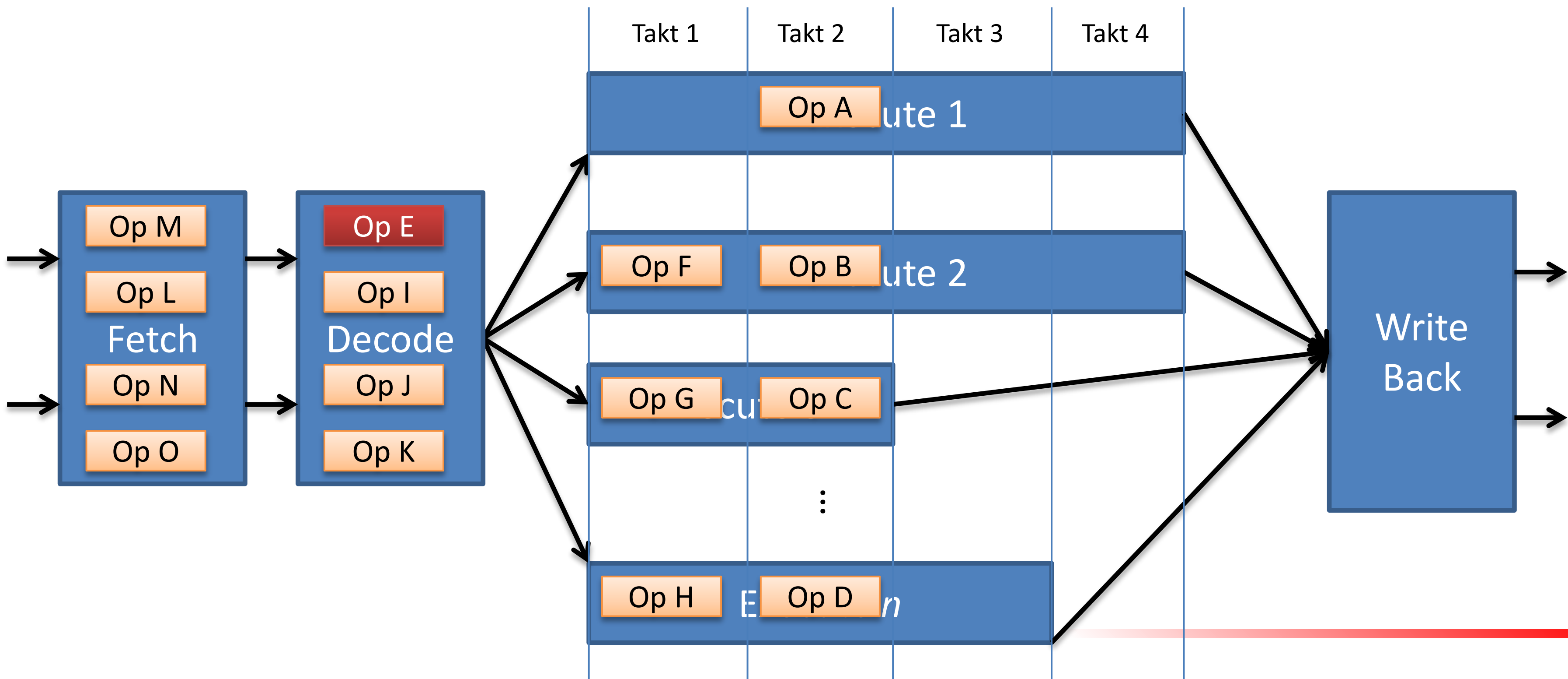
Effekt von Datenabhängigkeiten

- Datenabhängigkeiten müssen beachtet werden
- Verarbeitung einer Operation beginnt erst, wenn Daten vorhanden sind
- Out-of-order Ausgabe verhindert Pipeline-Stalls
- Annahme im Beispiel:
  - E benötigt Ergebnis von C



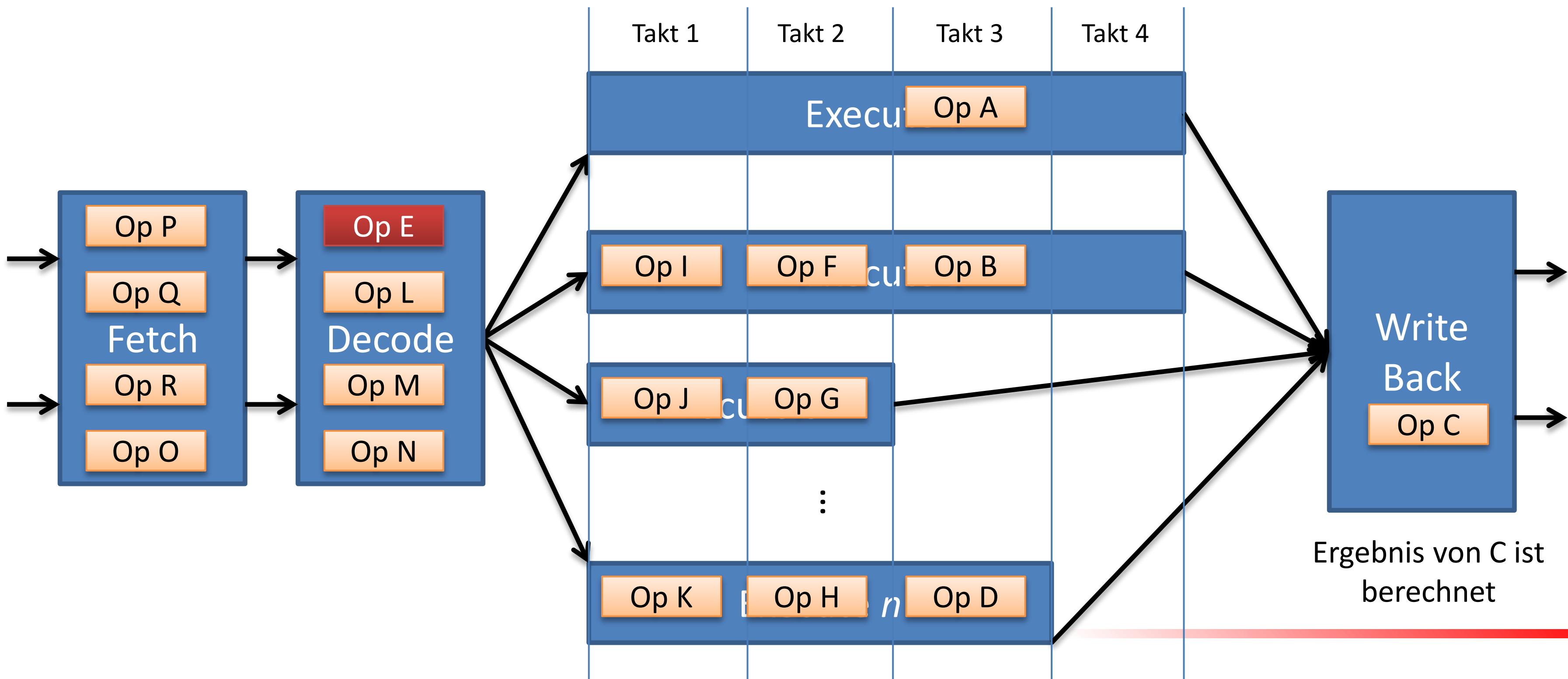
Effekt von Datenabhängigkeiten

- Datenabhängigkeiten müssen beachtet werden
- Verarbeitung einer Operation beginnt erst, wenn Daten vorhanden sind
- Out-of-order Ausgabe verhindert Pipeline-Stalls
- Annahme im Beispiel:
  - E benötigt Ergebnis von C



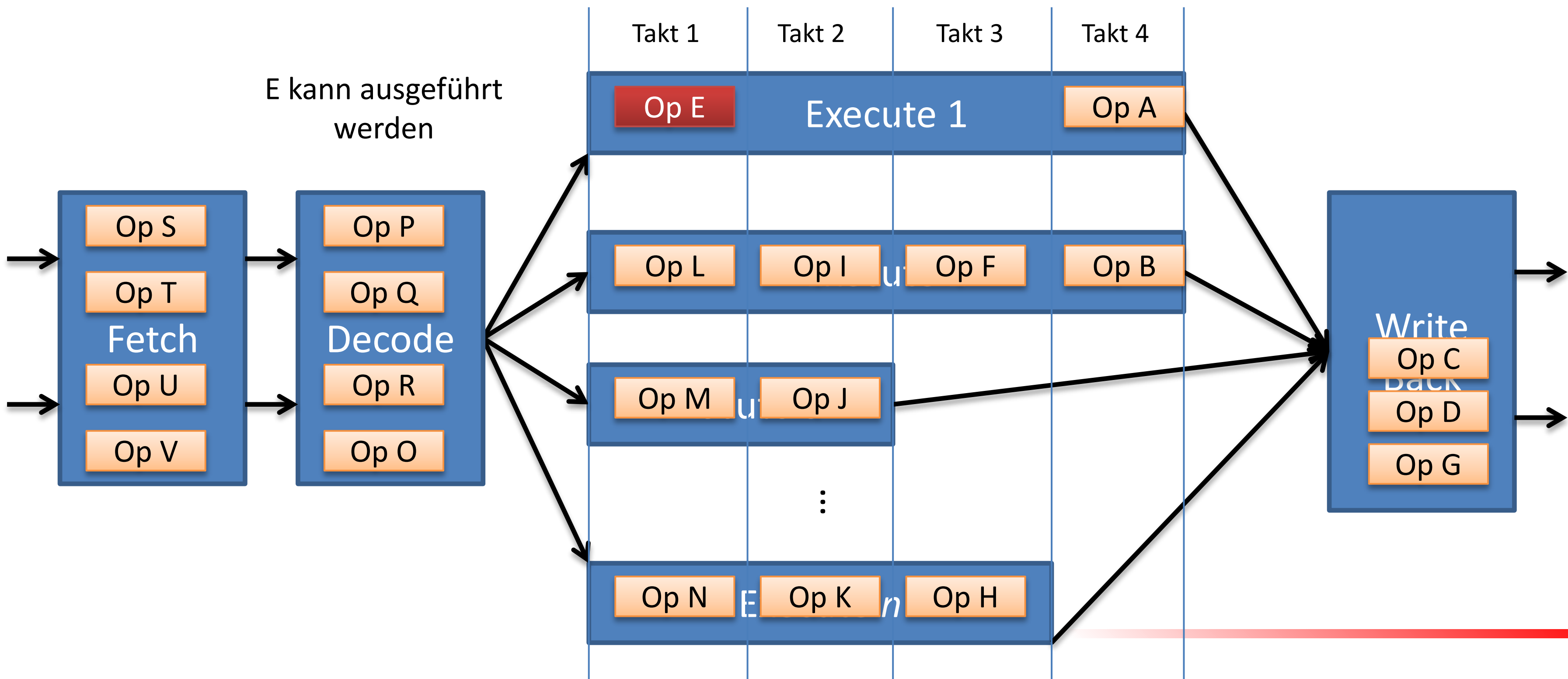
Effekt von Datenabhängigkeiten

- Datenabhängigkeiten müssen beachtet werden
- Verarbeitung einer Operation beginnt erst, wenn Daten vorhanden sind
- Out-of-order Ausgabe verhindert Pipeline-Stalls
- Annahme im Beispiel:
  - E benötigt Ergebnis von C



Effekt von Datenabhängigkeiten

- Datenabhängigkeiten müssen beachtet werden
- Verarbeitung einer Operation beginnt erst, wenn Daten vorhanden sind
- Out-of-order Ausgabe verhindert Pipeline-Stalls
- Annahme im Beispiel:
  - E benötigt Ergebnis von C



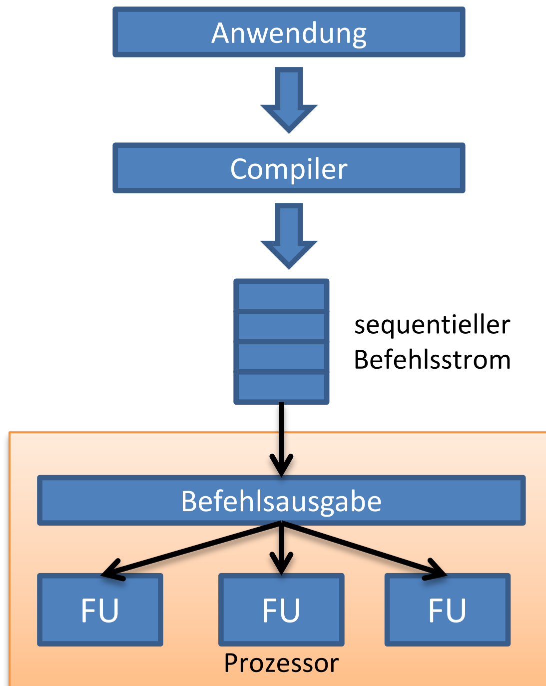
Zusammenfassung Pipelinearchitekturen

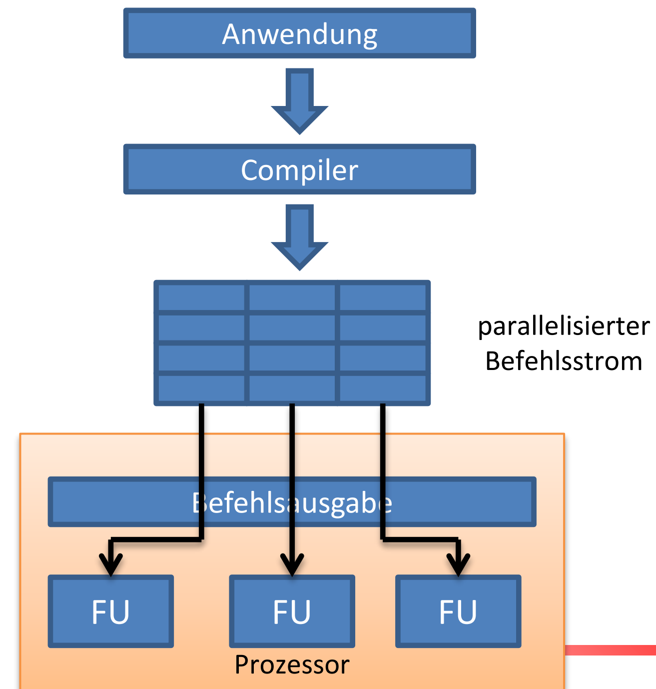
- Architekturen mit CPI < 1 nicht möglich:
  - skalarer Datenpfad mit Befehlspipeline
  - skalarer Datenpfad mit Befehls- und Datenpfadpipeline
    - In-order-Ausgabe
    - Out-of-order Ausgabe
  - Superskalarer Datenpfad, Ausführungseinheiten ohne Pipeline
    - In-order-Ausgabe
    - Out-of-order-Ausgabe
  - Superskalarer Datenpfad, Ausführungseinheiten mit Pipeline sinnvoll?

- Architekturen mit CPI < 1 möglich (superskalarer Prozessor):
  - Ausführungseinheiten ohne Pipeline
    - in-order Verarbeitung
    - out-of-order Verarbeitung
  - Ausführungseinheiten mit Pipeline
    - out-of-order Ausführung

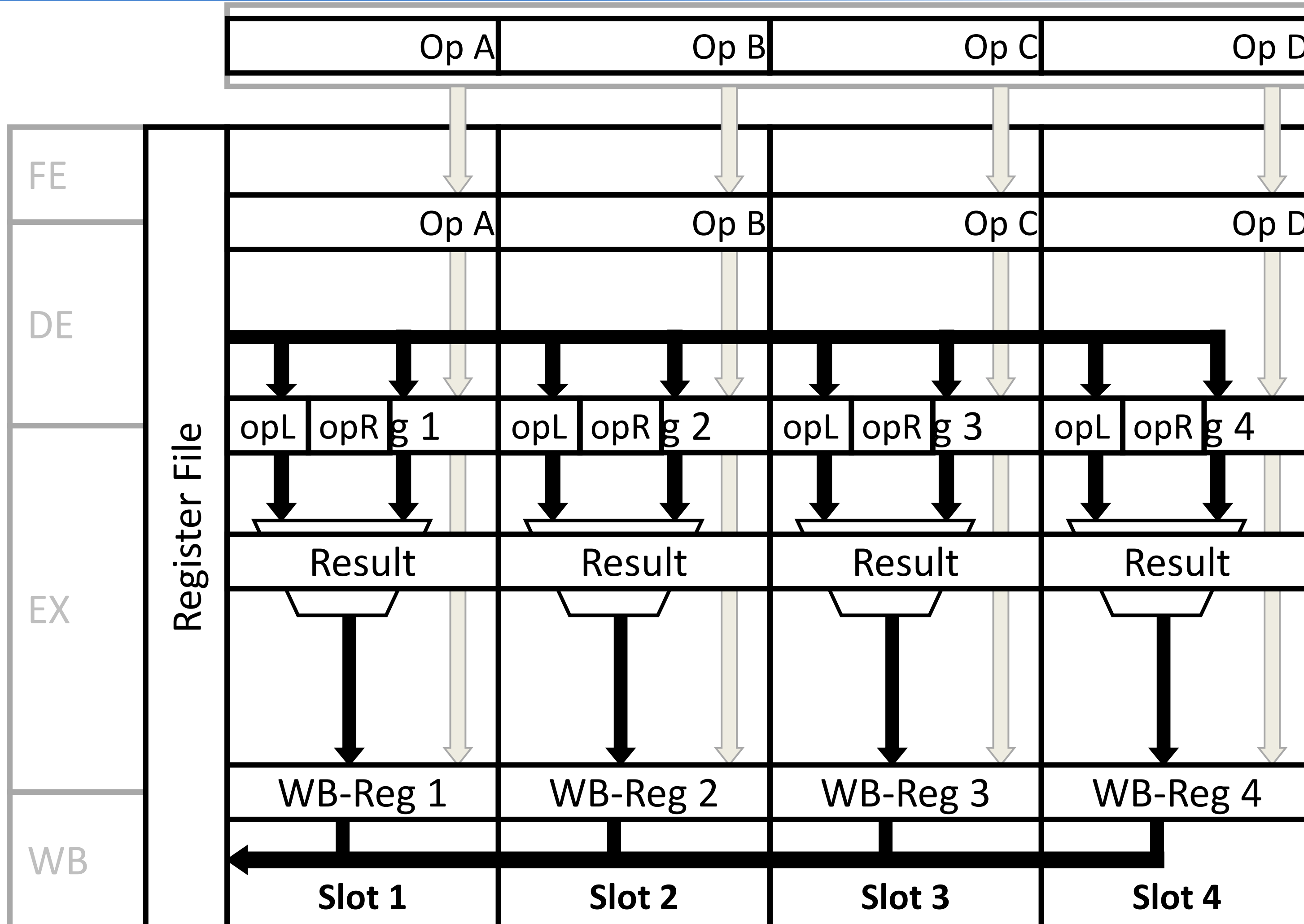
Zuordnung von Operationen zu Datenpfad-Pipelines

- Bisher nicht betrachtet bei Befehlsausgabe in Prozessoren mit superskalarem Datenpfad:
  - Zuordnung der Befehle zur Datenpfadpipeline
- Zuordnung der Befehle zu Pipelines erfolgt:
  - Dynamisch: Prozessor entscheidet zur Laufzeit in welcher Pipeline eine Befehl verarbeitet wird
    - Anzahl der Pipelines wird berücksichtigt
    - Aktuelle Belegung der Pipeline
  - Statisch: Compiler/Programmierer legt statisch im Programm die Zuordnung zu einer Pipeline fest
    - Programm ist an eine festgelegte Mikroarchitektur gebunden





Beispiel für statische Befehlsplanung (VLIW Prozessor)



Klassifizierung

Typ	Zuordnung	Scheduling	Charakteristikum	Beispiel
Superskalar	dynamisch	statisch	Befehle werden in ursprünglicher Reihenfolge abgearbeitet	Intel Pentium
Superskalar	dynamisch	dynamisch	Out-of-order (evtl. mit spekulativer Ausführung)	Core i7, Cortex-A9
VLIW (Very Long Instruction Word)	statisch	statisch	Parallelität wird im Code explizit gekennzeichnet; Zuordnung zu Datenpfadpipelines auch	TI C6x
EPIC (Explicit Parallel Instruction Computing)	dynamisch	statisch	Parallelität wird im Code explizit gekennzeichnet; Zuordnung zu Datenpfadpipelines erfolgt dynamisch	Itanium

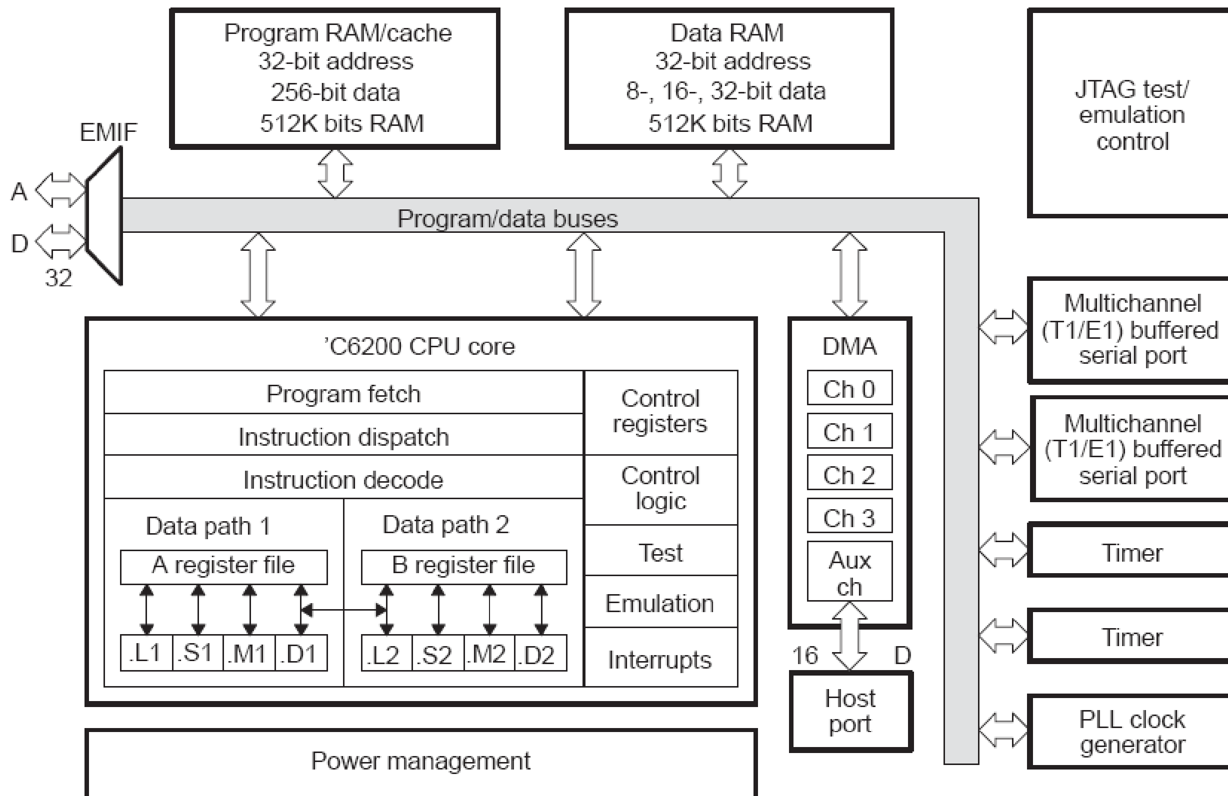
Geclusterter VLIW-Prozessor

- Registerbänke mit zu vielen Lese-/Schreibports zu teuer und langsam

- Aufteilung des Datenpfads in Cluster:
  - Jeder Cluster besitzt lokale Registerbank
  - FUs eines Clusters haben nur Zugriff auf lokale Registerbank

- Konsequenz:
  - Verbindungsnetzwerk zwischen Clustern erforderlich
  - Variante 1: Datenaustausch durch explizite Kopieroperationen
  - Variante 2: Überlappende Registerbänke

Beispiel: TMS320C6201 CPU



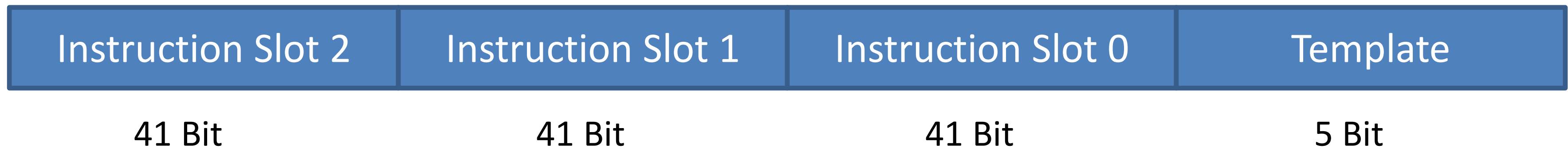
EPIC

- EPIC = Explicit Parallel Instruction Computing
  - VLIW: statisches Scheduling + statische Zuordnung
  - EPIC: statisches Scheduling + dynamische Zuordnung

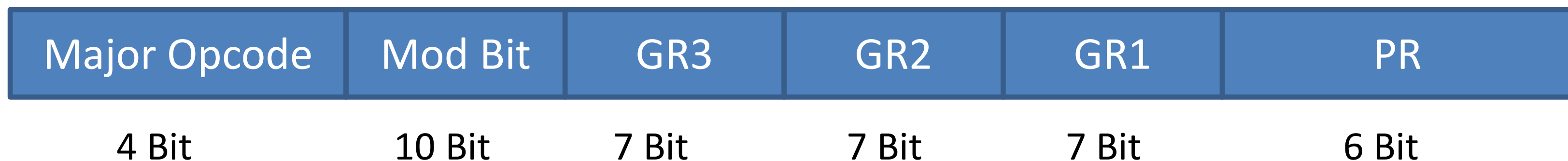
- Beispiel: Intel Itanium-Prozessor:
  - 128 64-Bit Integerregister + 128 82-Bit Gleitkommaregister
  - Typen von Ausführungseinheiten:
    - I-Unit (Integer)
    - M-Unit (Memory)
    - B-Unit (Branch)
    - F-Unit (Gleitkomma)
  - Compiler gruppiert parallel ausführbare Operationen, die dann gleichzeitig an vorhandene Ausführungseinheiten zugeordnet werden.

Befehlsformat

128-Bit Bündel:



41-Bit Instruktionsformate:



Bedeutung Template-Feld

Template	Slot 0	Slot 1	Slot 2
00000	M-unit	I-unit	I-unit
00001	M-unit	I-unit	I-unit + stop
00010	M-unit	I-unit + stop	I-unit
00011	M-unit	I-unit + stop	I-unit + stop
…			
10011	M-unit	B-unit	B-unit + stop

- Parallel ausführbare Operationen können über mehrere Bundles gehen
- *Stop* signalisiert Ende einer Folge von parallel ausführbaren Operationen
- Es können mehrere Bundles gleichzeitig geholt werden
- Anzahl Ausführungseinheiten skalierbar; trotzdem Binärkompatibilität

Zusammenfassung

- Superskalar: Mehre (Datenpfad-)Pipelines vorhanden
- Issue = Zuordnung der Operationen zu einer Datenpfad-Pipeline in einem superskalaren Prozessor
  - Statisch = wurde in Software fest codiert
  - Dynamisch = wird durch die Hardware zur Laufzeit festgelegt
- Scheduling = Festlegung der Ausführungsreihenfolge von Operationen
  - Statisch (in-order): Im Programm festgelegte Reihenfolge wird beim Starten der Operationen beibehalten; Operationen müssen wegen Abhängigkeiten evtl. warten
  - Dynamisch (out-of-order): Warten kann verhindert werden, weil datenunabhängige Operationen vorgezogen werden können