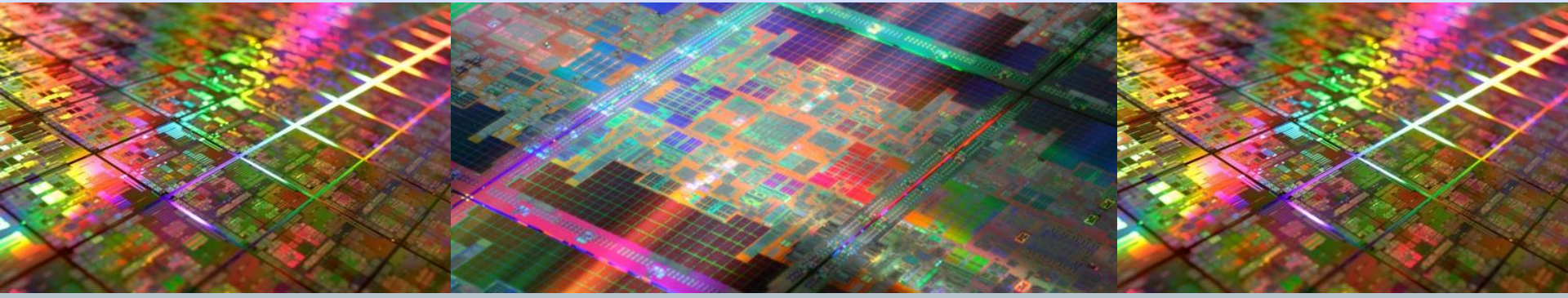
Prozessor
Datenpfad und Steuerwerk

Technische Grundlagen der Informatik für Wirtschaftsinformatik

Stefan Podlipnig

TU Wien



Lernziele

- Verstehen der Arbeitsweise eines CPU-Steuerwerks
- Detailliertes Verständnis der Abarbeitung eines Befehls in einer CPU am Beispiel MIPS

MIPS DATENPFAD

Einleitung

- Zwei MIPS-Implementierungen
  - Eine einfache Version
  - Eine verbesserte Version mit Pipelining
- Beschränkung auf folgende kleine Menge an Befehlen
  - Speicherzugriffe: `lw`, `sw`
  - Arithmetisch/logisch: `add`, `sub`, `and`, `or`, `slt`
  - Ablaufsteuerung: `beq`, `j`

Befehlsausführung

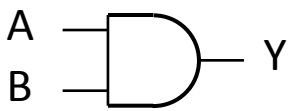
- Befehlszähler an den Speicher schicken, Befehl aus dem Speicher holen
- Lesen der Register (Dekodieren des Befehls)
  - Auswahl der Register erfolgt mithilfe der Registerfelder im Befehl
- Abhängig von der Befehlsklasse
  - Benutzung der ALU für Berechnungen
    - Arithmetisches Ergebnis
    - Speicheradresse für Laden/Speichern
    - Sprungadresse für Verzweigung
  - Zugriff auf den Speicher bei Lade-/Speicherbefehlen
  - Befehlszähler aktualisieren
    - Alter Befehlszähler + 4
    - Sprungadresse

Konventionen beim Entwurf

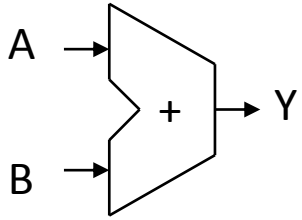
- Positive Logik
  - Hohe Spannung (High-Pegel) entspricht dem Wahrheitswert 1
  - Niedrige Spannung (Low-Pegel) entspricht dem Wahrheitswert 0
- Kombinatorische Logik
  - Operiert auf Daten
  - Ausgabe ist eine Funktion der Eingabe
- Sequentielle Logik
  - Speichert Information

Kombinatorische Logik (Beispiele)

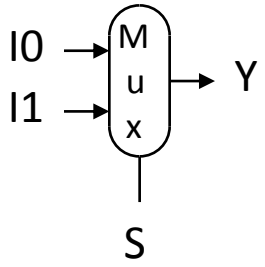
- UND-Gatter
  - Y = A & B



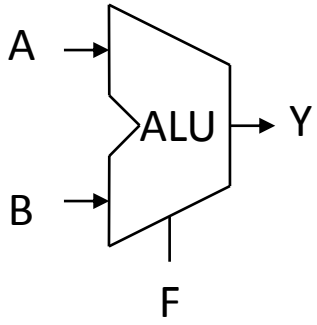
- Addierer
  - Y = A + B



- Multiplexer
  - Y = S ? I1 : I0



- Arithmetisch/Logische Einheit
  - Y = F(A, B)

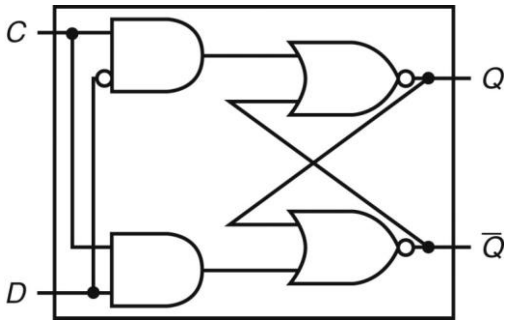


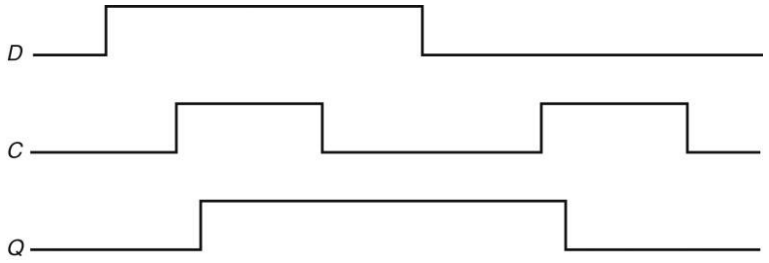
Sequentielle Logik (1)

- Register
  - Schnelle Speicher für Werte
  - Es wird ein Taktsignal (*clock*) verwendet
    - Bestimmt, wann ein Wert im Register verändert wird
  - Flankengesteuertes Taktverfahren
    - Update der Daten, wenn z.B. der Takt von 0 auf 1 wechselt (positive Taktflanke)
- MIPS-Register
  - 32 Bit
  - Jedes einzelne Bit wird zum Beispiel in einem D-Flip-Flop gespeichert

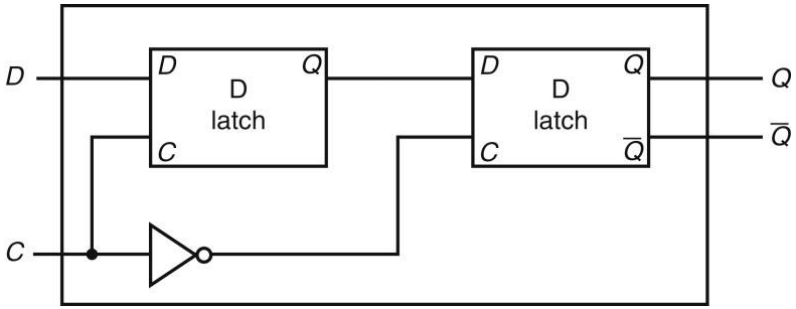
Sequentielle Logik (2)

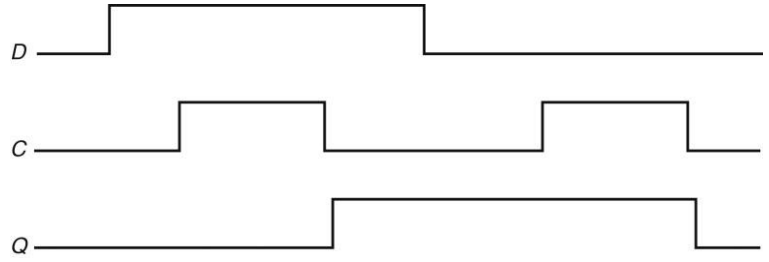
- Beispiel D-Latch





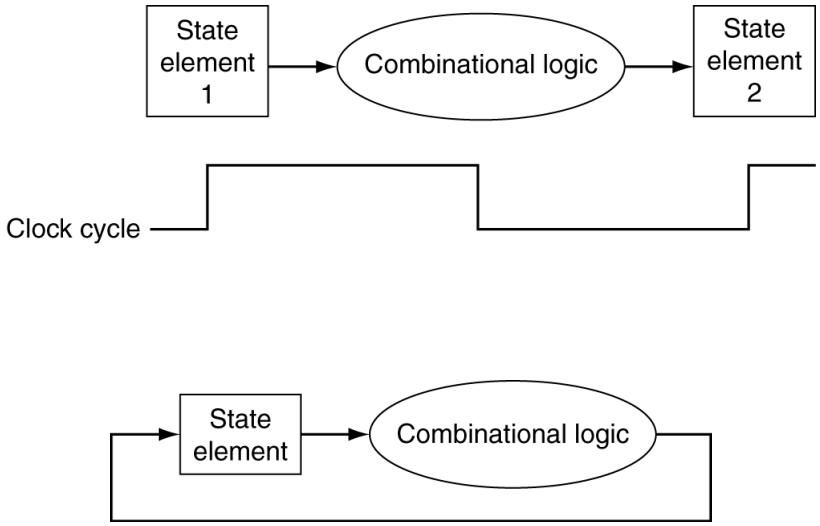
- D-Flip-Flop





Taktverfahren

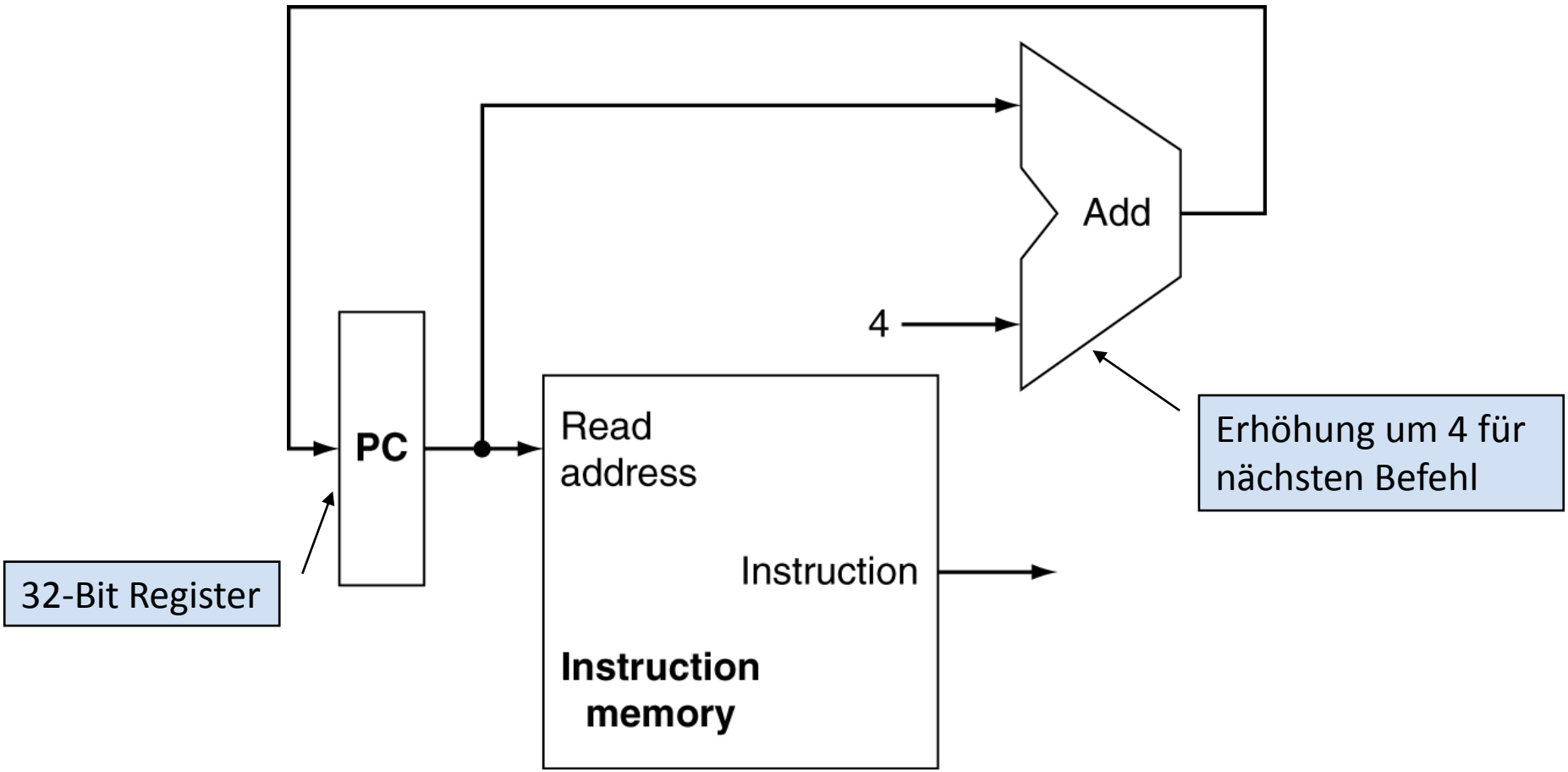
- Kombinatorische Logik transformiert Daten während eines Taktzyklus
  - Zwischen positiven Flanken
  - Eingabe kommt aus Speicherelementen
  - Ausgabe wird in Speicherelementen abgelegt
  - Die längste Verzögerung bestimmt die Taktperiode



Aufbau eines Datenpfades

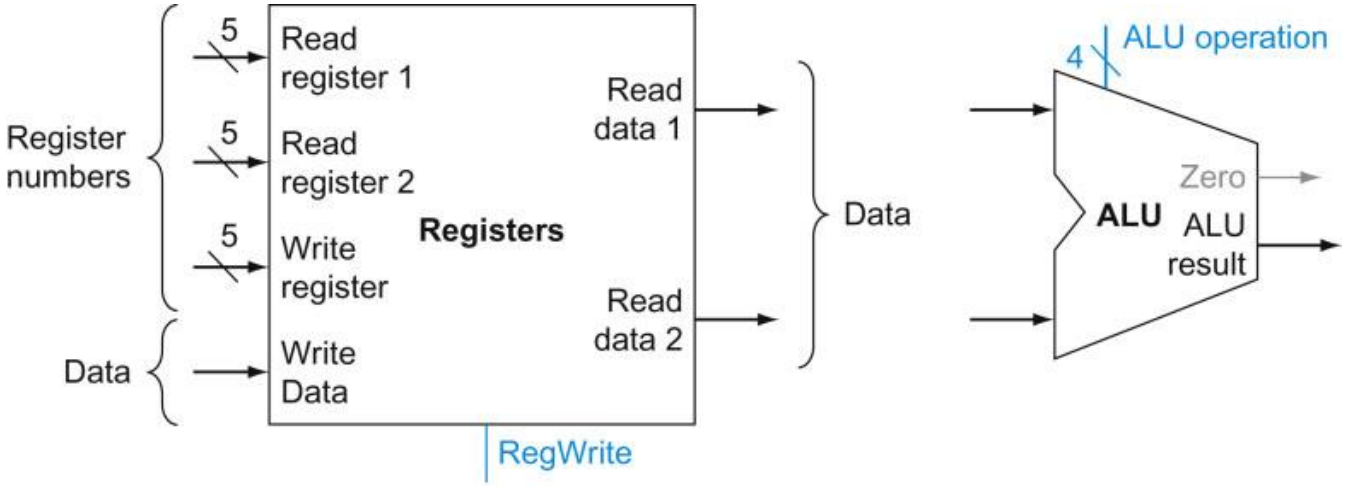
- Datenpfad beinhaltet Elemente, die Daten und Adressen in der CPU verarbeiten
  - Register
  - ALUs
  - Multiplexer
  - Speicher
- Nachfolgend wird der Aufbau des MIPS Datenpfades schrittweise erklärt
  - Zunächst Eintaktdatenpfad

Holen des Befehls



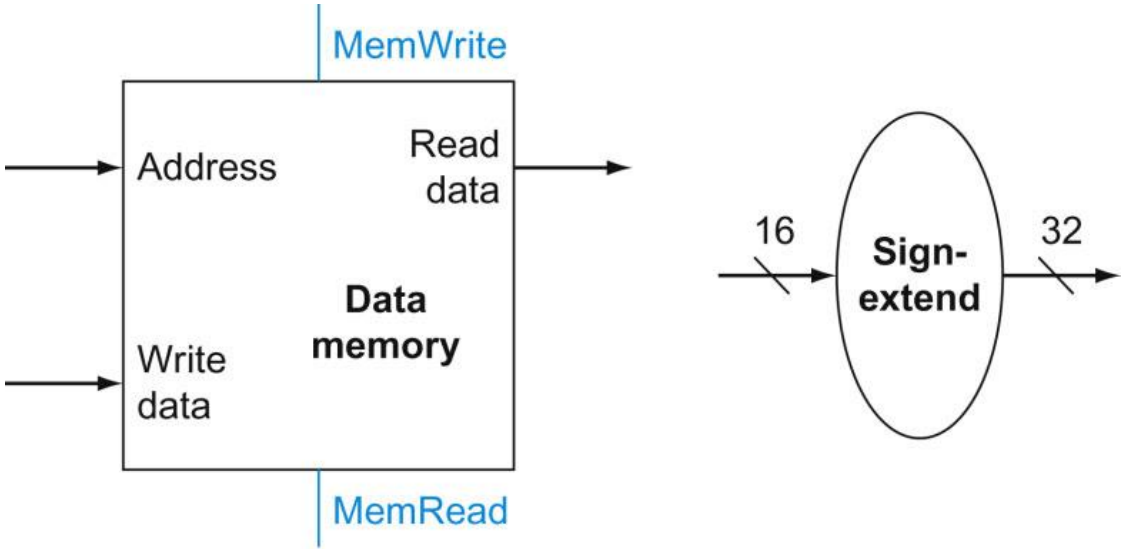
Befehle im R-Format

- Lesen zwei Register
- Führen mit dem Inhalt der Register eine ALU-Operation aus
- Schreiben das Ergebnis zurück
- Zwei Komponenten werden benötigt
  - Registersatz (alle 32 Register)
  - ALU



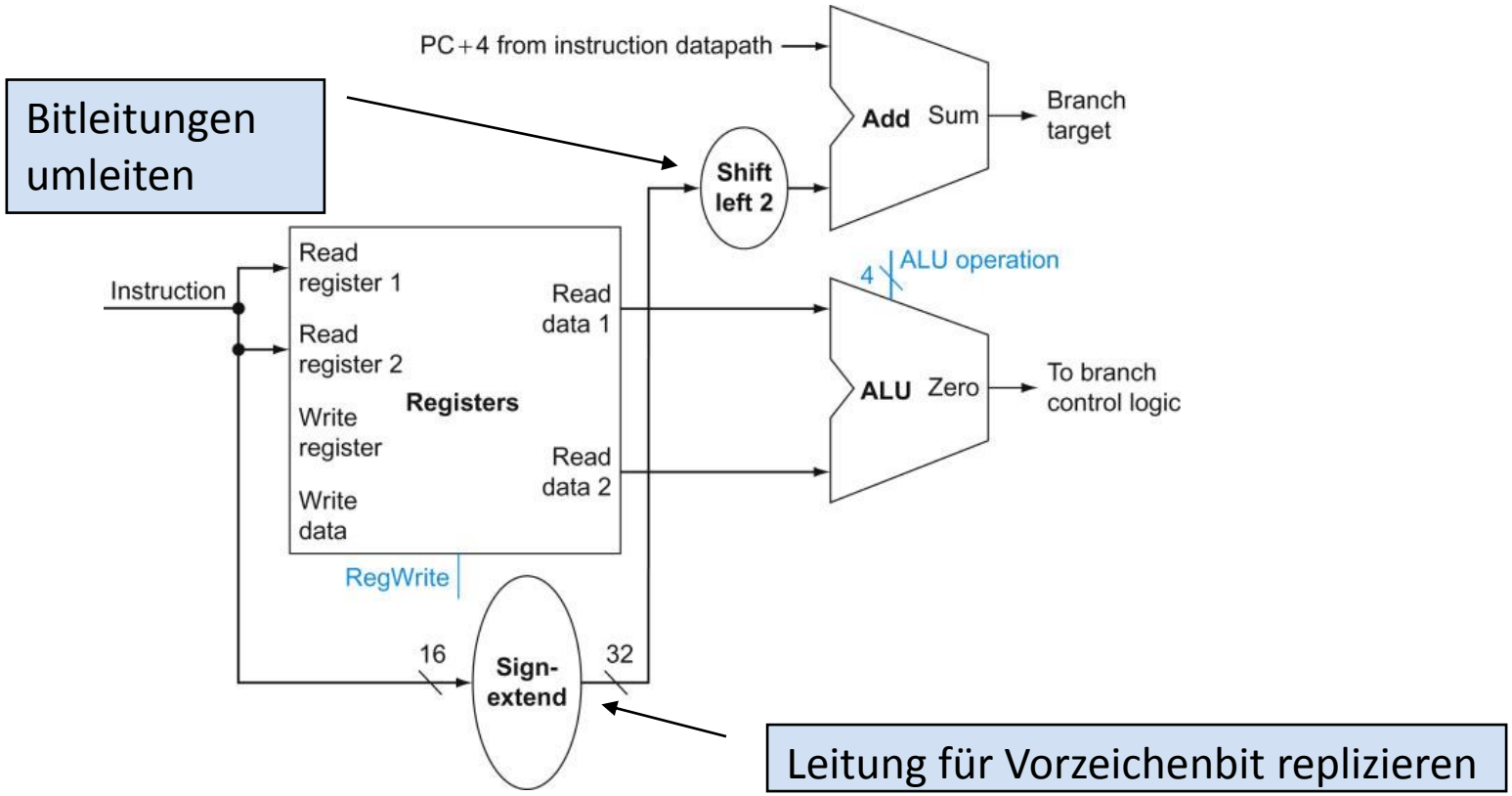
Befehle zum Laden und Speichern

- Lesen einen Operanden aus einem Register
- Berechnen Adresse mithilfe des 16-Bit Offsets
  - ALU wird benutzt
  - Ein Input ist der vorzeichenerweiterte (*sign extended*) Offset
- Laden: Aus dem Speicher lesen und in das Register speichern
- Speichern: Wert aus dem Register in den Speicher schreiben



Befehle für Verzweigungen

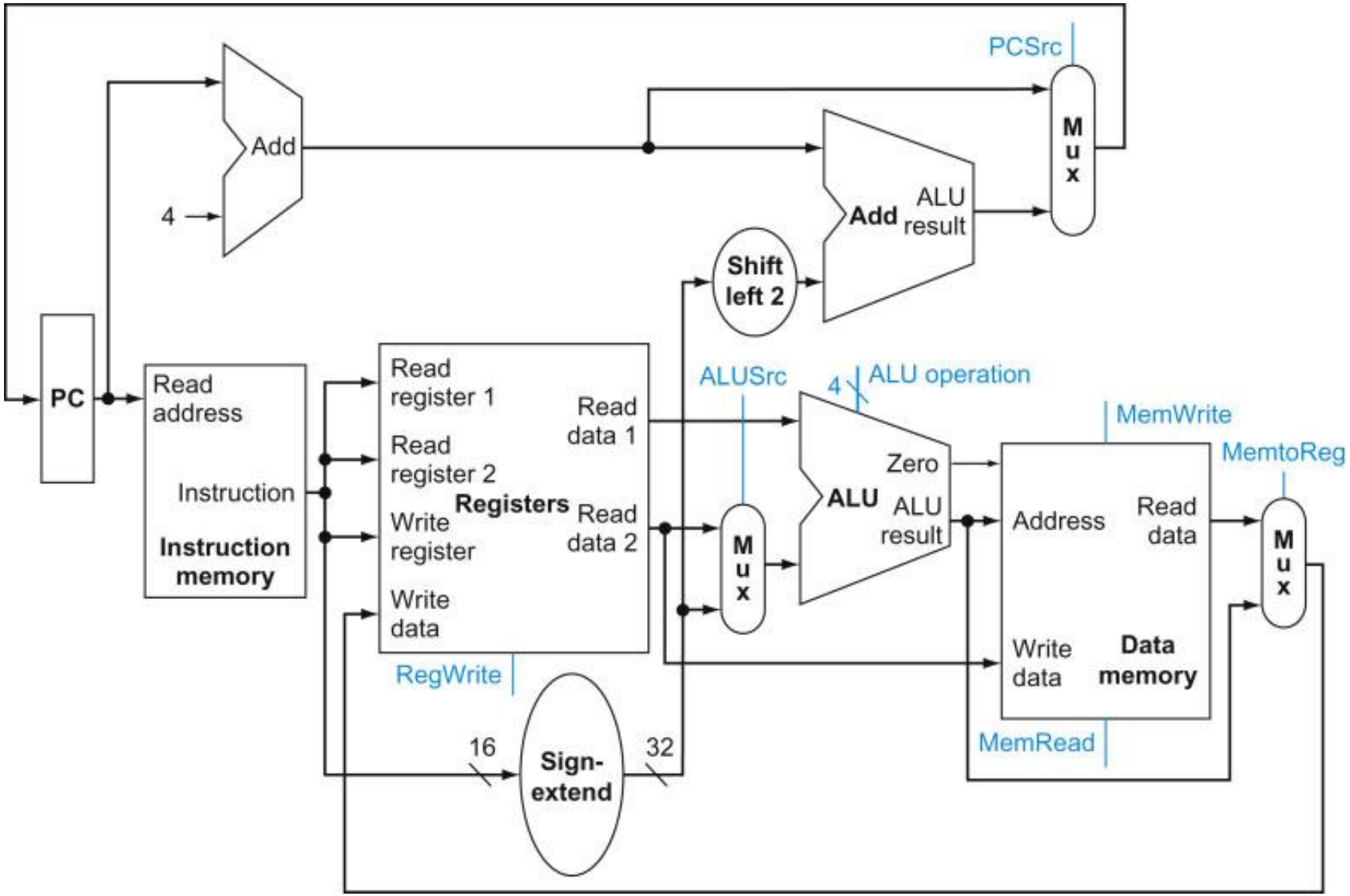
- Vergleiche Operanden aus den Registern (benutze ALU)
  - Zuerst Subtraktion, dann Überprüfung auf 0
- Berechne die Sprungzieladresse (*branch target address*)
  - Vorzeichenerweitertes Offset um 2 Bits nach links verschieben
  - Addiere 4 zum Befehlszähler



Aufbau eines einfachen Datenpfads

- Einfacher Datenpfad führt einen Befehl in einem Taktzyklus aus
  - Jedes Element im Datenpfad kann nur einmal pro Befehl verwendet werden
  - Mehrfach benötigtes Element muss mehrfach vorhanden sein
  - Befehlsspeicher (*instruction memory*) und Datenspeicher (*data memory*) müssen getrennt werden
- Multiplexer benutzen
  - Wenn ein Element von mehreren Befehlen benutzt werden soll
  - Steuersignale wählen zwischen den verschiedenen Eingängen aus

Datenpfad (R-Format, Laden/Speichern, Verzweigung)



ALU-Steuerung (1)

- ALU wird benutzt für
  - Laden/Speicher: Addieren
  - Verzweigung: Subtrahieren (Vergleich in `beq`)
  - Operationen im R-Format: Operation hängt von den Bits im Feld `funct` ab

ALU-Steuerleitungen	Operation
0000	and
0001	or
0010	add
0110	sub
0111	slt
1100	nor

ALU-Steuerung (2)

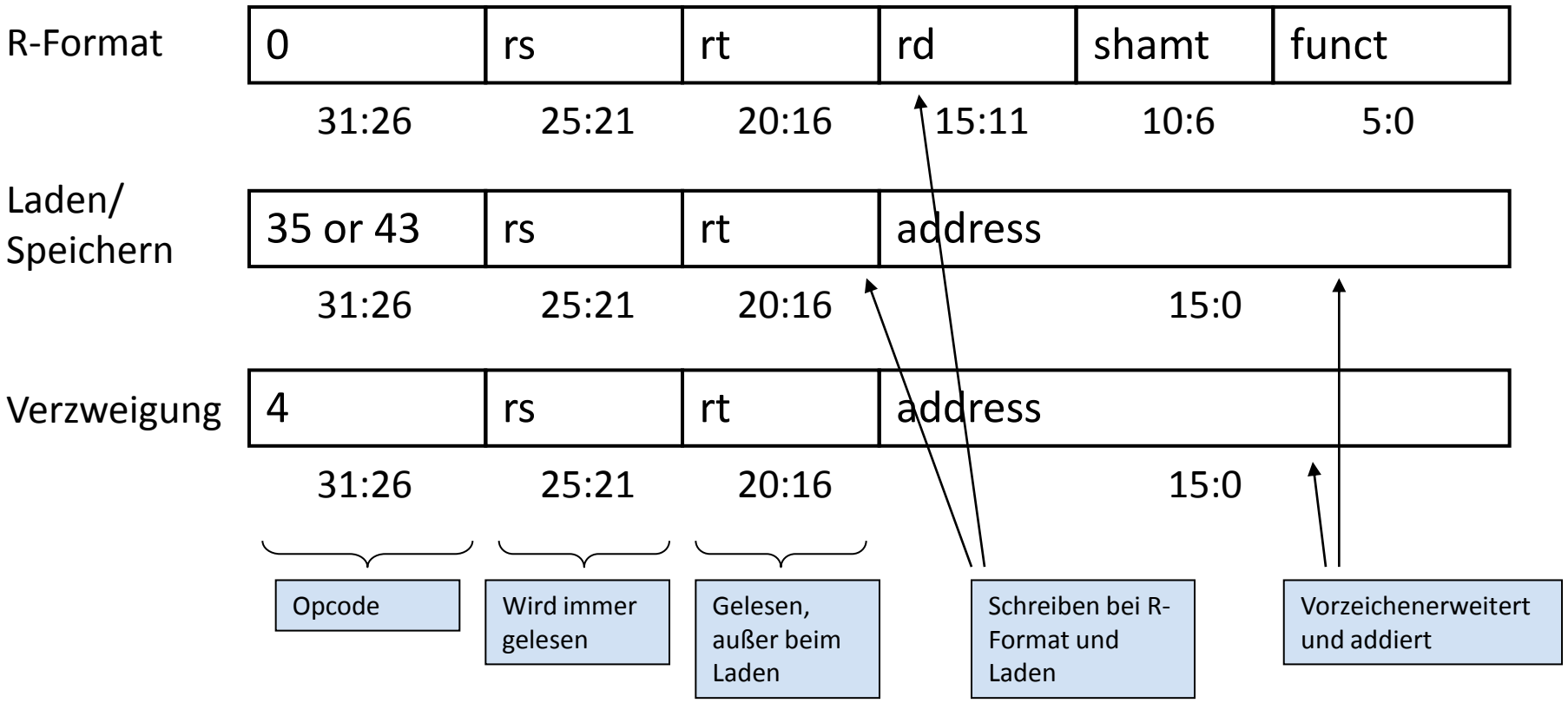
- Zum Setzen der 4 Steuerbits
  - Es gibt 2 ALUOp-Bits, die aus dem Opcode erzeugt werden
  - Mithilfe kombinatorischer Logik werden die 4 Steuerbits ermittelt

Opcode	ALUOp	Operation	funct	ALU-Aktion	ALU-Steuereingang
lw	00	load word	XXXXXX	Addition	0010
sw	00	store word	XXXXXX	Addition	0010
beq	01	branch equal	XXXXXX	Subtraktion	0110
R-type	10	add	100000	Addition	0010
		subtract	100010	Subtraktion	0110
		AND	100100	UND	0000
		OR	100101	ODER	0001
		set-on-less-than	101010	kleiner als	0111

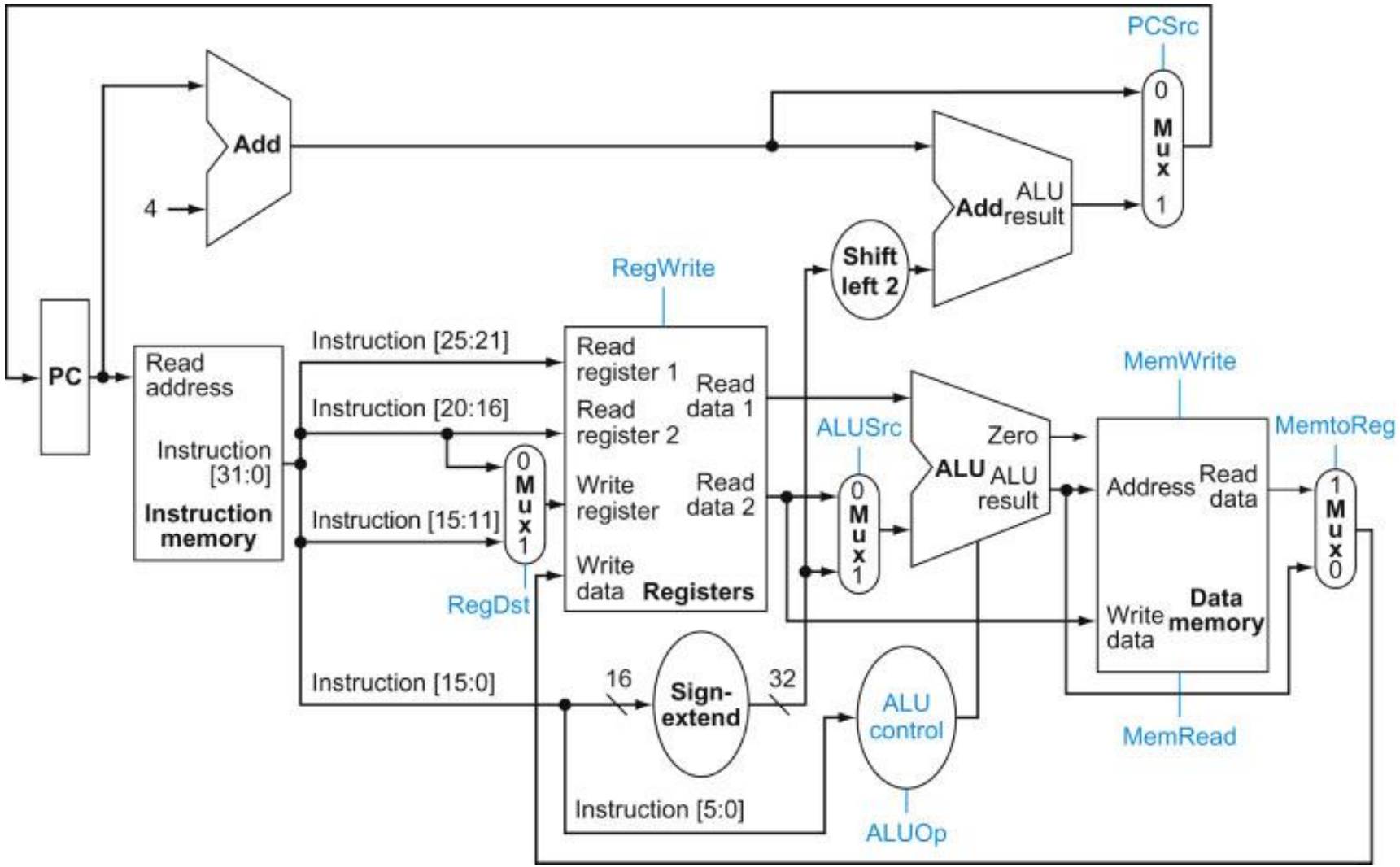
  - X bedeutet „Don't-Care “, d.h. die Bits in Feld funct können beliebige Folgen von Nullen und Einsen sein und das hat hier keine Auswirkung

Hauptsteuereinheit

- Die Steuersignale werden aus den Befehlen erzeugt



Datenpfad (erweitert)

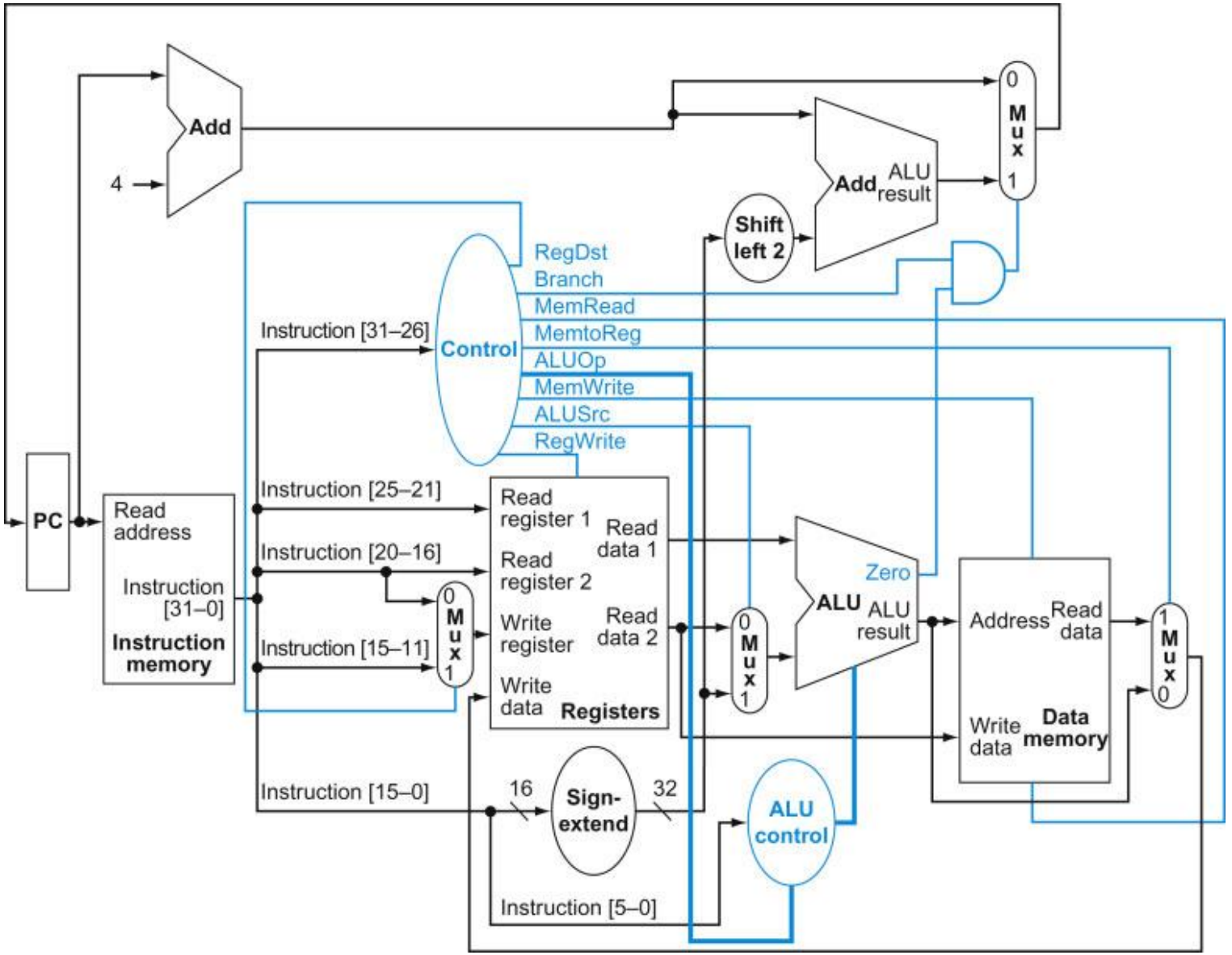


Steuersignale und Belegung der Steuerleitungen

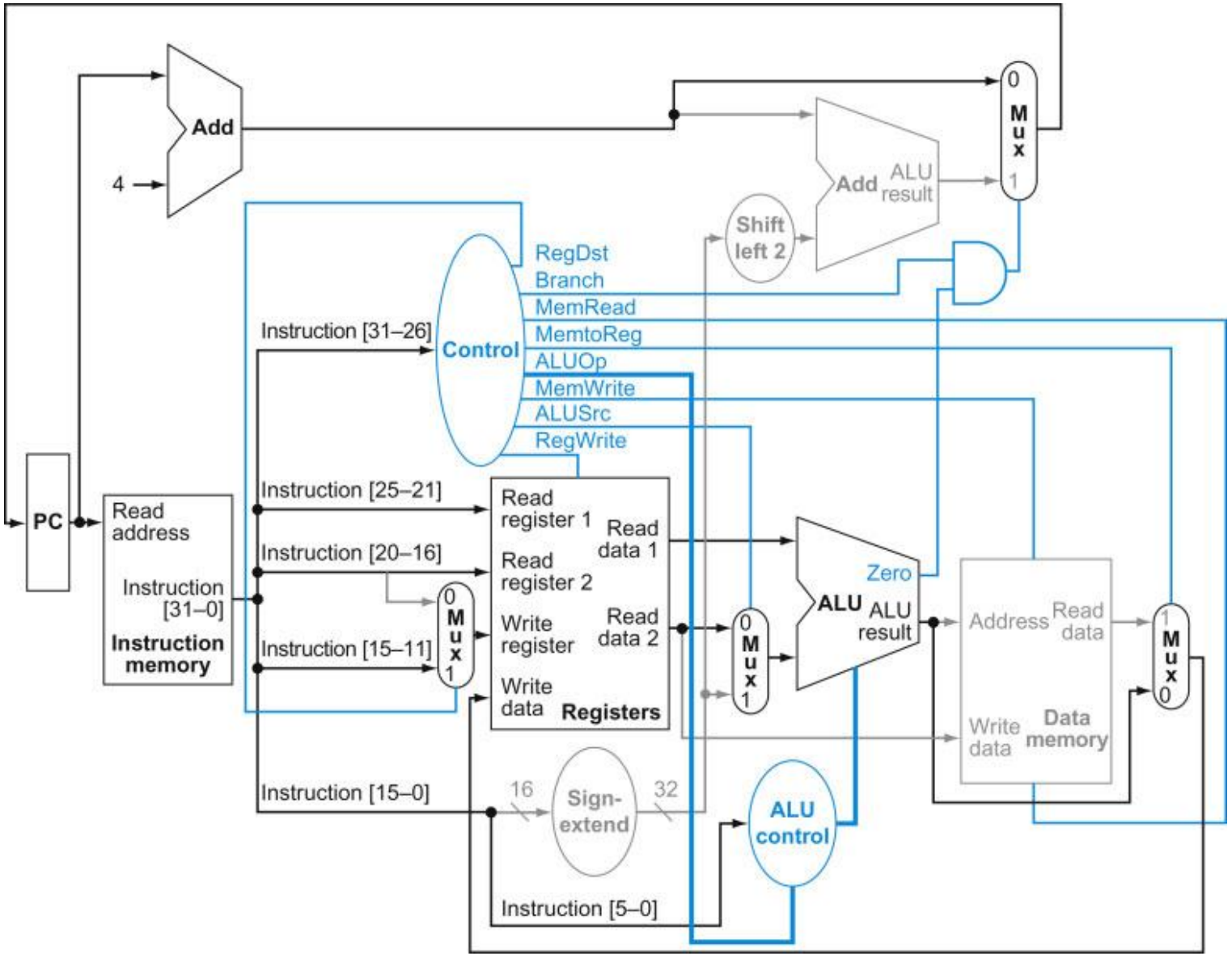
Signalname	Wirkung, wenn logisch 0	Wirkung, wenn logisch 1
`RegDst`	Die Adresse des Zielregisters für den *Write-register*-Eingang wird vom rt-Feld (Bits 20:16) bereitgestellt.	Die Adresse des Zielregisters für den *Write-register*-Eingang wird vom rd-Feld (Bits 15:11) bereitgestellt.
`RegWrite`	Keine.	Das Register am *Write-register*-Eingang wird mit dem Wert am *Write-data*-Eingang beschrieben.
`ALUSrc`	Der zweite ALU-Operand wird vom zweiten Registersatzausgang (*Read data 2*) bereitgestellt.	Der zweite ALU-Operand besteht aus den vorzeichenerweiterten, unteren 16 Bits des Befehls.
`PCSrc`	Der Befehlszählerwert wird durch den Ausgangswert des Addierers ersetzt, der den Befehlszählerwert und 4 addiert.	Der Befehlszählerwert wird durch den Ausgangswert des Addierers ersetzt, der das Sprungziel berechnet.
`MemRead`	Keine.	Durch den Adresseingang bestimmter Datenspeicherinhalt wird an den *Read-data*-Ausgang (*data memory*) gelegt.
`MemWrite`	Keine.	Durch den Adresseingang bestimmter Datenspeicherinhalt wird durch den Wert am *Write-data*-Eingang (*data memory*) ersetzt.
`MemtoReg`	Der am *Write-data*-Eingang des Registers angelegte Wert wird von der ALU bereitgestellt.	Der am *Write-data*-Eingang des Registers angelegte Wert wird vom Datenspeicher bereitgestellt.

Befehl	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp2
R-Format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

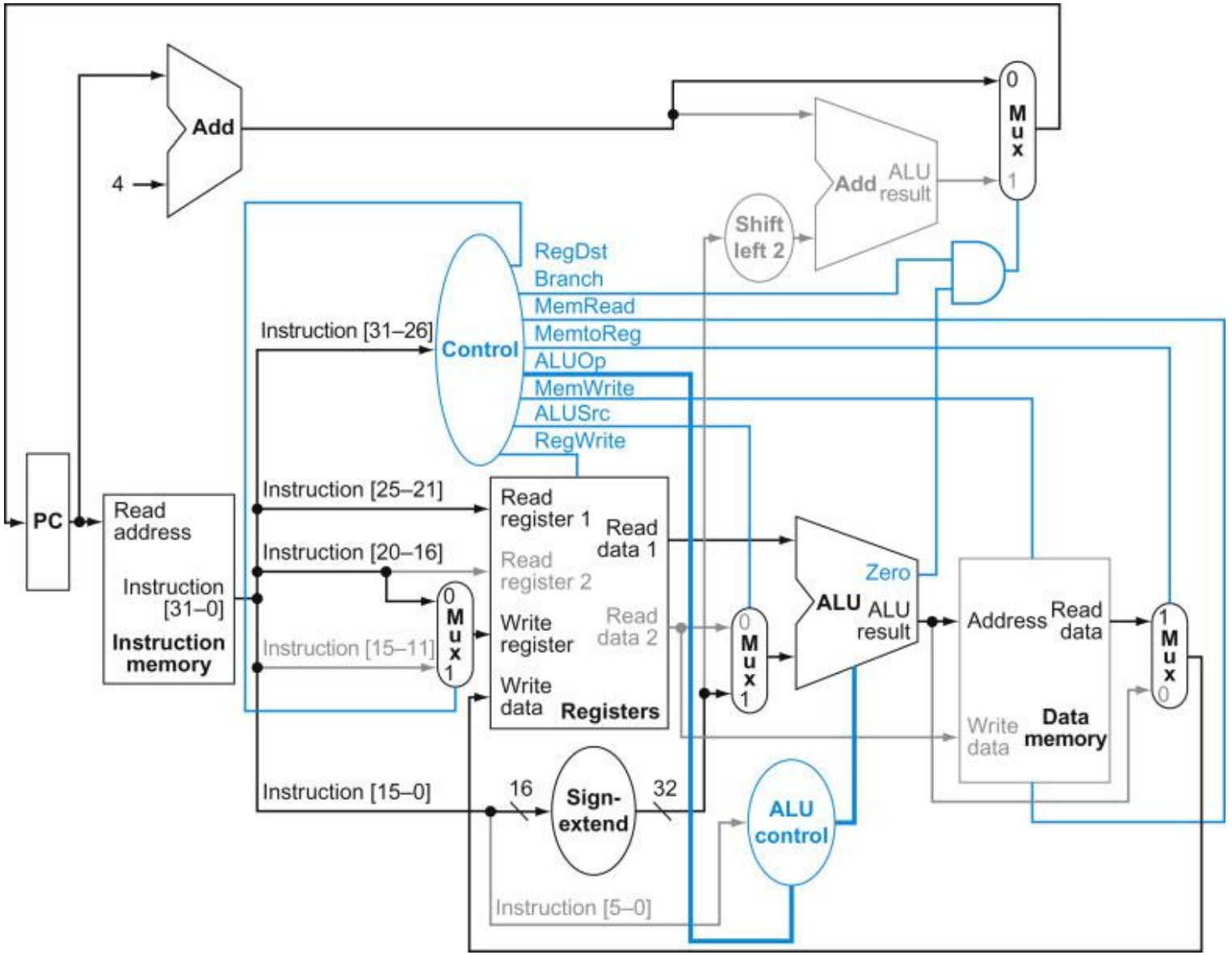
Datenpfad mit Steuereinheit



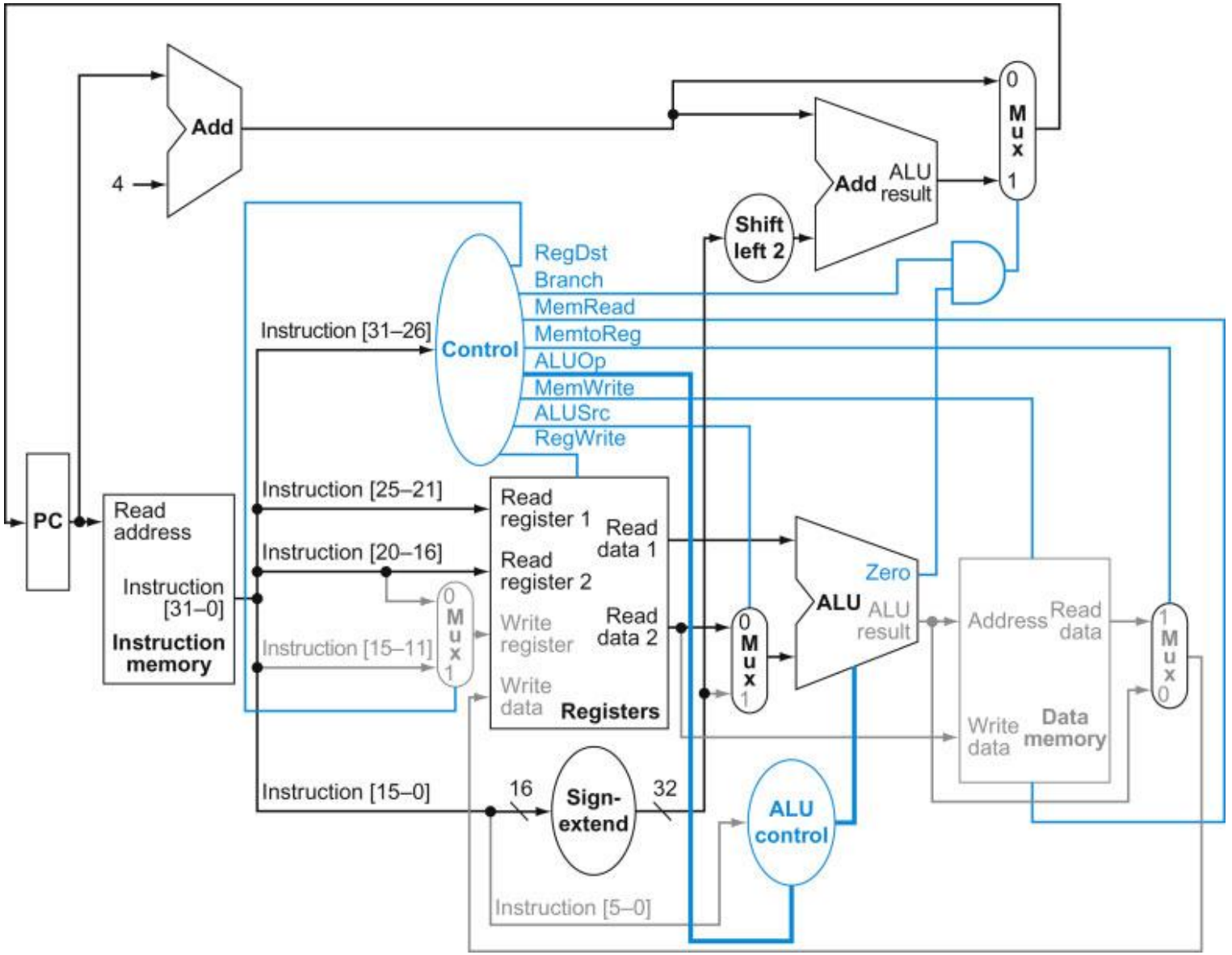
Datenpfad für Befehl im R-Format



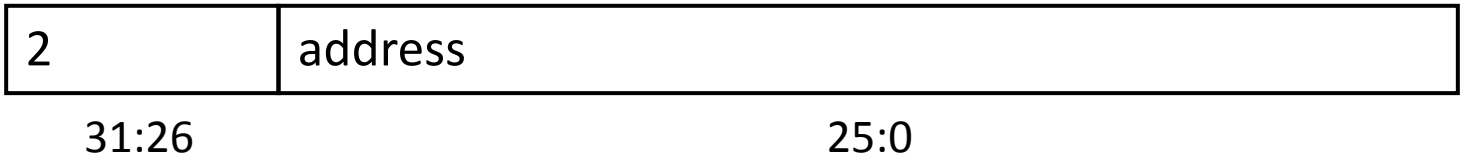
Datenpfad für Ladebefehl



Datenpfad für beq-Befehl

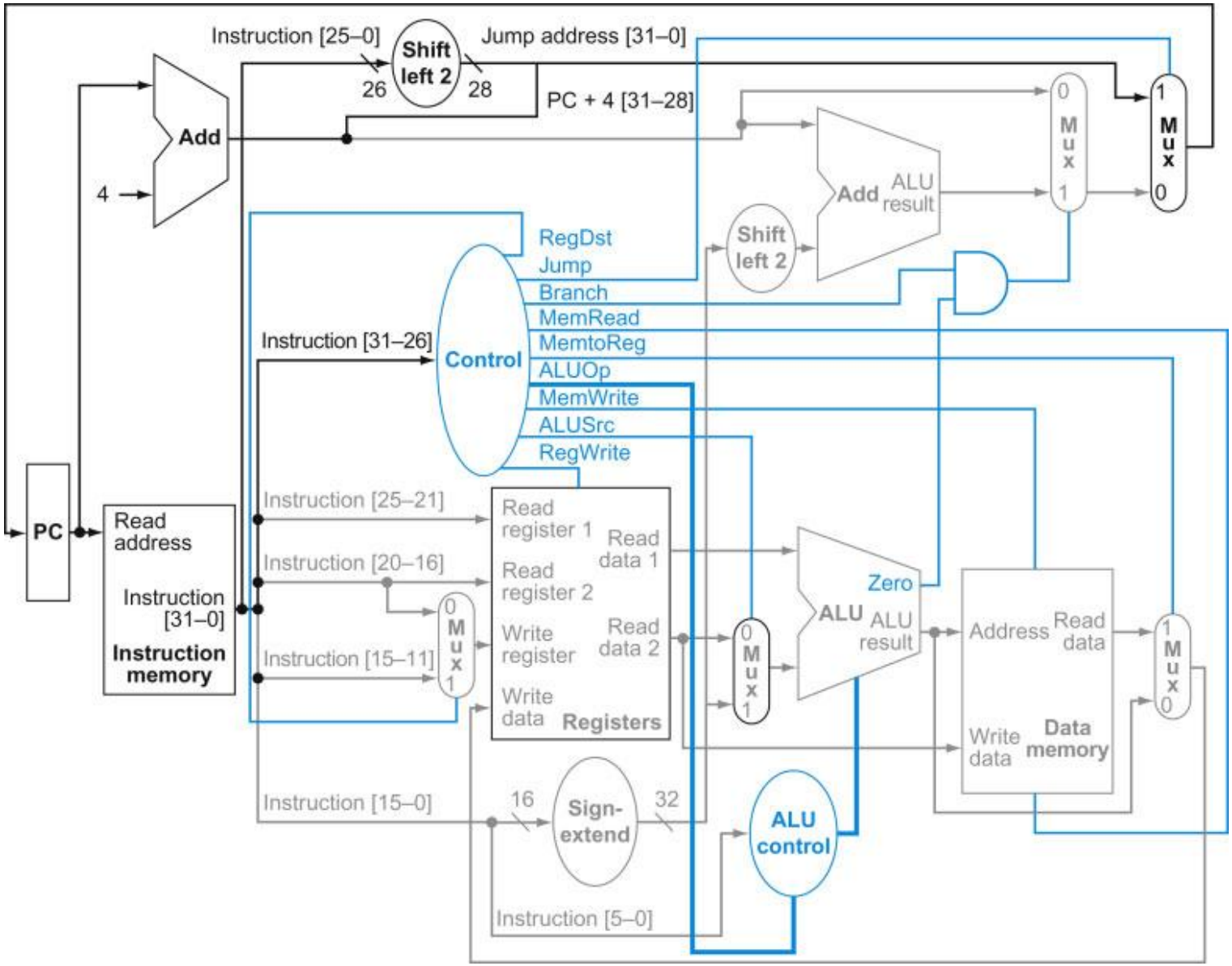


Implementierung von Sprüngen



- Sprünge benutzen Wortadressen
- Neuer Befehlszähler setzt sich zusammen aus
  - Die oberen 4 Bits des aktuellen Befehlszählers
  - 26 Bit der Sprungadresse im Befehl
  - 00 (Shift um 2 nach links, Multiplikation mit 4)
- Es wird ein zusätzliches Steuersignal benötigt (aus dem Opcode abgeleitet)

Datenpfad für Sprungbefehl



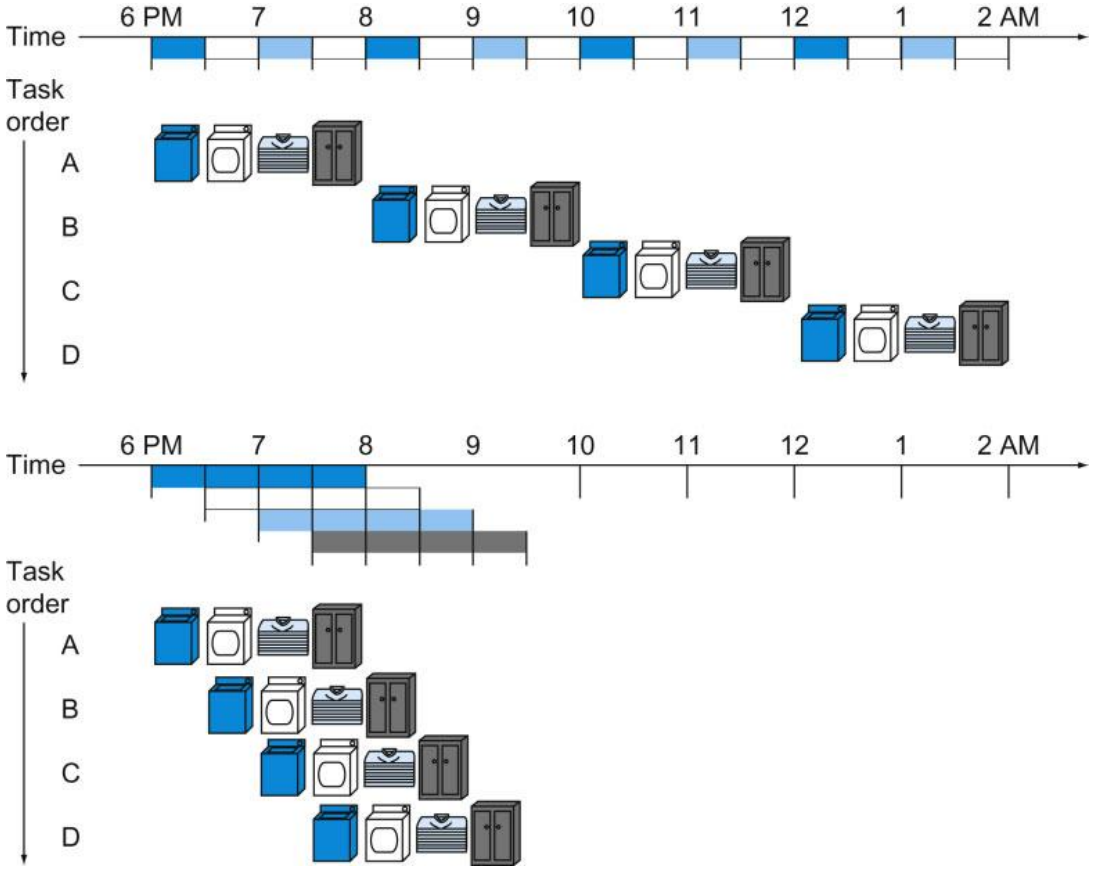
Probleme

- Die längste Verzögerung bestimmt die Taktperiode
  - Kritischer Pfad: `lw` Befehl
  - Befehlsspeicher → Registersatz → ALU → Datenspeicher → Registersatz
- Es ist nicht möglich die Taktperiode für unterschiedliche Befehle zu variieren
  - Verletzt eines unserer Entwurfsprinzipien
  - Optimiere den häufig vorkommenden Fall
- Lösung?
  - Pipelining

PIPELINING

Pipelining

- Pipelining: Eine Implementierungstechnik, bei der mehrere Befehle ähnlich wie bei einem Fließband überlappend ausgeführt werden
- Analogie: Wäschewaschen
  - Parallele Verarbeitung verbessert die Leistung



- Vier Waschgänge
  - Speedup
    = 8/3.5 = 2.3
- Non-stop
  - Speedup
    = 2n/0.5n + 1.5 ≈ 4
    = Anzahl der Stufen

MIPS Pipeline

- MIPS-Befehle werden in fünf Schritten (bzw. Stufen) ausgeführt

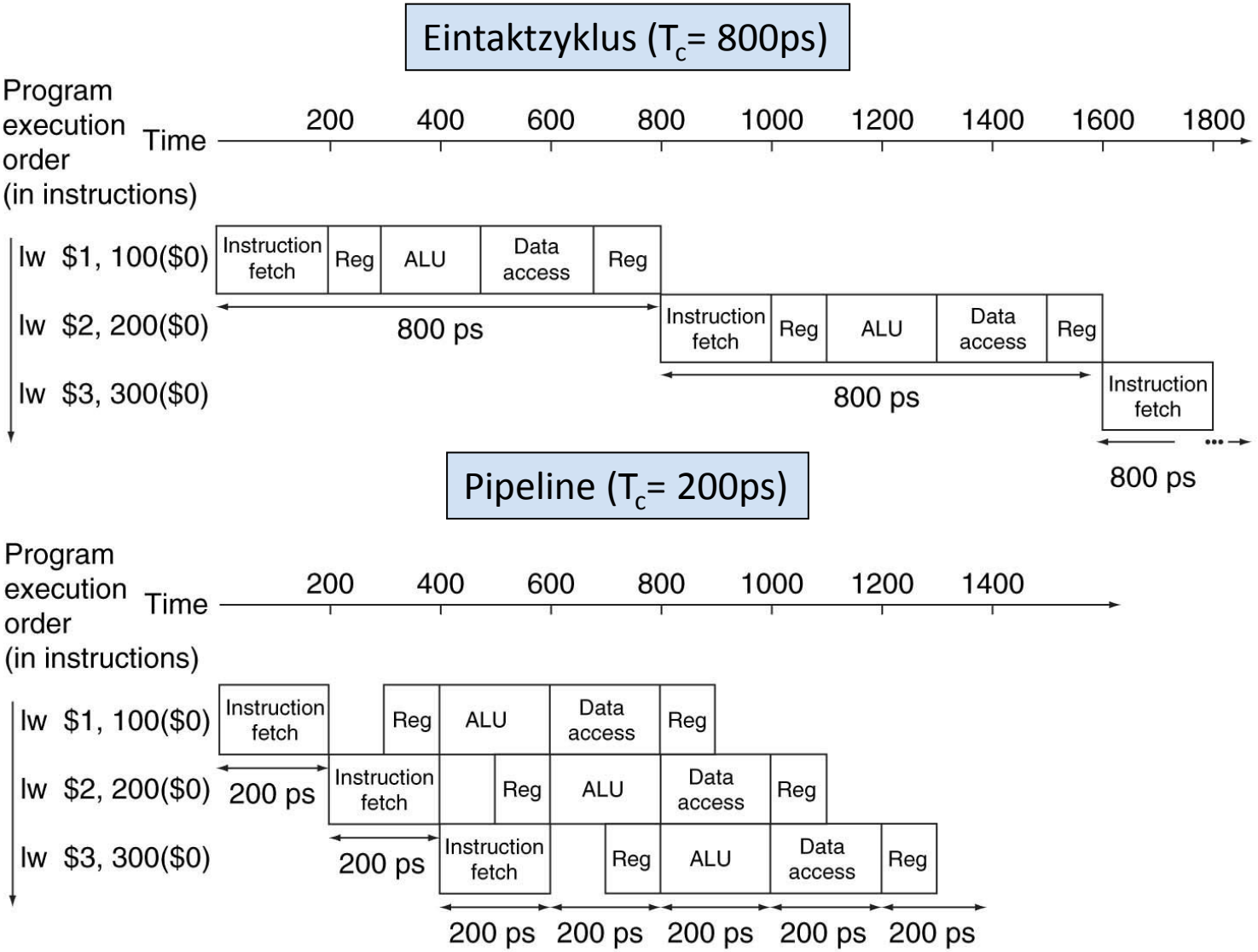
1. IF (*instruction fetch*)
   - Holen des Befehls aus dem Speicher
2. ID (*instruction decode*)
   - Lesen der Register und gleichzeitiges Entschlüsseln des Befehls
3. EX (*execute*)
   - Ausführen der Operation oder Berechnen einer Adresse
4. MEM (*memory access*)
   - Zugriff auf den Datenspeicher
5. WB (*write back*)
   - Schreiben des Ergebnisses in ein Register

Leistung der Pipeline (Zeiten für Eintaktzyklus)

- Es werden folgende Zeiten angenommen
  - 100 ps für das Schreiben bzw. Lesen von Registern
  - 200 ps für alle anderen Stufen

Befehl	IF	ID	EX	MEM	WB	Gesamtzeit
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Leistung der Pipeline (Eintaktzyklus - Pipelining)



Befehlssätze für Pipelining

- MIPS-ISA wurde für Pipelining entworfen
- Alle Befehle haben 32 Bits
  - Einfacher in einem Zyklus zu holen und zu dekodieren
  - Gegenbeispiel x86 (unterschiedlich lange Befehle)
- Wenige und einheitliche Formate
  - Dekodieren und Lesen der Register kann in einem Schritt stattfinden
- Speicheroperanden nur bei Lade- bzw. Speicherbefehlen
  - Adresse kann in der dritten Stufe berechnet werden
  - Zugriff auf den Speicher erfolgt in der vierten Stufe

Konflikte

- Einschränkung beim Pipelining
  - **Pipelinekonflikte** verhindern, dass der nächste Befehl im nachfolgenden Taktzyklus ausgeführt werden kann
- Es gibt drei verschiedene Typen
  - **Strukturkonflikte** (*structural hazard*)
  - **Datenkonflikte** (*data hazard*)
  - **Steuerkonflikte** (*control hazard*)

Strukturkonflikte

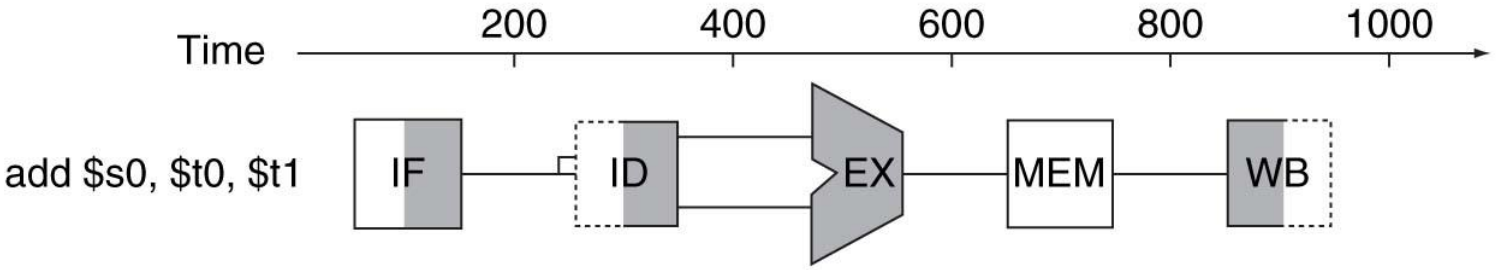
- Konflikt bei der Benutzung einer Ressource
  - Zwei Befehle müssen gleichzeitig (in unterschiedlichen Pipeline-Stufen) auf eine Ressource zugreifen
- Beispiel in einer MIPS-Pipeline mit nur einem Speicher
  - Laden/Speichern erfordert Datenzugriff
  - Holen eines Befehls und gleichzeitiger Zugriff auf den Speicher durch einen anderen (bereits weiter fortgeschrittenen) Befehl
    - Pipeline müsste den ersten Befehl verzögern
    - Einfügen von einem Leertakt (*pipeline stall*, *bubble*)
- Daher benötigt ein Datenpfad mit Pipelining separate Befehls- und Datenspeicher (siehe MIPS Datenpfad)
  - Oder separate Befehls- und Daten-Caches

Datenkonflikte (1)

- Ein Befehl hängt vom Abschluss eines Datenzugriffs eines anderen Befehls ab

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

- Grafische Darstellung der MIPS-Pipeline für add

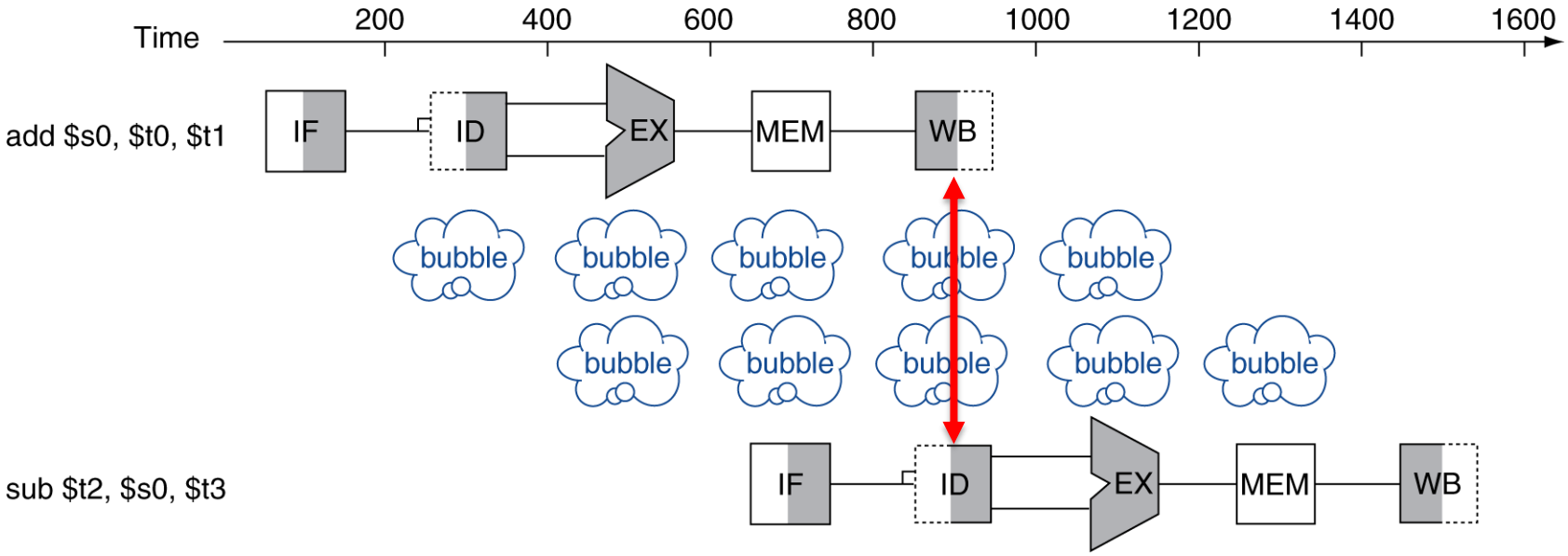


  - 5 Elemente (Befehlsspeicher, Register zum Lesen, ALU, Datenspeicher, Register zum Schreiben)
  - Grauschattierung gibt an, dass das Element verwendet wird
    - Grauschattierung in der rechten Hälfte
      - Element wird in dieser Stufe gelesen
    - Grauschattierung in der linken Hälfte
      - Element wird in dieser Stufe geschrieben

Datenkonflikte (2)

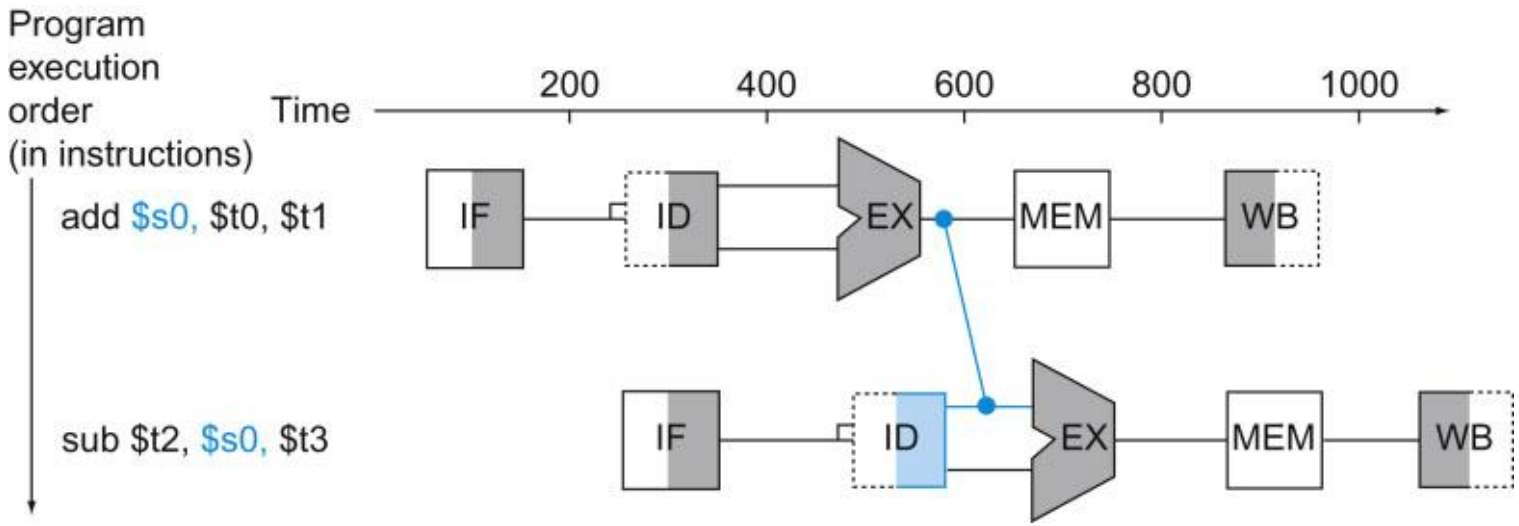
- Datenkonflikt bei

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



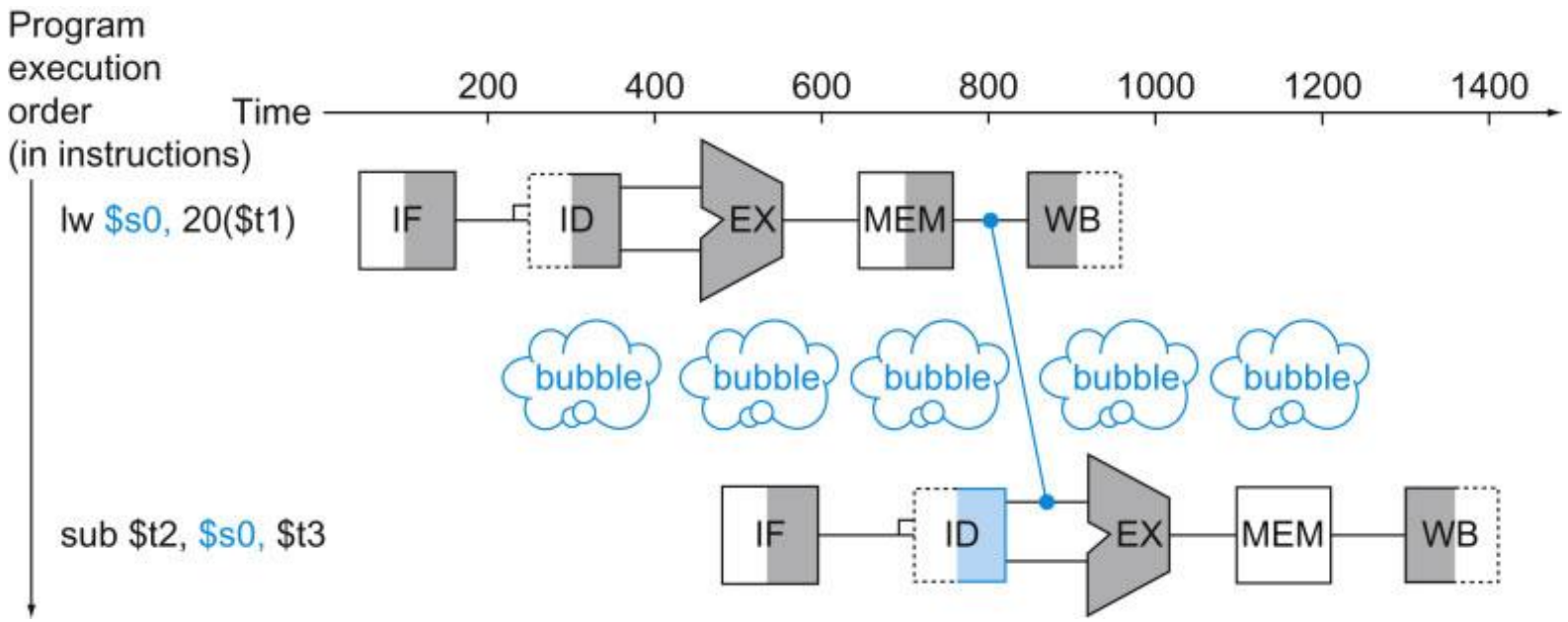
Forwarding (Bypassing)

- Forwarding
  - Das fehlende Datenelement wird aus internen Pufferspeichern abgerufen
  - Es wird nicht darauf gewartet, bis dieses aus den für den Programmierer sichtbaren Registern oder aus dem Speicher kommt
- Forwarding erfordert zusätzliche Verbindungsleitungen



Load-use-Konflikt

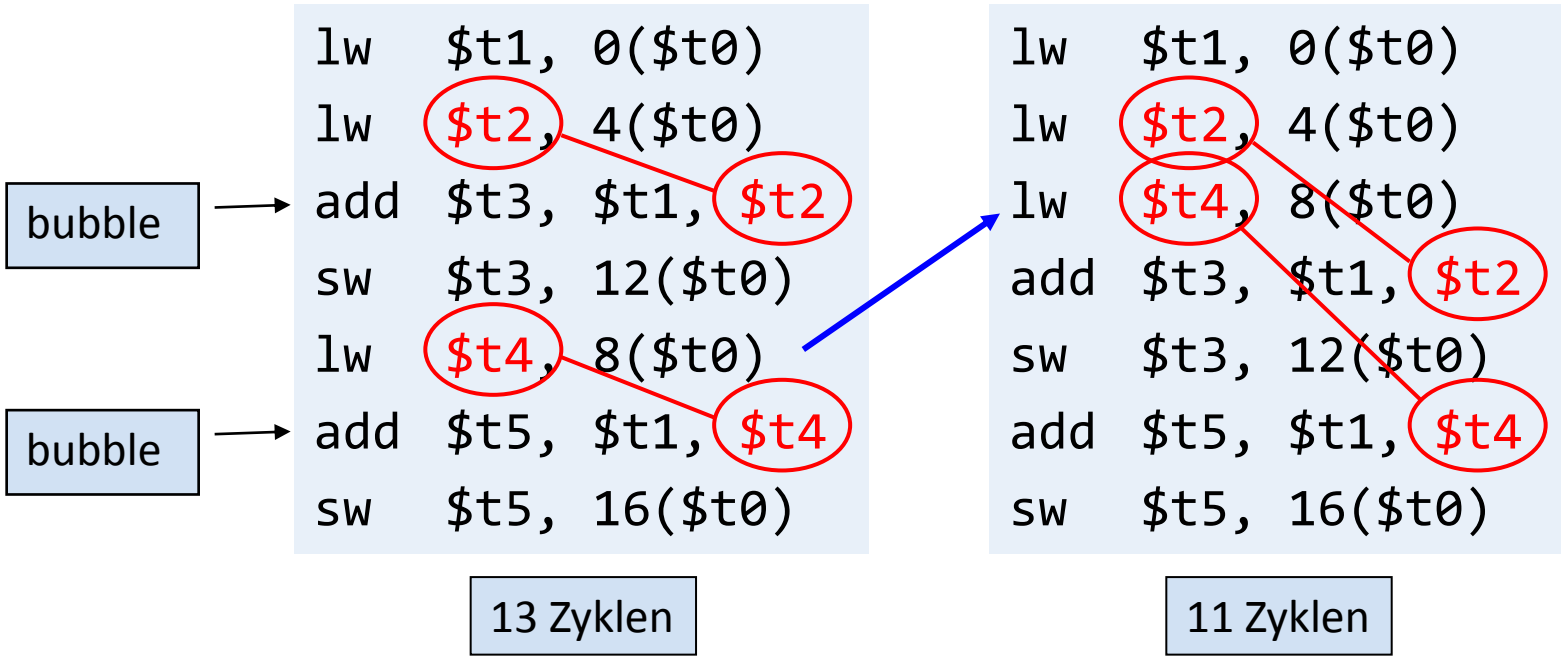
- Forwarding kann nicht alle Konflikte verhindern
- Load-use-Konflikt (*load-use data hazard*)



- Spezielle Behandlung dieser Situation
  - Hardware- oder Softwareerkennung (Code umordnen)

Umordnen von Code

- Code umordnen um Load-use-Konflikte zu vermeiden
  - Auf Prozessor mit Pipelining und Forwarding



Steuerkonflikte

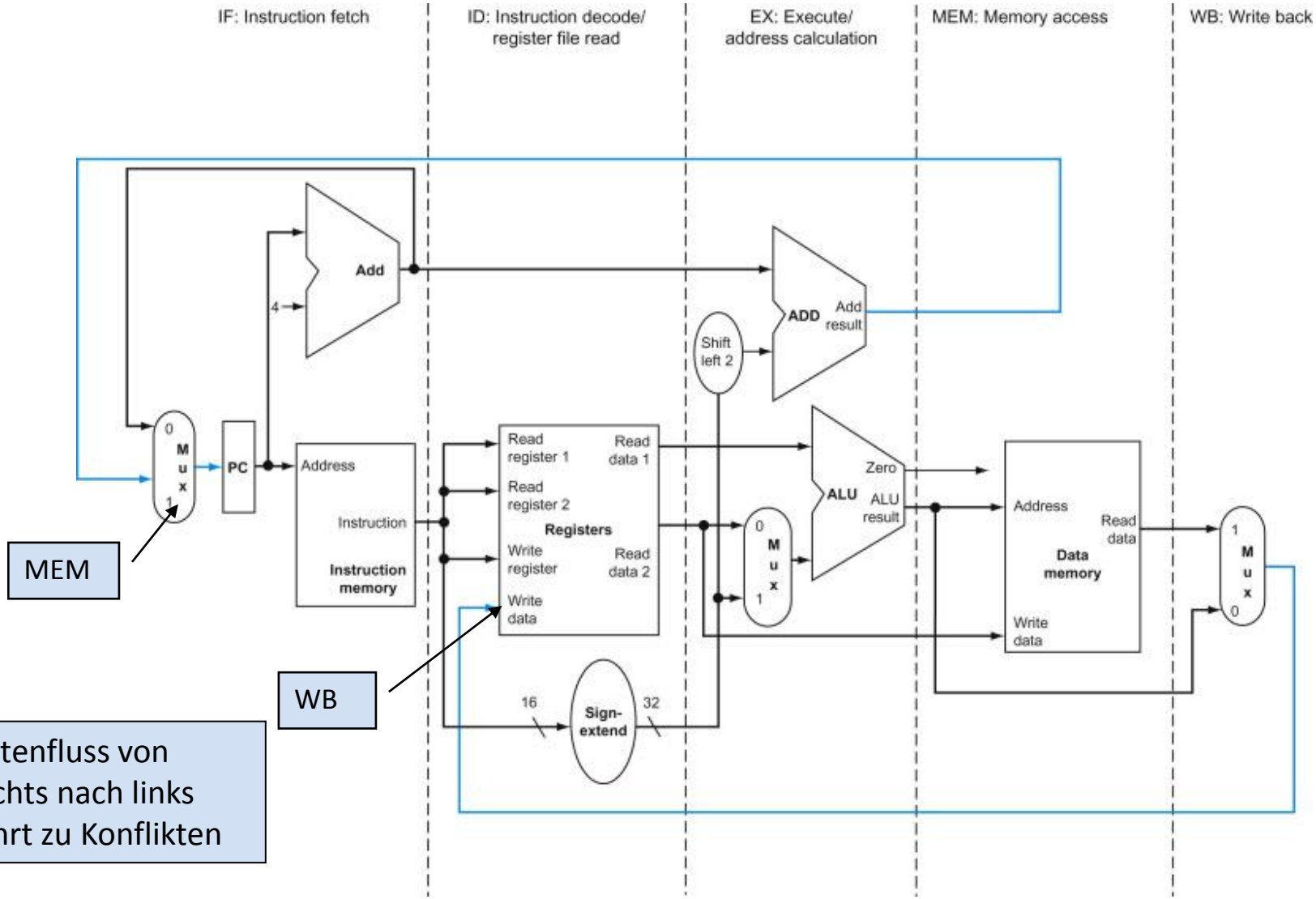
- Bedingte Verzweigung bestimmt den Kontrollfluss in einem Programm
  - Der nächste Befehl hängt vom Ausgang der Bedingungsauswertung ab
  - Die Pipeline kann nicht immer den korrekten nächsten Befehl holen
    - Befehl für die Verzweigung befindet sich erst in ID-Stufe
- Konservativer Ansatz
  - Bei einem Befehl für eine bedingte Verzweigung solange warten (Bubbles benutzen), bis das Sprungziel berechnet ist und dann neuen Befehl holen

Sprungvorhersage

- Bei längeren Pipelines kann das Ergebnis der Auswertung nicht früh bestimmt werden
  - Große Verzögerung, wenn man auf das Ergebnis wartet
- Daher verwenden moderne Prozessoren eine Sprungvorhersage
  - Es gibt nur Leertakte, wenn die Vorhersage falsch ist
- MIPS-Pipeline
  - Einfacher Ansatz: Immer vorhersagen, dass Sprünge nicht ausgeführt werden (*untaken branch*, *branch not taken*)
  - Die Pipeline holt die nächsten Befehle im Code
    - Keine Verzögerung
    - Wenn Sprung, dann werden die geholten Befehle abgebrochen

PIPELINING IM DATENPFAD

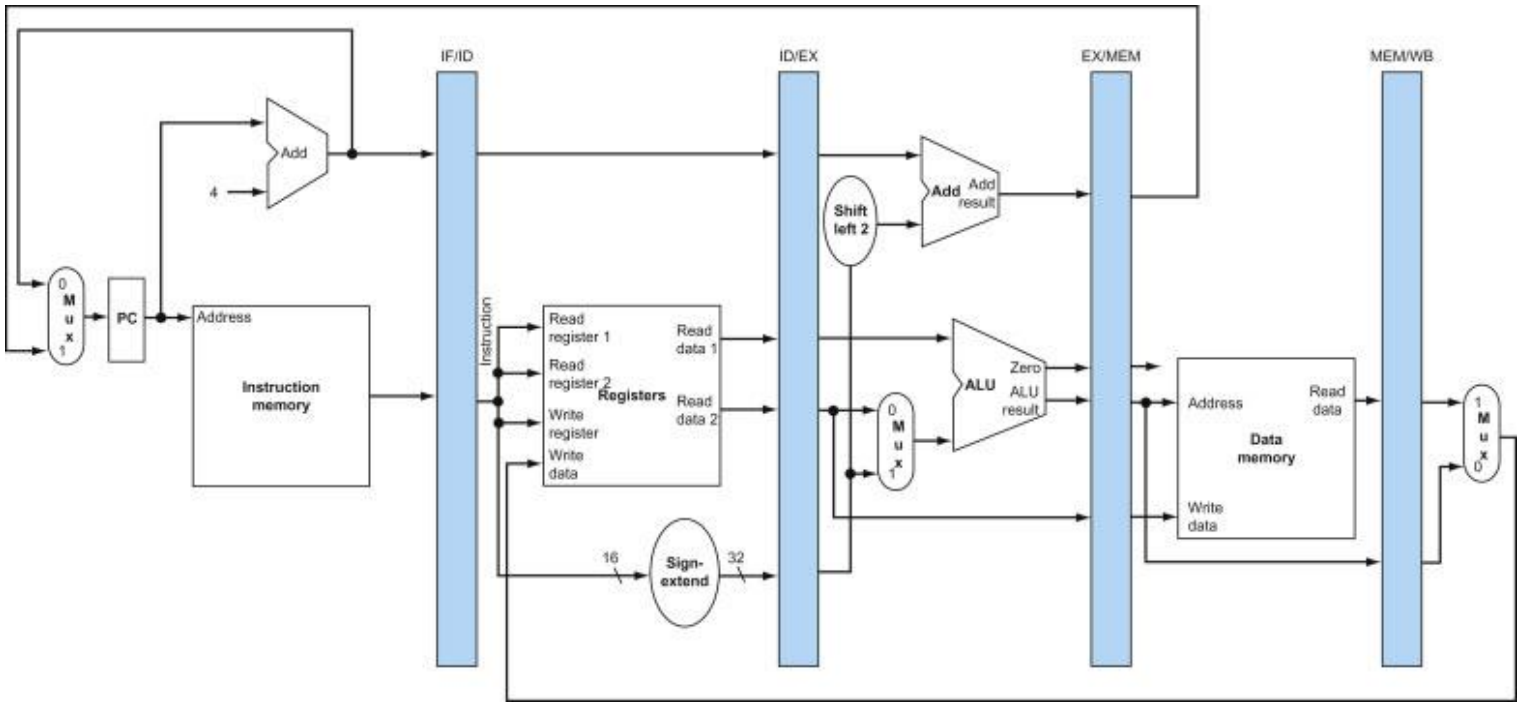
Pipelining im MIPS-Datenpfad



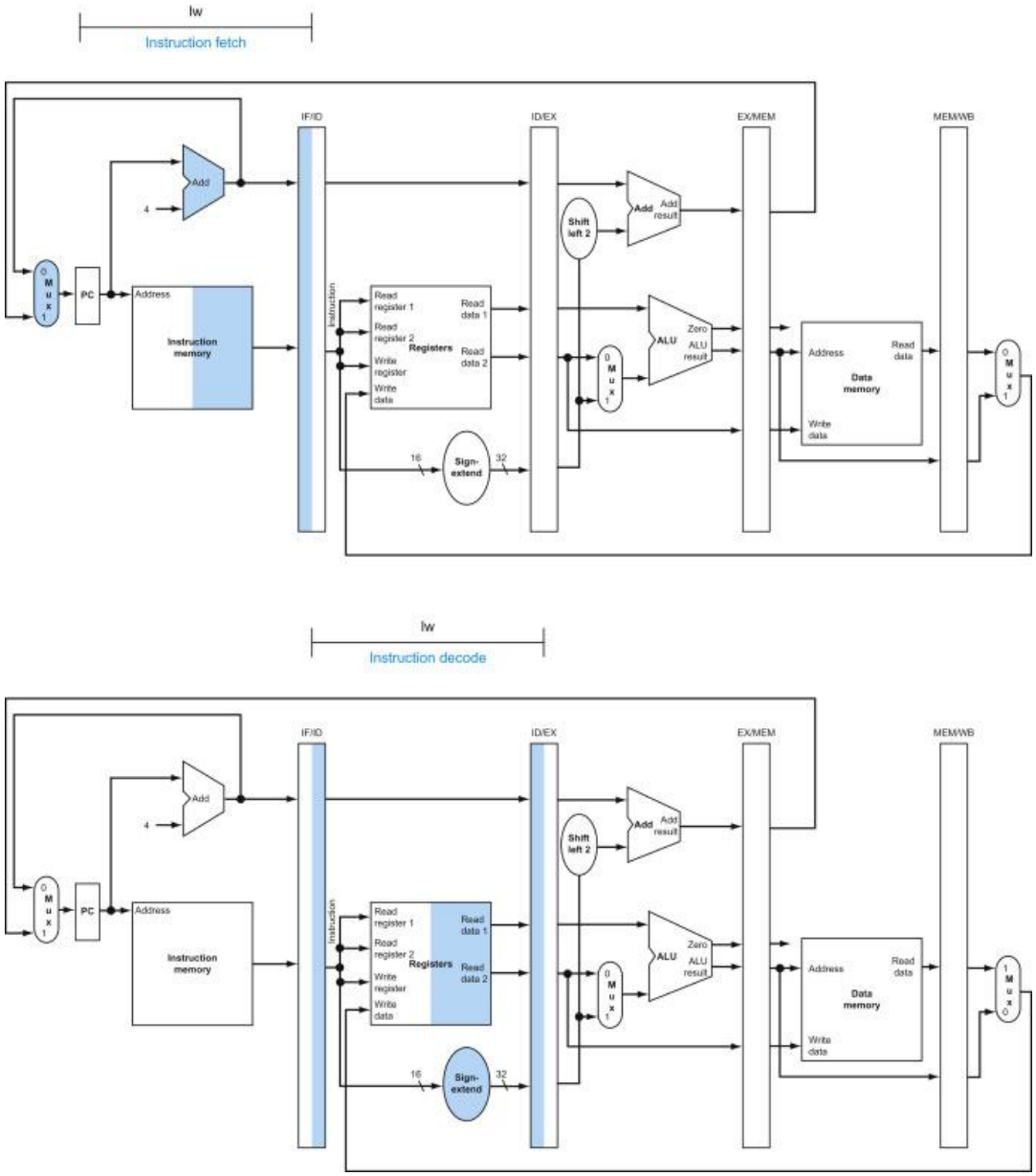
Datenfluss von rechts nach links führt zu Konflikten

Pipelineregister

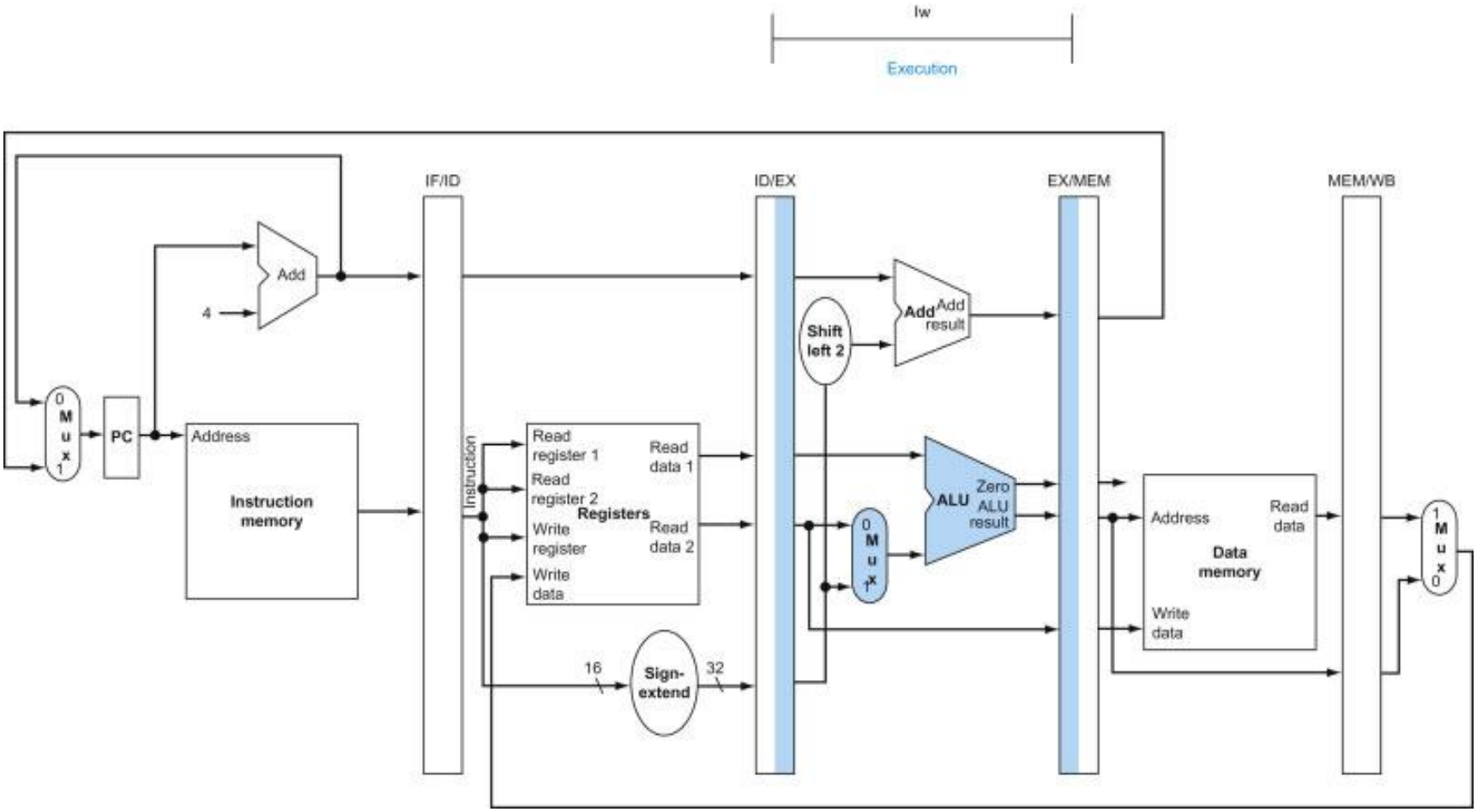
- Pipelineregister zwischen den einzelnen Stufen der Pipeline
  - Speichern die Informationen aus dem vorherigen Zyklus
  - Damit können mehrere Befehle in der Pipeline ausgeführt werden



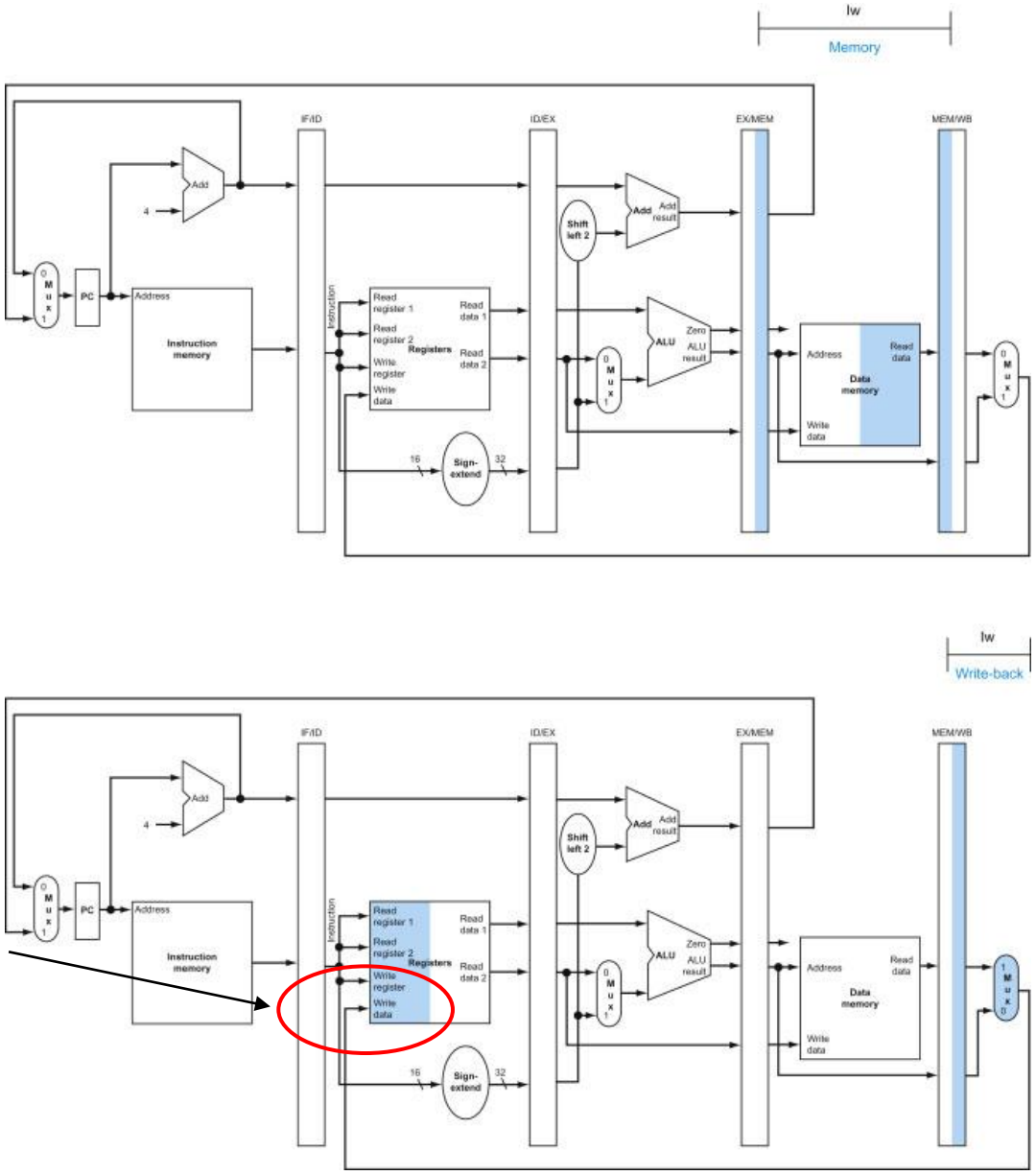
IF-Stufe und ID-Stufe (Laden)



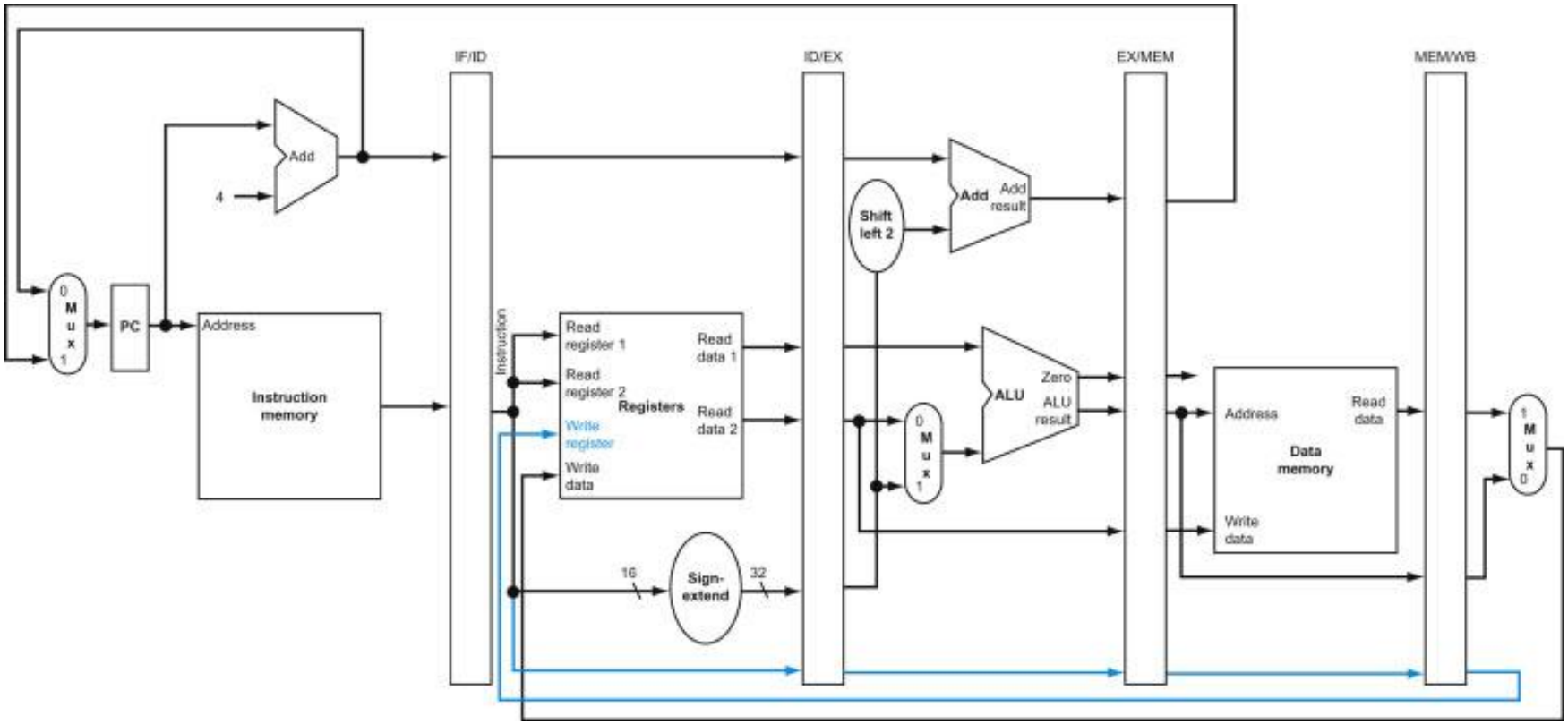
EX-Stufe (Laden)



MEM-Stufe und WB-Stufe (Laden)

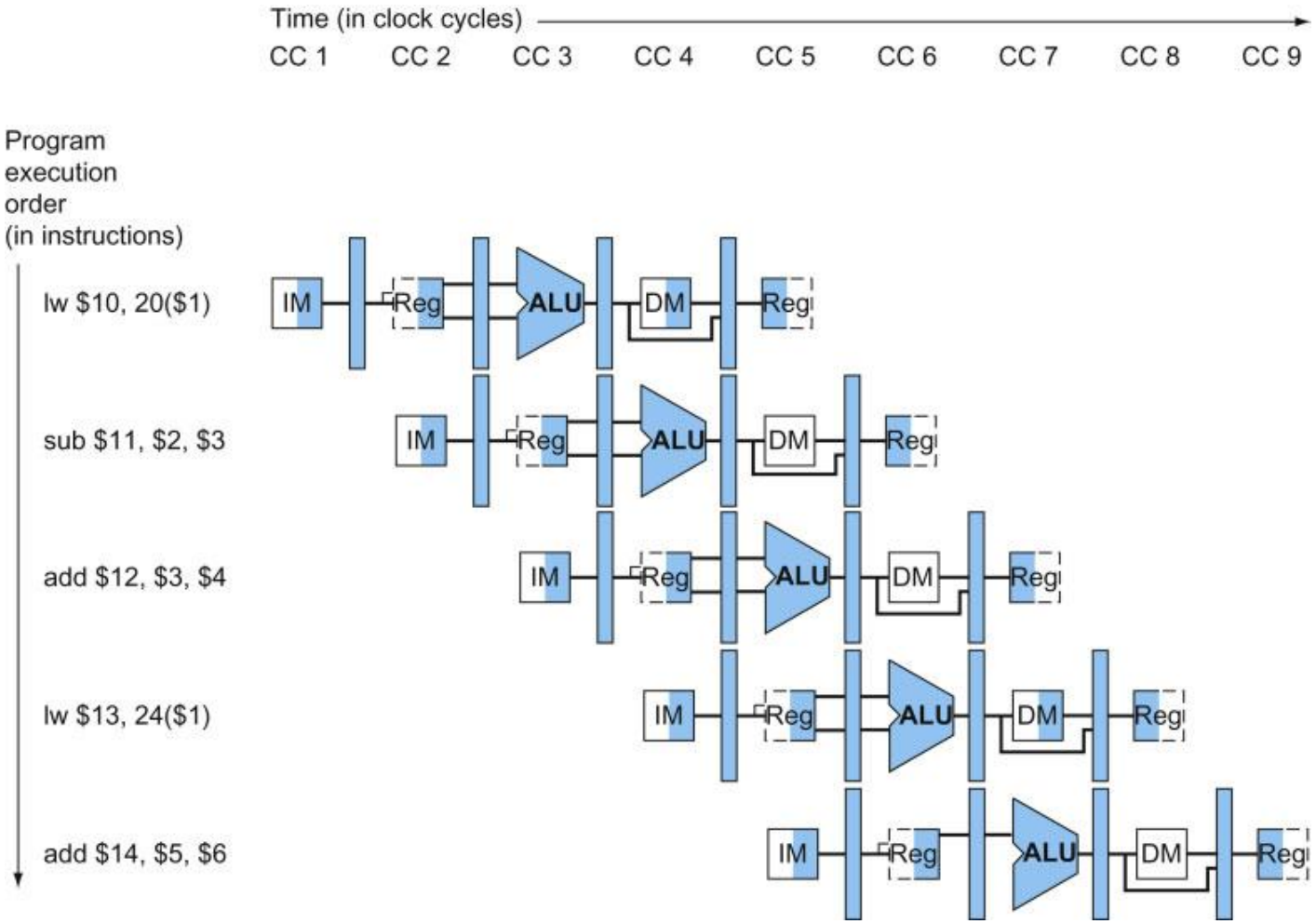


Korrigierter Datenpfad für `lw`



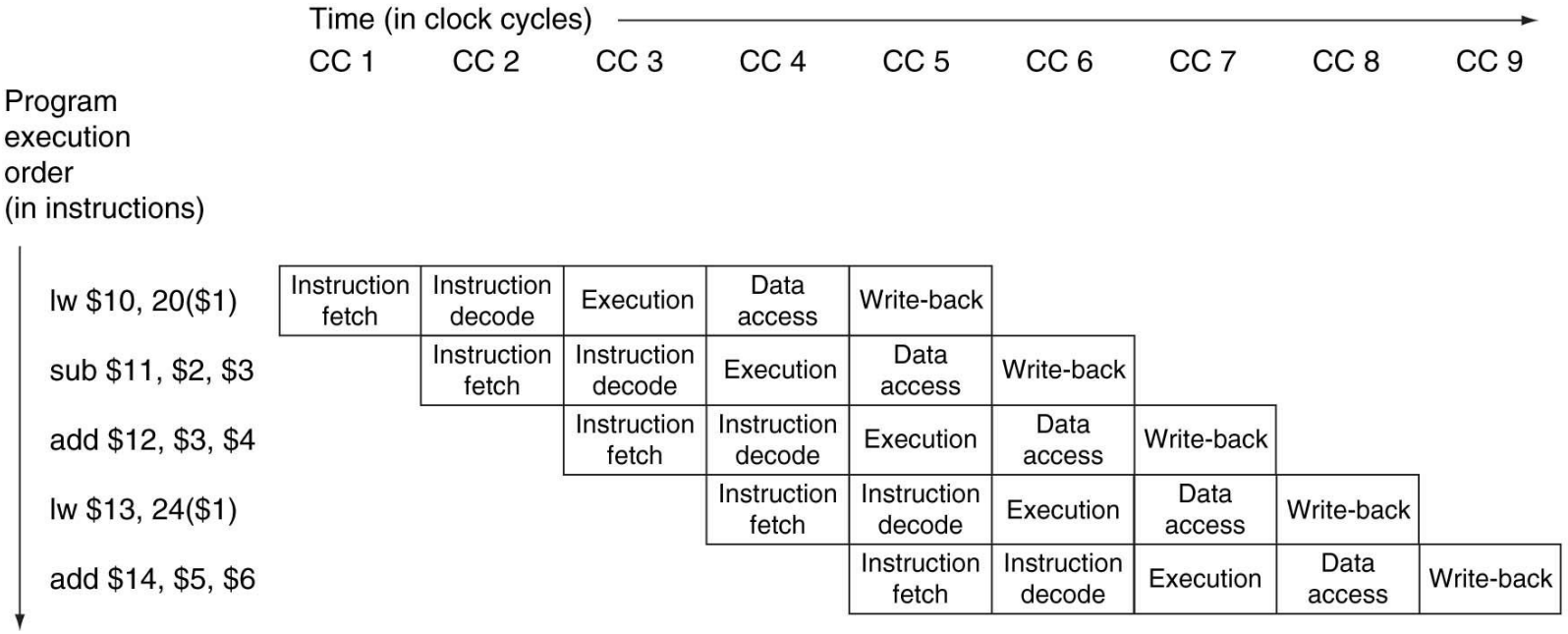
Mehrzyklen-Pipelinediagramm (1)

- Zeigt die vollständige Ausführung von Befehlen



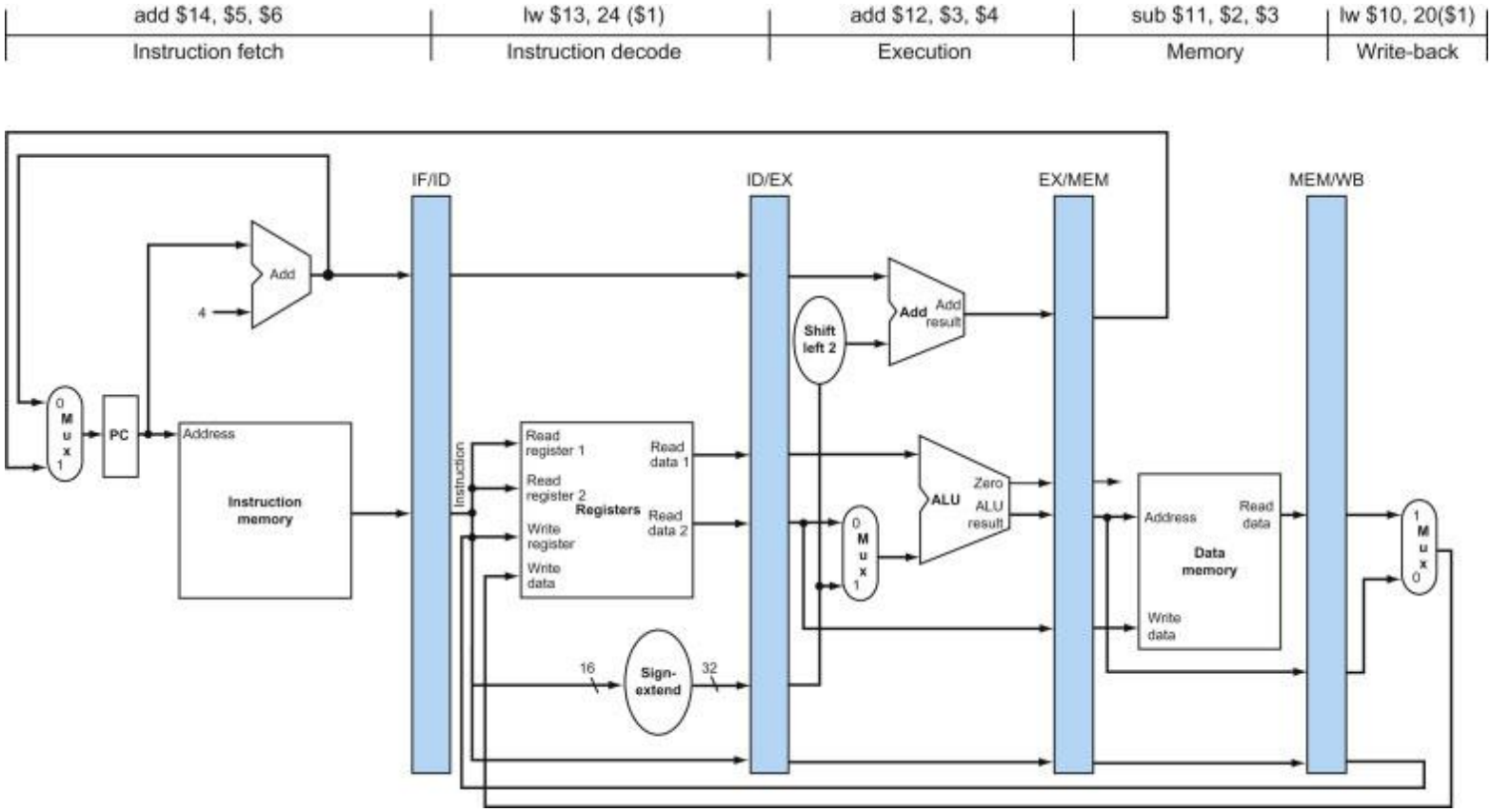
Mehrzyklen-Pipelinediagramm (2)

- Traditionelle Form



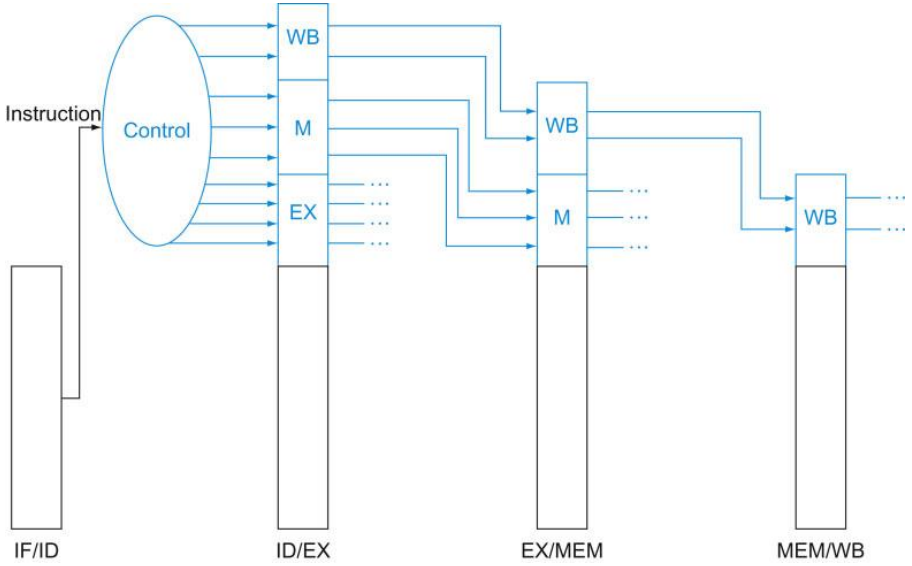
Einzyklen-Pipelinediagramm

- Zeigen den Zustand des gesamten Datenpfades während eines Taktzyklus
- Beispiel (CC5 aus vorheriger Folie)

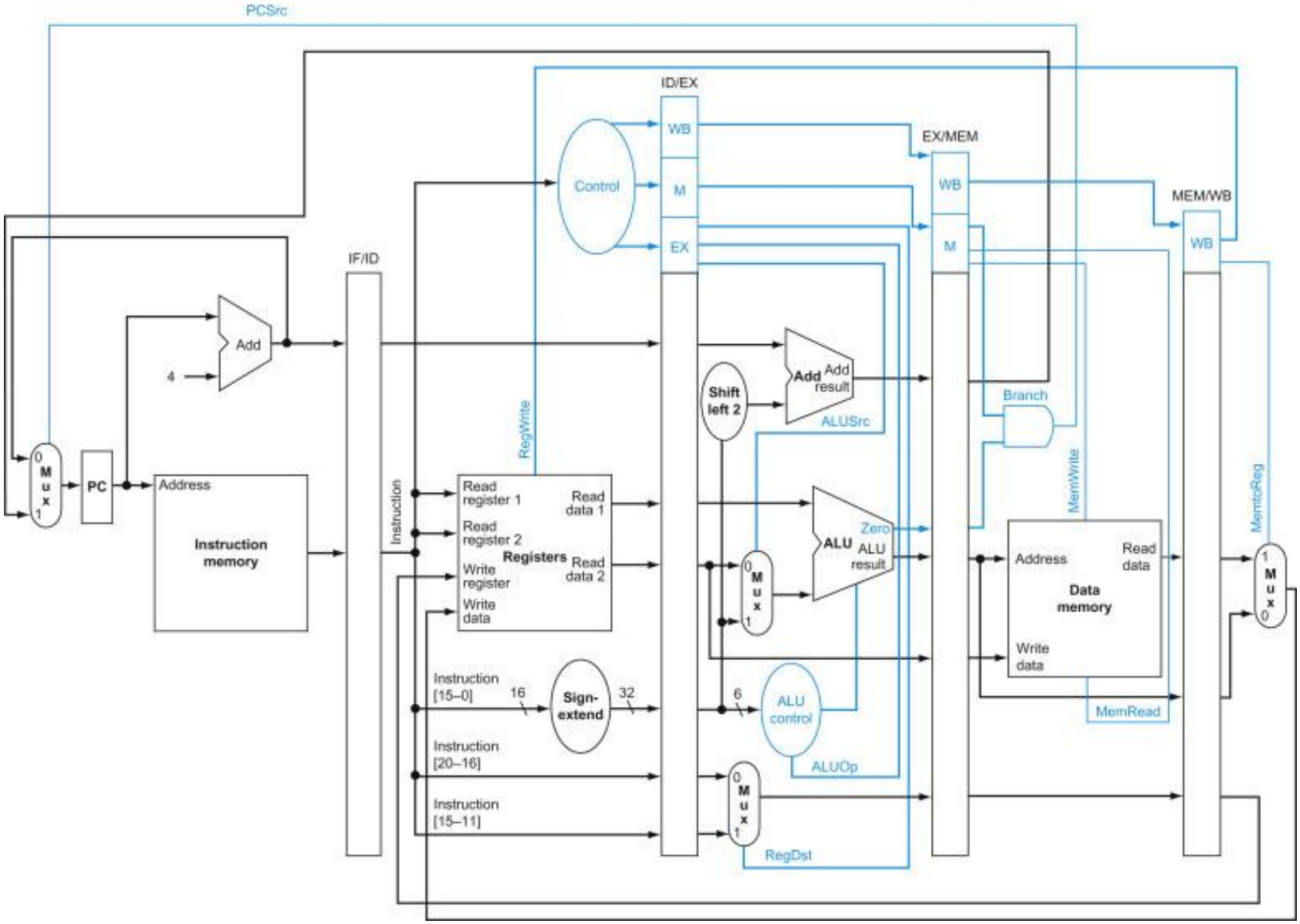


Pipelining der Steuerung (1)

- Steuersignale werden aus den Befehlen abgeleitet
  - Wie in der Eintaktimplementierung
  - Müssen aber zwischengespeichert werden
  - EX-Stufe
    - RegDst, ALUOp, ALUSrc
  - MEM-Stufe
    - Branch, MemRead, MemWrite
  - WB-Stufe
    - RegWrite, MemtoReg



Pipelining der Steuerung (2)



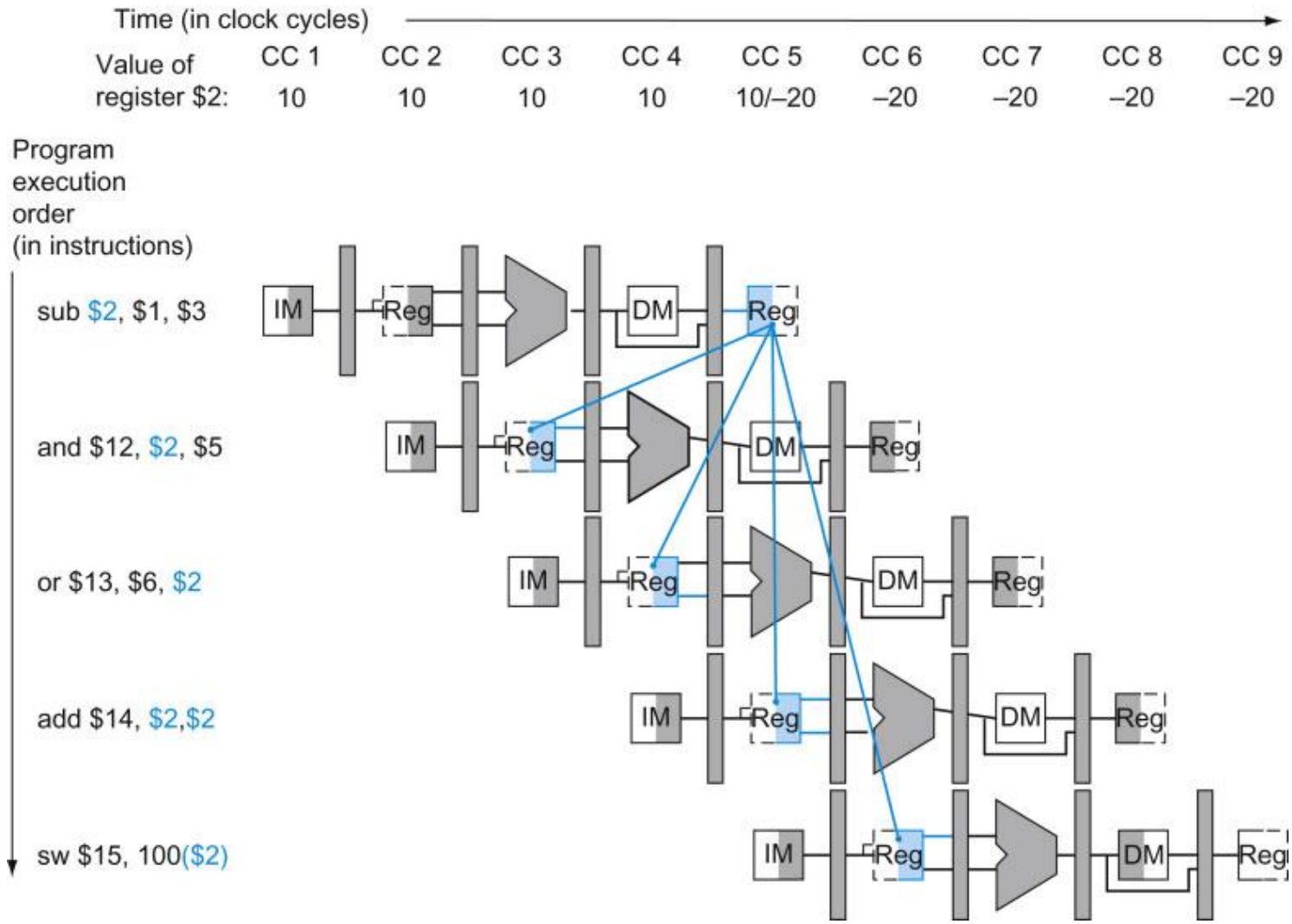
Datenkonflikte bei ALU-Befehlen

- Es sei folgende Befehlssequenz gegeben

```
sub $2,$1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- Konflikte können durch Forwarding aufgelöst werden
- Wie erkennt man, dass Forwarding verwendet werden muss?
  - Registernummern in den unterschiedlichen Pipeline-Registern vergleichen
    - Beispiel: Datenkonflikt durch Forwarding auflösen wenn EX/MEM.RegisterRd = ID/EX.RegisterRs
  - Wird von einer speziellen Einheit (Forwarding-Unit) übernommen
    - Generiert Steuersignale für spezielle Multiplexer

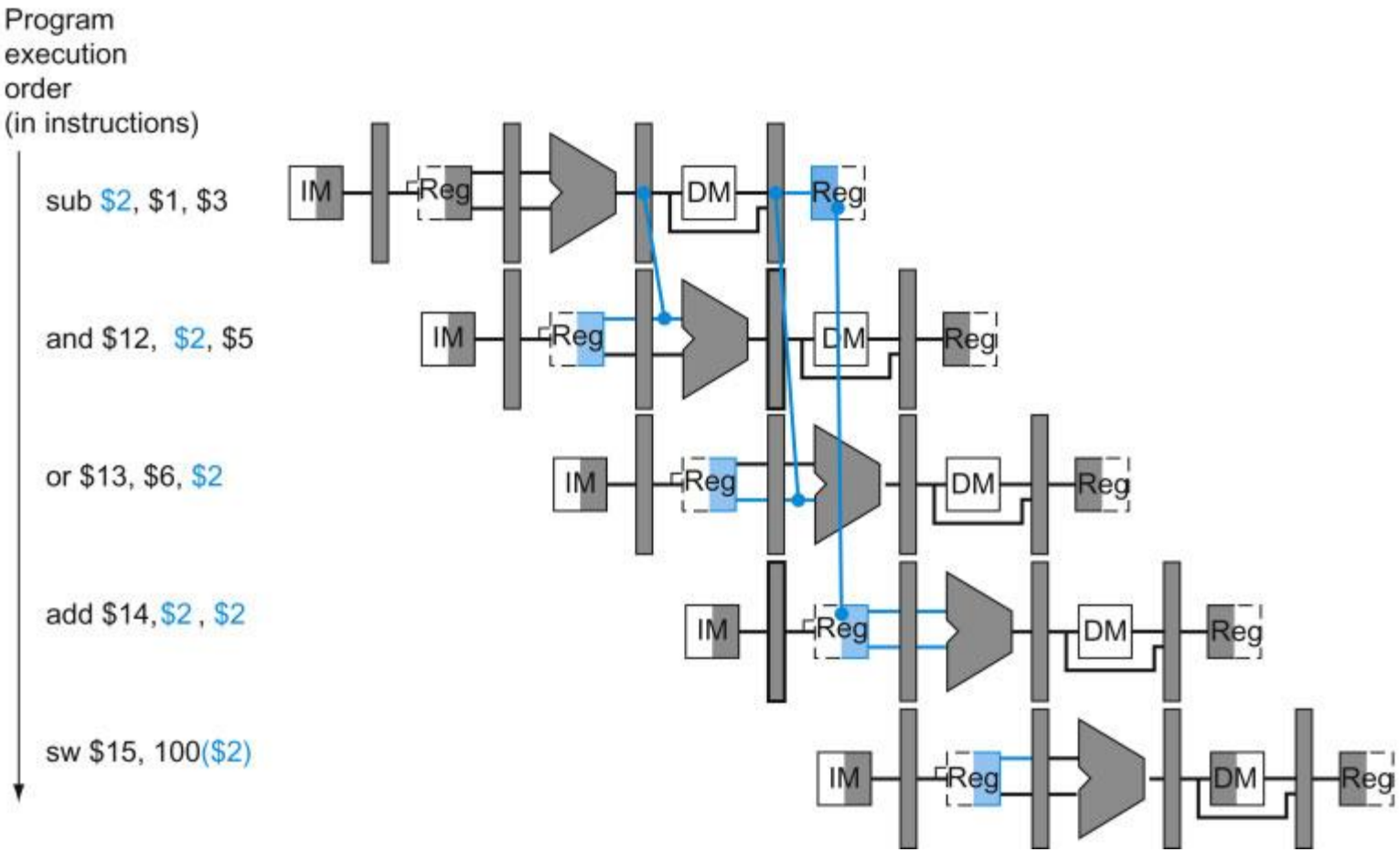
Abhängigkeiten und Forwarding (1)



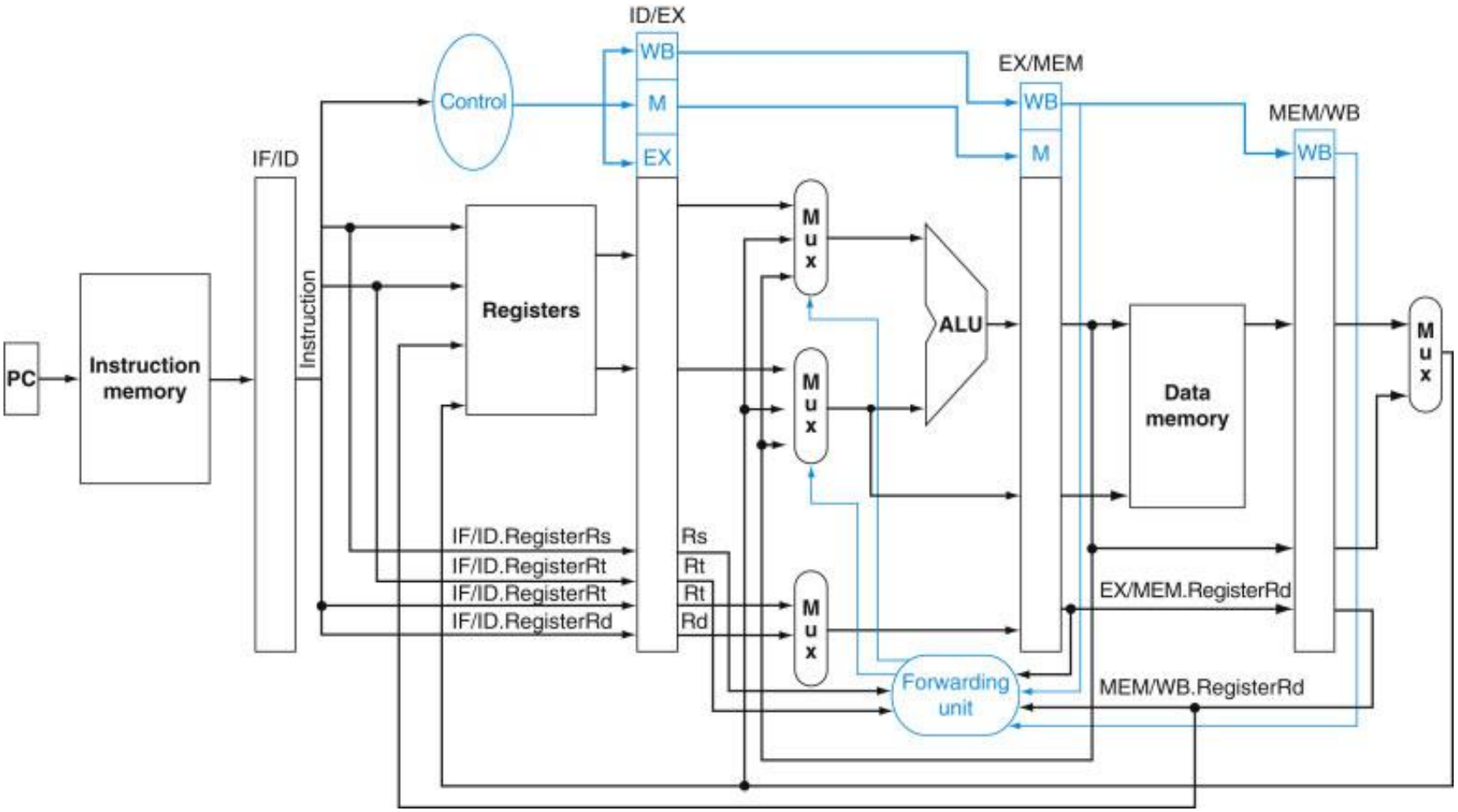
Abhängigkeiten und Forwarding (2)

Time (in clock cycles) →

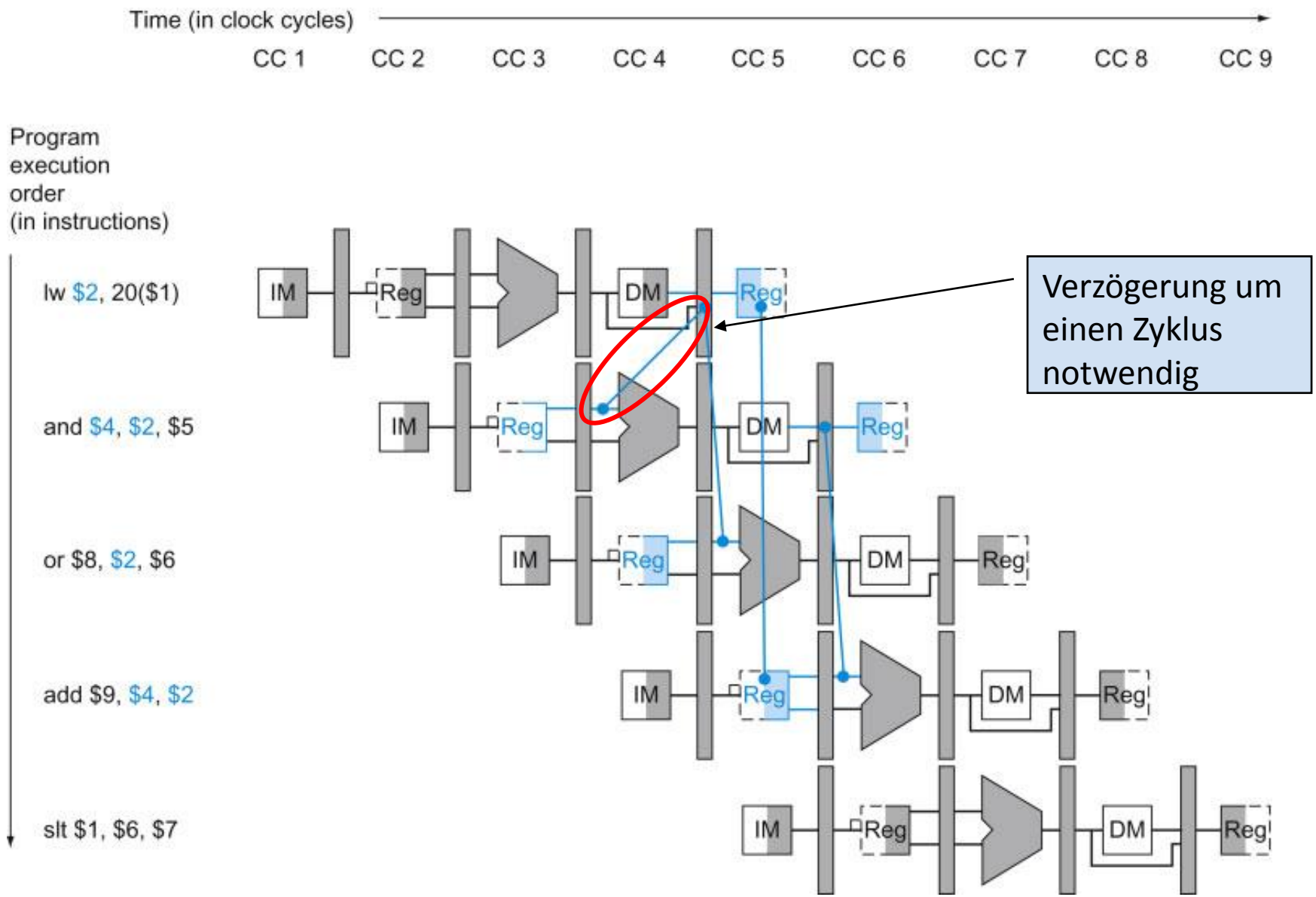
	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X



Datenpfad mit Forwarding (nicht vollständig)



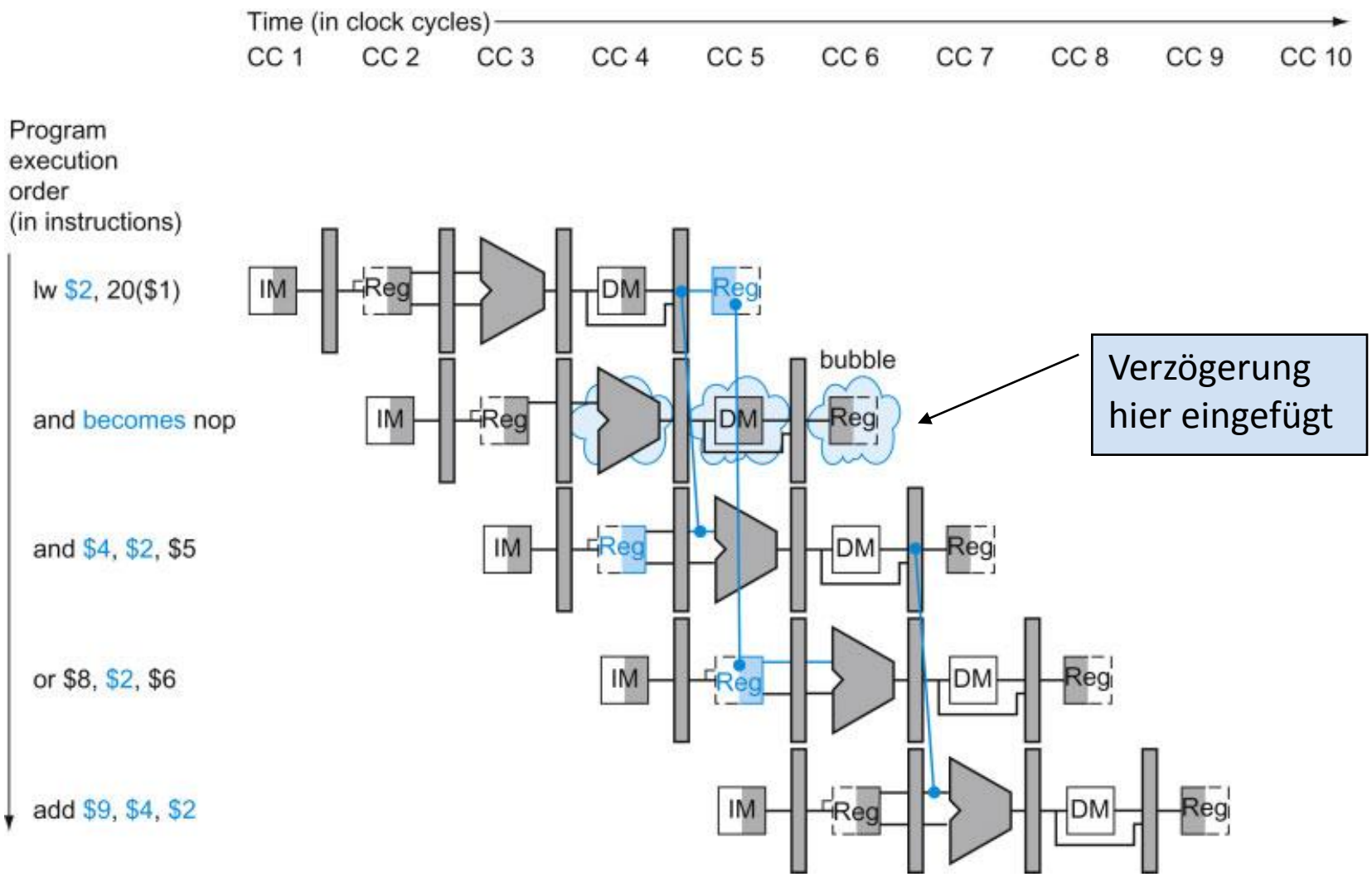
Beispiel (Load-use-Konflikt)



Load-use-Konflikt erkennen

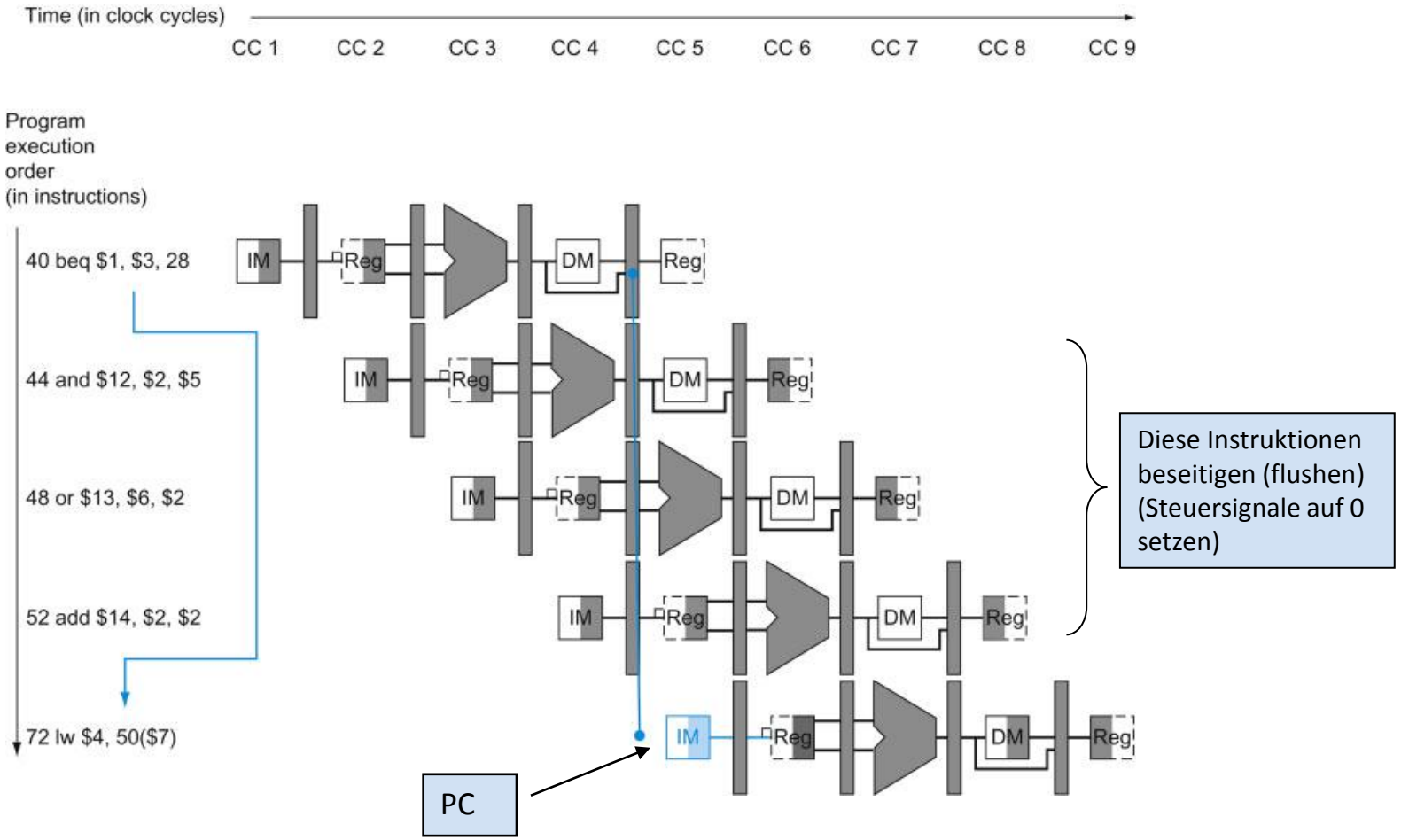
- Überprüfung des Befehls beim Dekodieren in der ID-Stufe
- Wenn Load-use-Konflikt
  - Alle Steuersignale in ID/EX-Register auf 0 setzen
    - EX, MEM und WB führen einen NOP(no operation)-Befehl aus
    - NOP-Befehl führt eine Operation aus, die zu keiner Zustandsänderung führt
  - Befehlszähler und IF/ID-Register dürfen nicht verändert werden
    - Der Befehl wird noch einmal dekodiert
    - Der nachfolgende Befehl wird noch einmal geholt
    - Verzögerung um einen Zyklus erlaubt der MEM-Stufe für den `lw`-Befehl die Daten zu lesen
    - Danach mit Forwarding zur EX-Stufe des nachfolgenden Befehls

Verzögerung/Bubble in der Pipeline



Steuerkonflikte

- Das Ergebnis der Bedingungsauswertung steht erst in der MEM-Stufe zur Verfügung



Behandlung von Steuerkonflikten

- Steuerkonflikte treten seltener auf als Datenkonflikte
- Es gibt keine wirklich wirksame Methode gegen Steuerkonflikte
  - Nicht wie das Forwarding gegen Datenkonflikte
- Behandlung
  - Verzögern (beeinflusst CPI)
  - Entscheidung möglichst früh in der Pipeline treffen (zusätzliche Hardware)
  - Compiler-Unterstützung
  - Vorhersage

ZUSAMMENFASSUNG

Zusammenfassung

- Befehlssatz beeinflusst den Entwurf des Datenpfads und der Steuerung
- Datenpfad und Steuerung beeinflussen den Entwurf des Befehlssatzes
- Pipelining verbessert den Befehlsdurchsatz durch parallele Ausführung
  - Mehr Befehle pro Sekunde werden abgeschlossen
  - Ausführungszeit (Latenz) pro Befehl wird nicht reduziert
- Konflikte: Struktur, Daten, Steuerung

Literatur

- Grundliteratur
  - D. A. Patterson, J. L. Hennessy: **Computer Organization and Design**, 5. Auflage, Morgan Kaufmann, 2013 – Kapitel 4