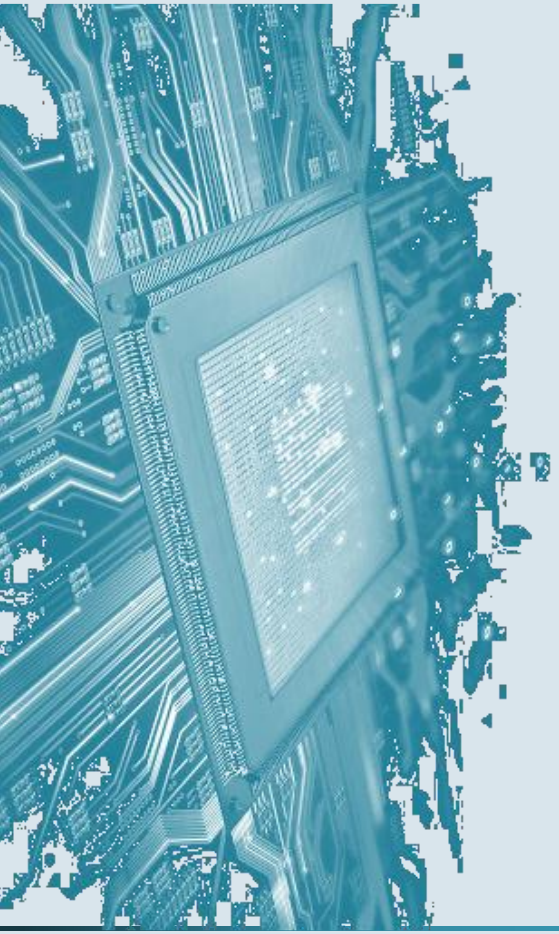


Rechnerarchitektur (AIN 2)

SoSe 2021

Kapitel 3

Der Prozessor- Datenpfad und Steuerwerk

Prof. Dr.-Ing. Michael Blaich
mblaich@htwg-konstanz.de

Kapitel 3 - Lernziele

- In Kapitel 2: Übersicht der MIPS ISA
  - MIPS Instruktionen
  - Format in Maschinensprache
- Kapitel 3: Umsetzung der MIPS ISA in der MIPS CPU
  - Aufbau einer einfachen (idealisierten) MIPS CPU
    - Datenpfad (Datapath)
    - Steuerwerk (Control)
  - Ausführung der MIPS Instruktionen in der CPU
    - Wie fließen die Daten durch den Datenpfad?
    - Was steuert das Steuerwerk?
    - Beispiele: R-Format, Laden/Speichern, Branch und Jump

Kapitel 3 - Gliederung

Kapitel 3: Prozessor – Datenpfad und Steuerwerk

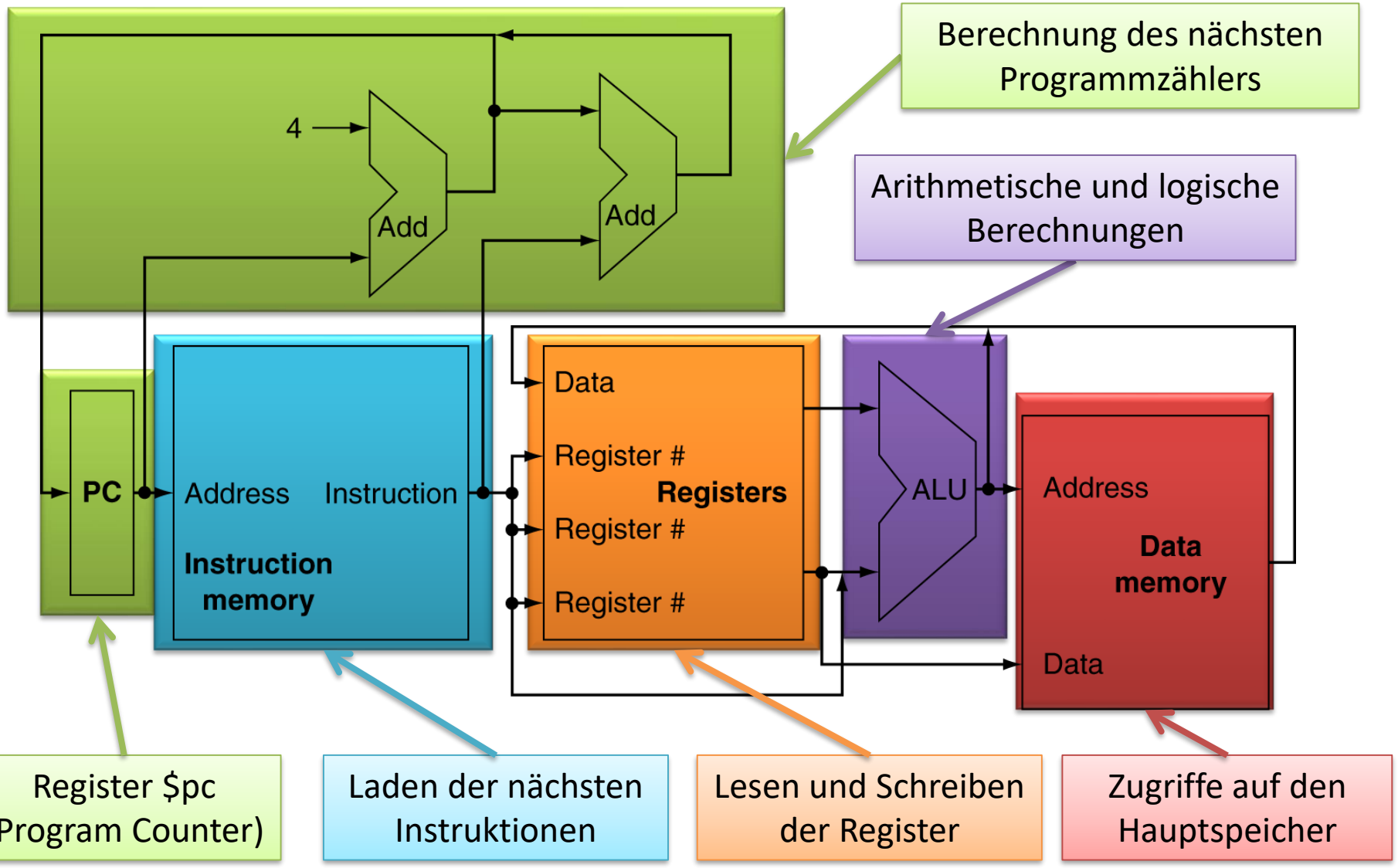
3.1 Aufbau einer CPU

3.2 Datenpfad (Datapath)

3.3 Steuerwerk (Control)

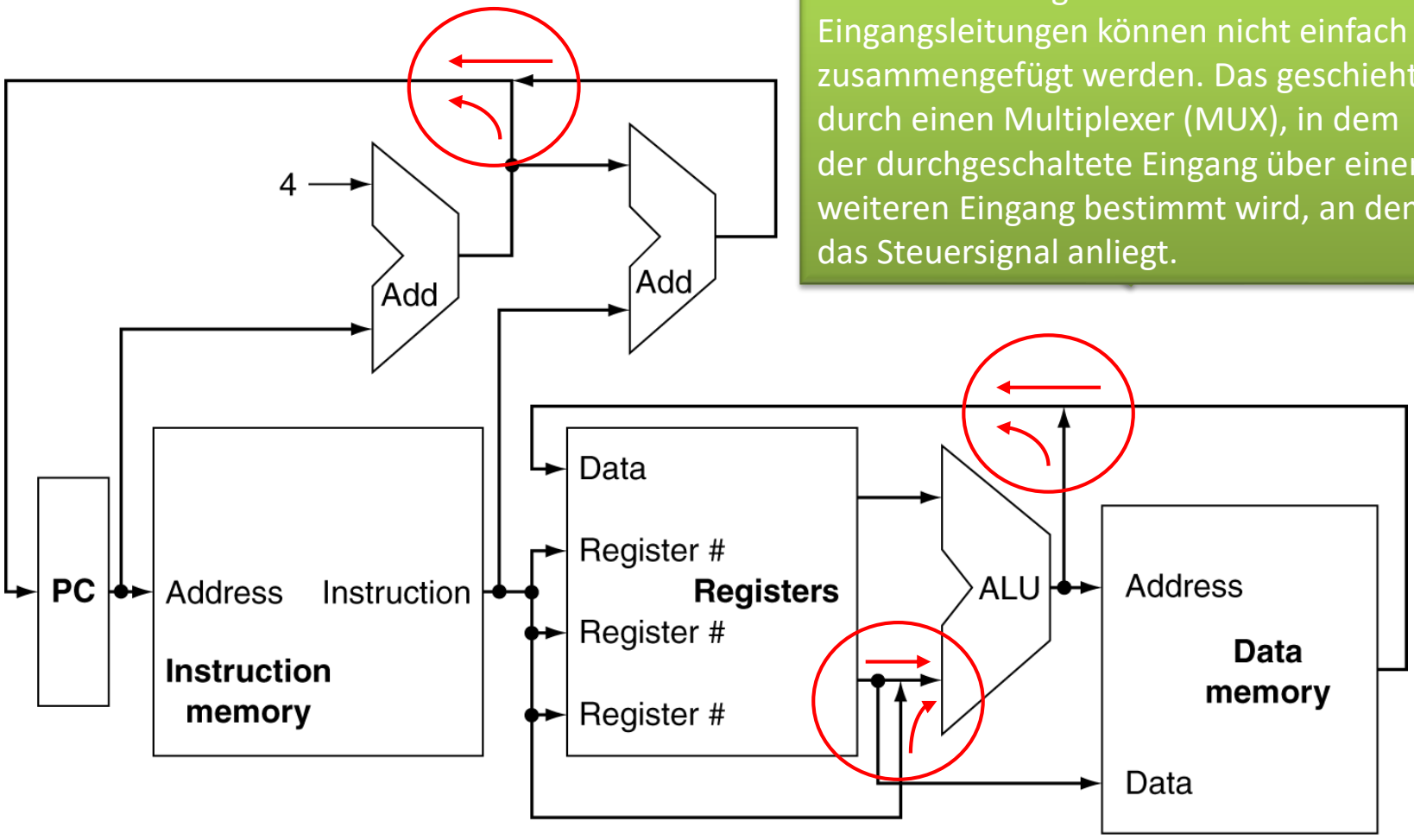
3.4 Zusammenfassung und Bewertung

Vereinfachter Aufbau einer CPU



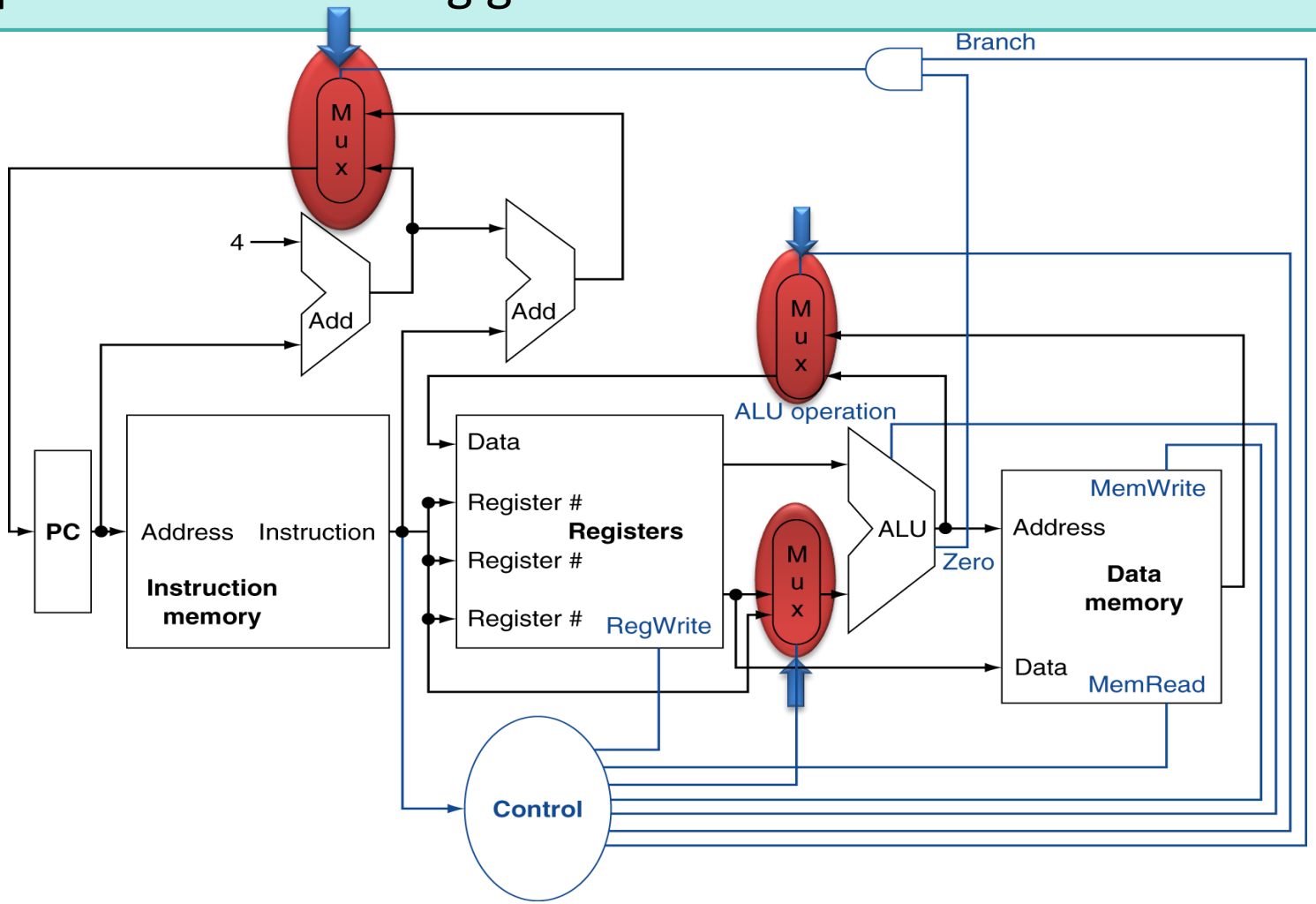
Multiplexer

Elektronische Signale auf zwei Eingangsleitungen können nicht einfach zusammengefügt werden. Das geschieht durch einen Multiplexer (MUX), in dem der durchgeschaltete Eingang über einen weiteren Eingang bestimmt wird, an dem das Steuersignal anliegt.



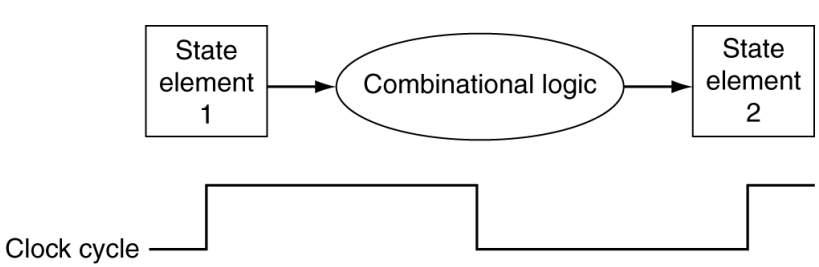
Control

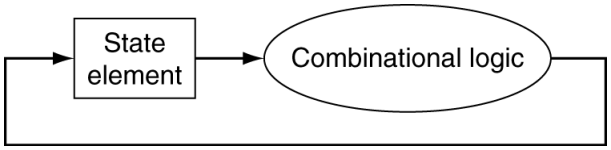
Control (blaue Pfeile) steuert die Operationen der verschiedenen Komponenten in Abhängigkeit der Instruktion



Hardware-Bausteine

- Kombinatorische Elemente (combinational logic)
  - Hardware-Element, das eine logische Funktion berechnet
- Speicherelemente (state element, sequential element)
  - speichert Daten (mehrere Bits)
  - bei Umschalten der Uhr (clock) (steigende Flanke) werden die Werte der Eingangsleitungen gespeichert
  - bis zum nächsten Umschalten der Uhr
    - liegen diese Werte an den Ausgangsleitungen an
    - werden die Eingangsleitungen "blockiert", d.h. es werden keine neuen Werte übernommen

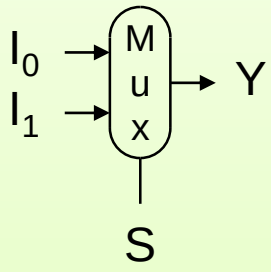




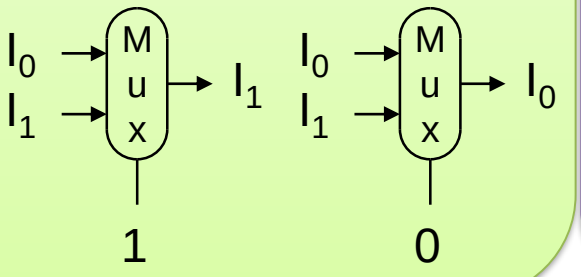
Hardware-Bausteine: Kombinatorische Elemente

Kombinatorische Elemente haben keinen Speicher. Abhängig von den Werten (0/1 oder low/high voltage) der Eingangsleitungen liegt nach einer kurzen Verzögerungen das Ergebnis auf den Ausgangsleitungen an. Die hier gezeigten Eingänge können aus mehreren Einzelleitungen bestehen.

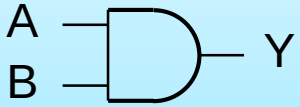
Multiplexer



Wenn S "gesetzt" ist, dann schalte Leitung I_1 durch sonst I_0

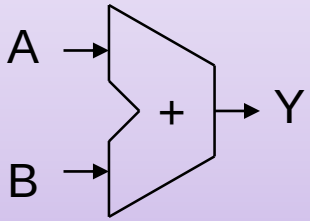


AND-Gatter



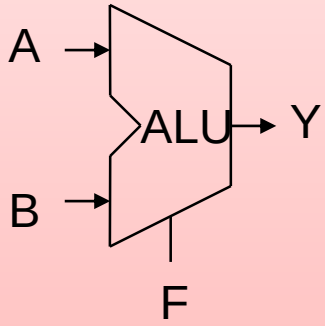
Y=A AND B

ADDER



Y=A + B

ALU

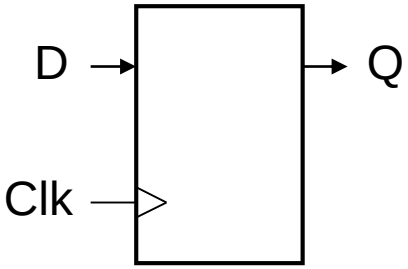


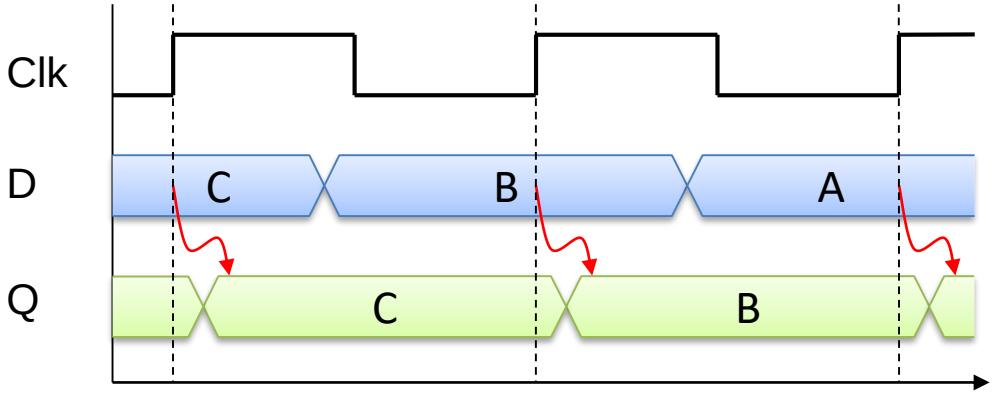
ALU (Arithmetic Logic Unit) kann eine begrenzte Anzahl von Funktionen auf den Operanden A und B ausführen. Die Funktion wird über Eingabe F ausgewählt.

Y=F(A,B)

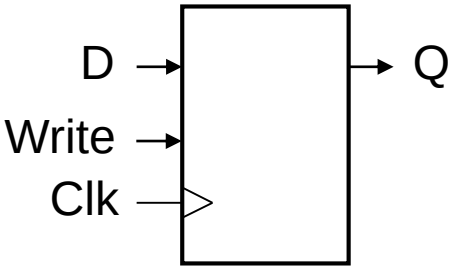
Hardware-Bausteine: Speicher-Elemente

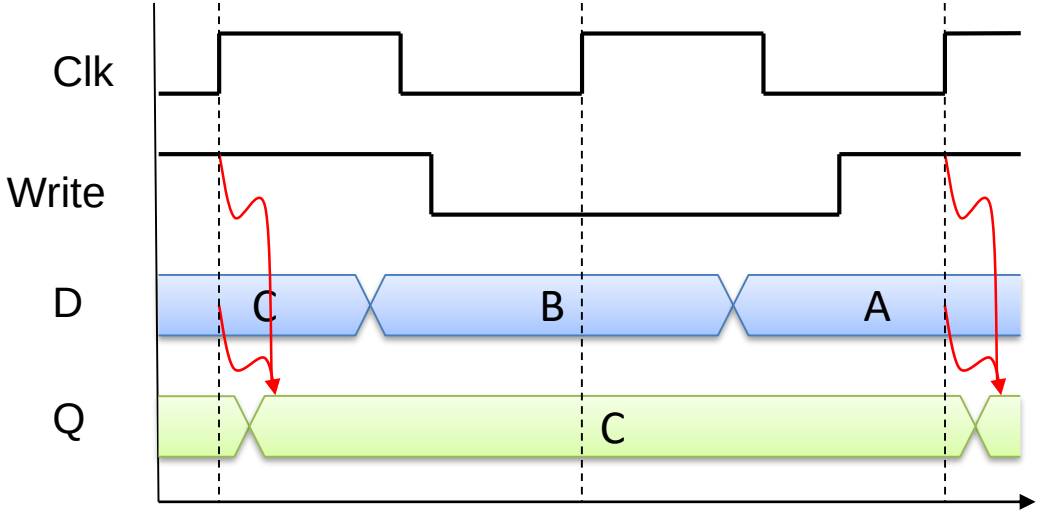
- Speicher-Element ohne Kontrolle des Schreibzugriffs





- Speicher-Element mit Kontrolle des Schreibzugriffs
  - Eingangsleitung wird nur übernommen, wenn "Write" gesetzt ist





Ausführung einer Instruktion

- Instruktion aus dem Speicher laden
  - Program Counter enthält Adresse der Instruktion
- Abhängig von der Instruktion
  - Register auswählen zum Lesen und Schreiben
    - Instruktion enthält die Registernummern
  - Berechnung mit der ALU durchführen
    - arithmetische oder logische Operation
    - Speicheradresse für Datentransferbefehle
    - Sprungadresse für Branches
  - Speicherzugriff bei Datentransferbefehlen
  - Program Counter auf Speicheradresse der nächsten Instruktion setzen

Datapath und Control

- Datapath
  - Hardware-Elemente in der CPU, die Daten verarbeiten und adressieren
    - Register, ALU, MUX, Speicher

- Control
  - Hardware-Elemente, die die Operation der Hardware-Elemente des Datenpfades abhängig von der Instruktion steuern

Wir entwickeln Schritt für Schritt den Aufbau der MIPS CPU: erst Datapath dann Control

Kapitel 3 - Gliederung

Kapitel 3: Prozessor – Datenpfad und Steuerwerk

3.1 Aufbau einer CPU

3.2 Datenpfad (Datapath)

3.2.1 Komponenten

3.2.2 Instruktionen für Arithmetik und Datentransfer

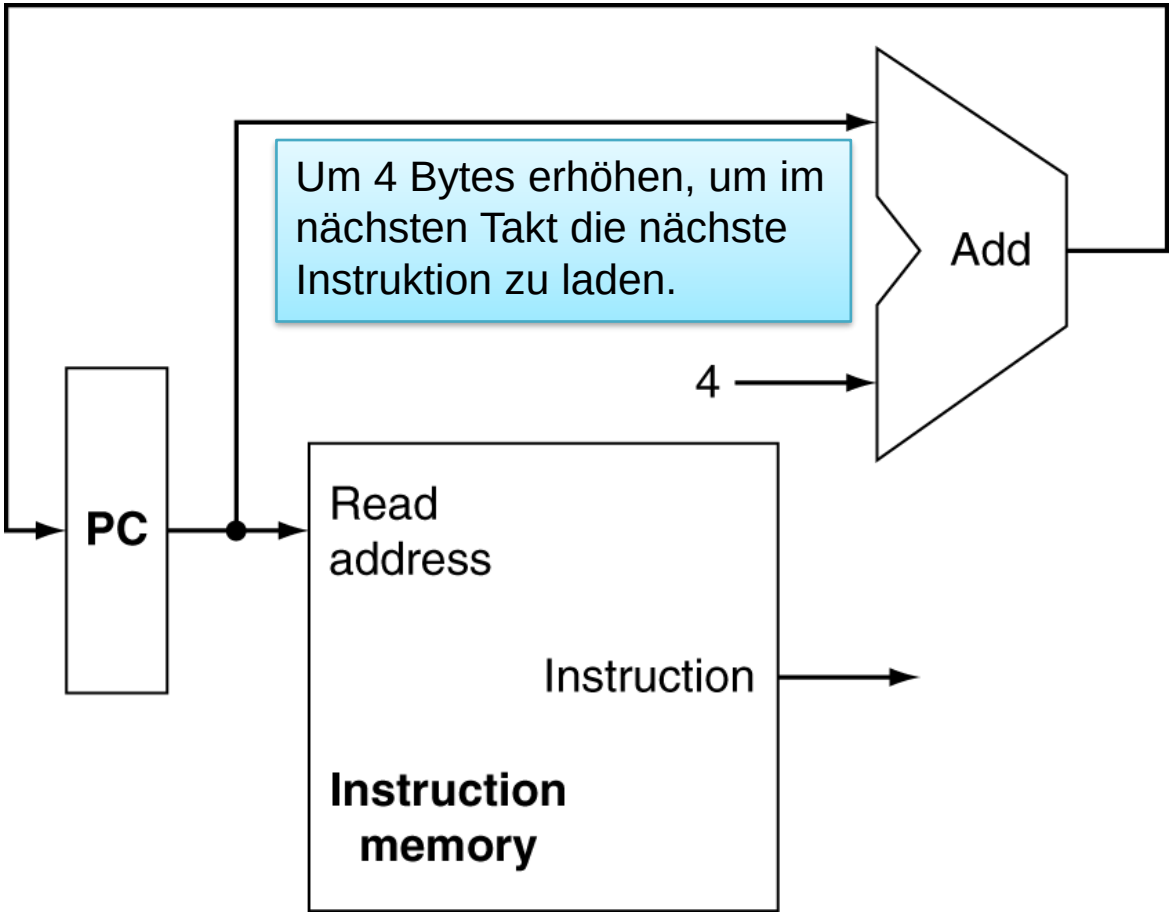
3.2.3 Branch-Instruktionen

3.3 Steuerwerk (Control)

3.4 Zusammenfassung und Bewertung

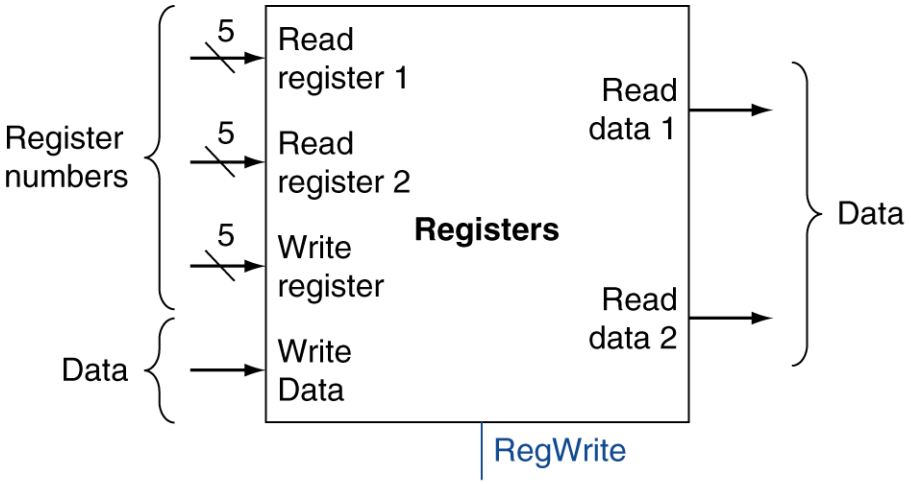
Instruktion laden und PC erhöhen

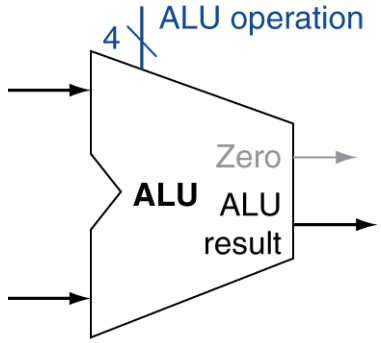
PC: 32-bit Register, das den Program Counter enthält.
Ausgang: Adresse der Instruktion für den aktuellen Takt.
Eingang: Adresse der Instruktion für den nächsten Takt.



Instruktionsspeicher: Aus dem Instruktionsspeicher die Instruktion laden, die an der durch den Eingang “Read Address” spezifizierten Adresse steht. Die geladene Instruktion steht am Ausgang “Instruction”. Wir gehen zunächst davon aus, dass das Laden der Instruktion aus dem Speicher einen Takt dauert.

Register und ALU

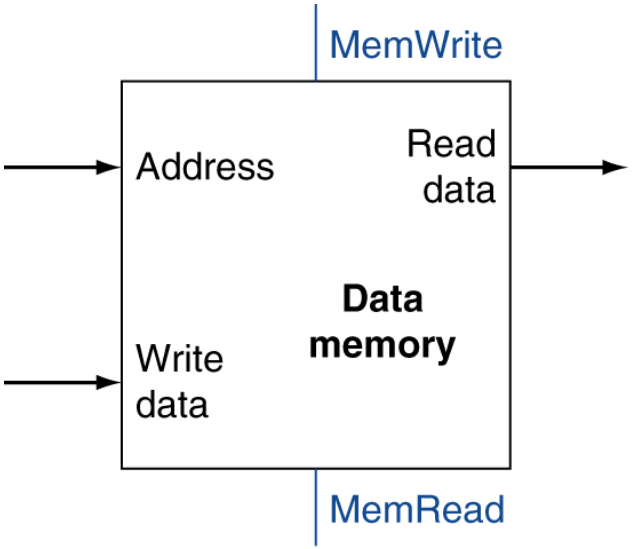


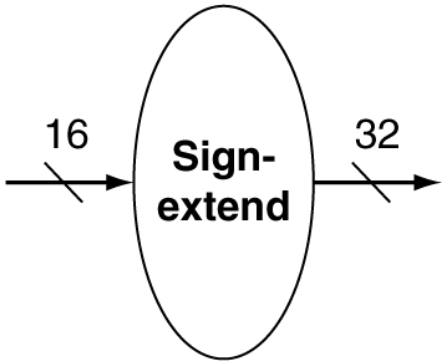


- Der **Registersatz** enthält Speicherplätze für die 32 MIPS Register.
- Über die Eingänge **Register Numbers** werden zwei Register zum Lesen und ein Register zum Schreiben ausgewählt.
- An den Ausgängen **Read Data 1** und **Read Data 2** liegen nach dem Schalten der Uhr die Inhalte der beiden zum Lesen ausgewählten Register an.
- Am Eingang **"Data"** liegen die Daten an, die in das zum Schreiben ausgewählte Register geladen werden sollen.
- Über das Control-Signal **"RegWrite"** bestimmt die Control-Einheit, ob der Eingang **Data** in das durch den Eingang **Write-Register** spezifizierte Register geschrieben werden soll.

- Die 32-Bit **ALU** enthält zwei Eingänge mit je 32-Bit und einen Ausgang mit 32 Bit.
- An den beiden Eingängen liegen die Operanden an.
  - Der „obere“ Eingang enthält immer den **Registeroperanden** aus dem Register rs (Read Data 1).
  - Der „untere Eingang“ enthält je nach Instruktion entweder den **Registeroperanden** aus dem Register rt (Read Data 2) oder die in der Instruktion enthaltene **Konstante**.
- Der Ausgang **ALU result** enthält das Ergebnis der ALU Berechnung.
- Der Ausgang **zero** wird gesetzt, falls das Ergebnis der Berechnung in der ALU Null ist.
- Über das 4-Bit Control-Signal **ALU operation** bestimmt die Control-Einheit die **Funktion**, die von der ALU berechnet wird.

Speicherzugriffe: Laden und Speichern



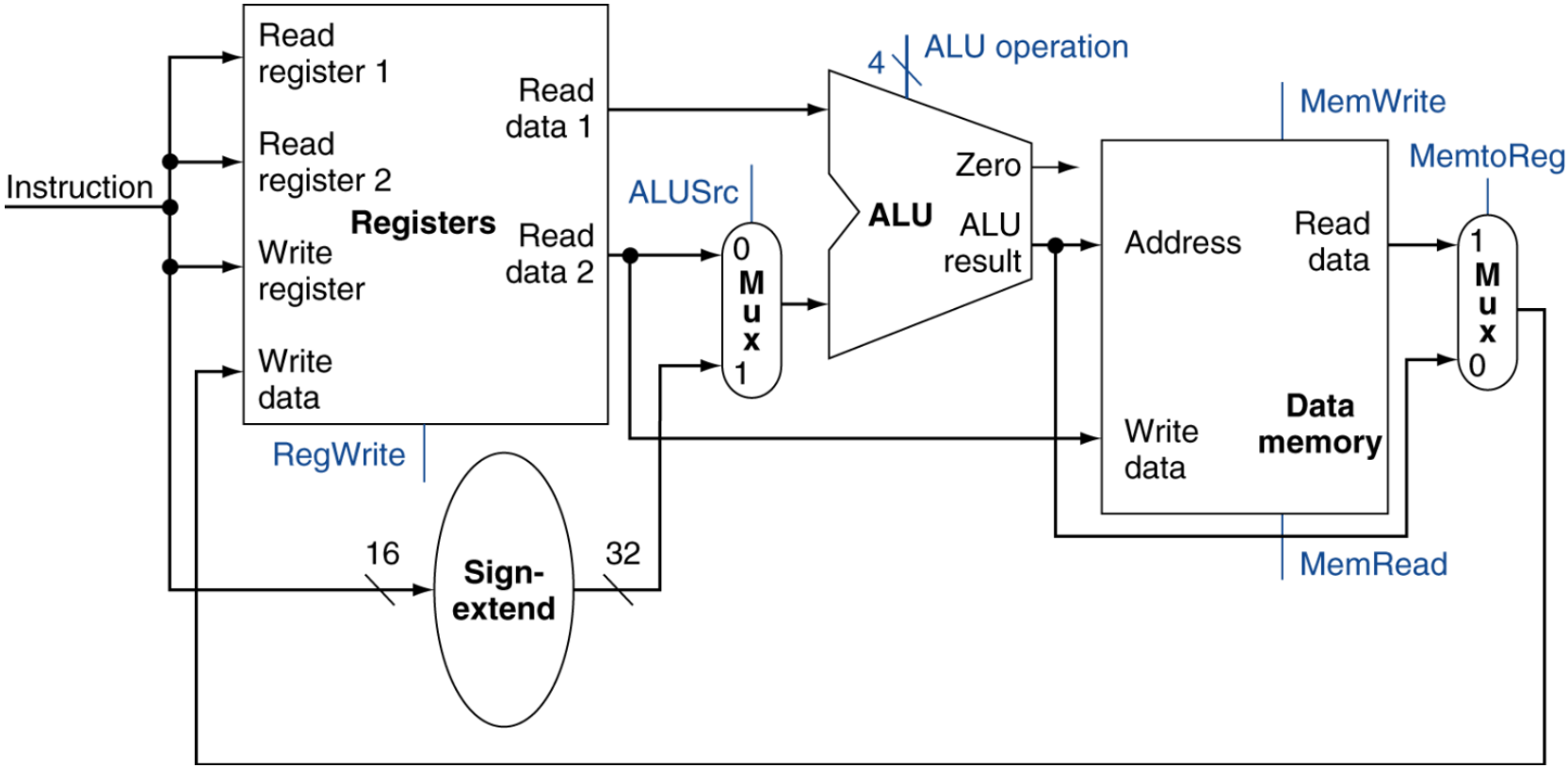


- Der Speicherblock dient zum Zugriff auf Daten im Hauptspeicher. In der idealisierten CPU nehmen wir an, dass die Daten in der CPU liegen und der Zugriff in einem Takt erfolgen kann.
- Ist der Control-Eingang **MemWrite** gesetzt, so werden die Daten, die am Eingang **Write data** anliegen, an die durch den Eingang **Address** bestimmte Speicheradresse geschrieben.
- Ist der Control-Eingang **MemRead** gesetzt, so werden die Daten an der durch den Eingang **Address** bestimmten Speicheradresse aus dem Speicher gelesen und in den Ausgang **Read data** geschrieben.

- Das **Sign-Extend-Element** wird benötigt, um einen vorzeichen-behafteten 16-Bit Wert in ein 32-Bit Wort zu transferieren.
- Beispiele sind 16-Bit **Adressen** und **Konstanten** in der Instruktion, die auf 32 Bit erweitert werden müssen, damit die ALU damit rechnen kann.
- Das **Vorzeichen** bleibt bei der Erweiterung auf 32 Bit erhalten. Das höchst-wertige Bit des 16-Bit-Eingangs wird in die höchst-wertigen 16 Bit des 32-Bit Ausgangs geschrieben.

Zusammengefügt: Arithmetik und Datentransfer

- Teil des Datenpfads zur Ausführung von Instruktionen im **R-Format** (arithmetische/logische Operationen) und Instruktionen im **I-Format** (arithmetische/logische Operationen mit Konstanten, Datentransfer)
- Die Control-Unit setzt die **Steuerbefehle** so, dass die aktuelle Instruktion ausgeführt wird. Dadurch kann die **gleiche Hardware für unterschiedliche Instruktionen** verwendet werden.



Kapitel 3 - Gliederung

Kapitel 3: Prozessor – Datenpfad und Steuerwerk

3.1 Aufbau einer CPU

3.2 Datenpfad (Datapath)

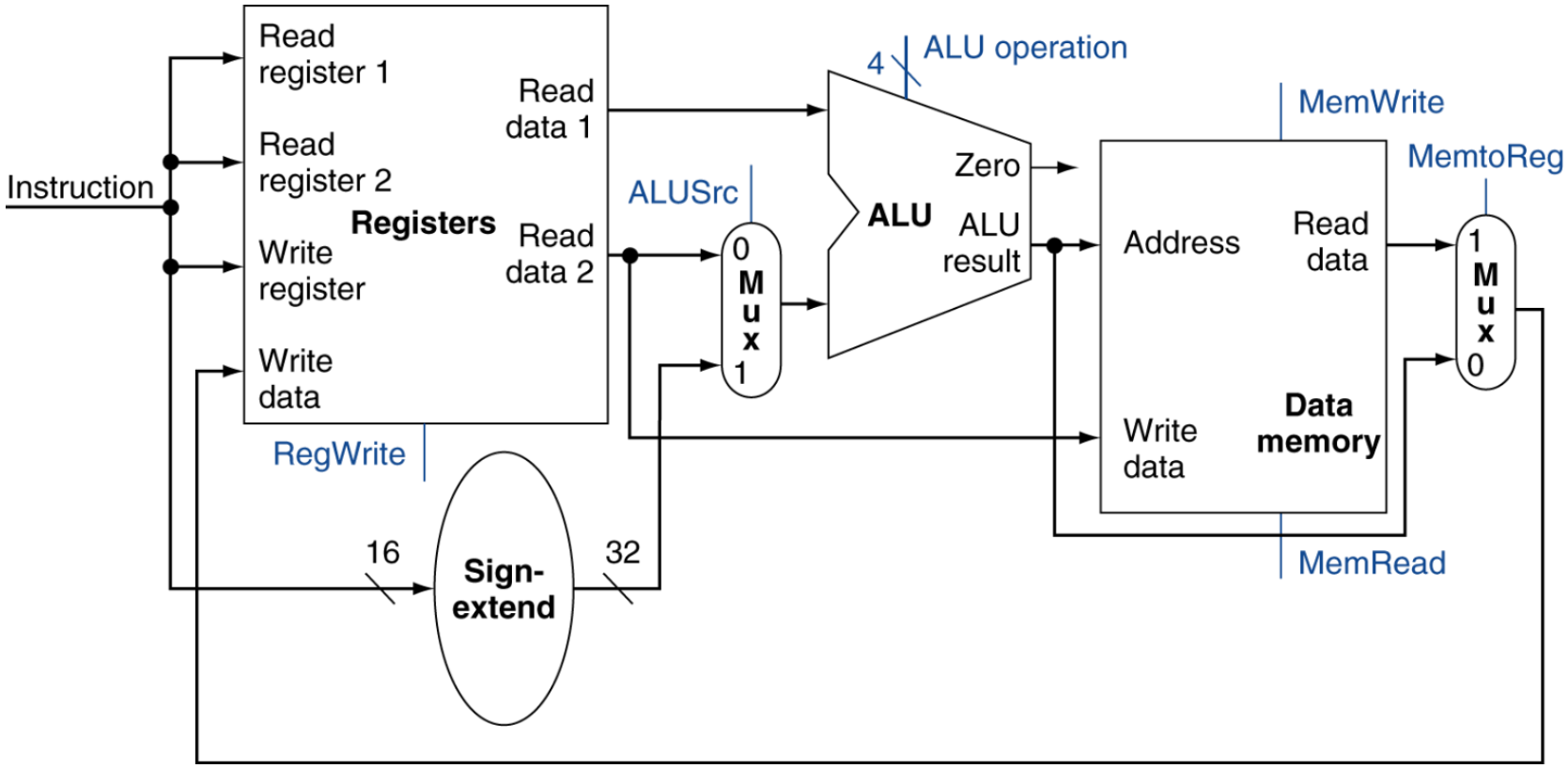
- 3.2.1 Komponenten
- **3.2.2 Instruktionen für Arithmetik und Datentransfer**
- 3.2.3 Branch-Instruktionen

3.3 Steuerwerk (Control)

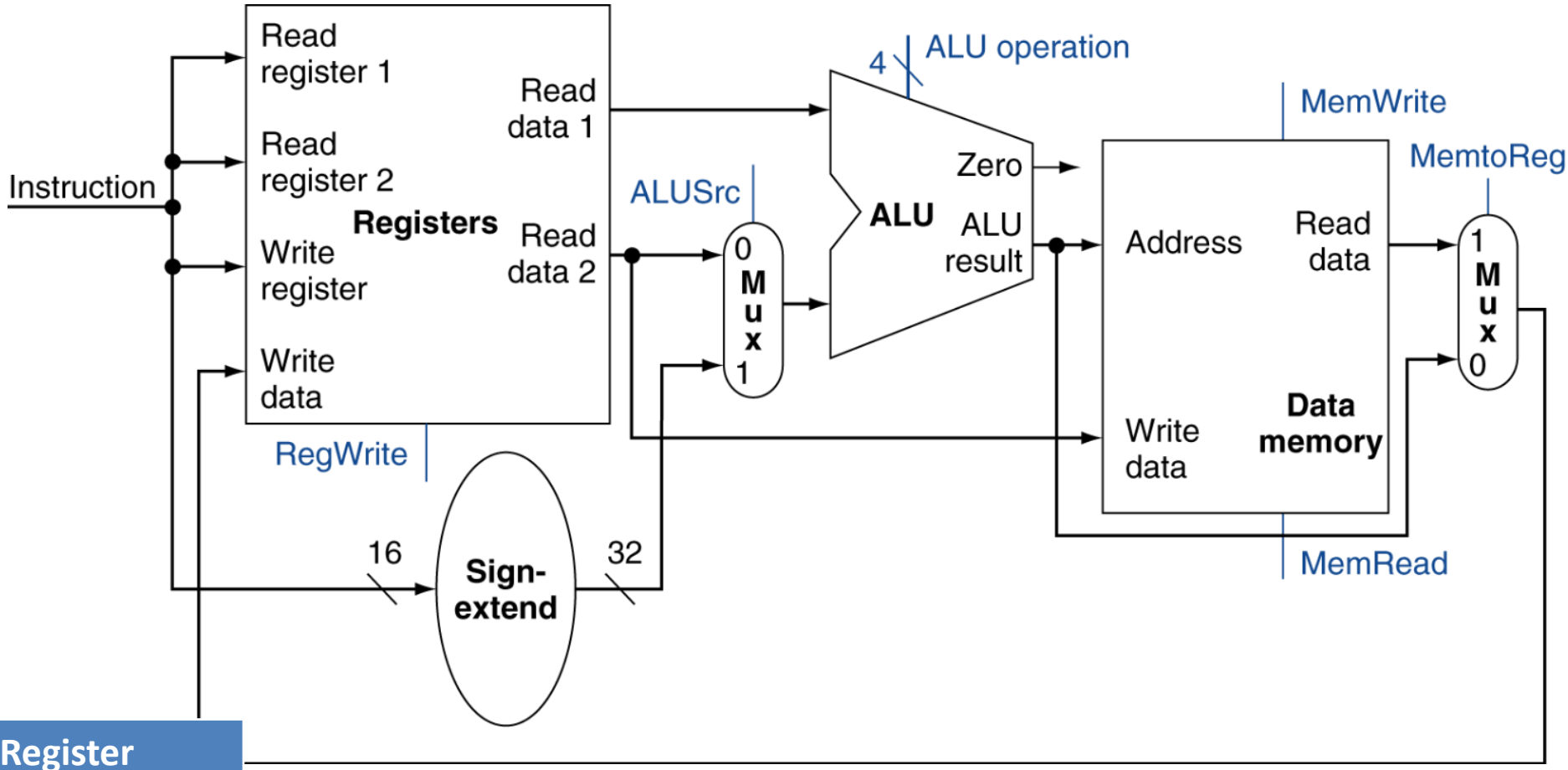
3.4 Zusammenfassung und Bewertung

R-Format: ADD $t0, $t1, $t2

Als erstes Beispiel betrachten wir, wie die Steuersignale für die Instruktion ADD gesetzt werden und wie dadurch die Instruktion ausgeführt wird.



R-Format: ADD $t0, $t1, $t2

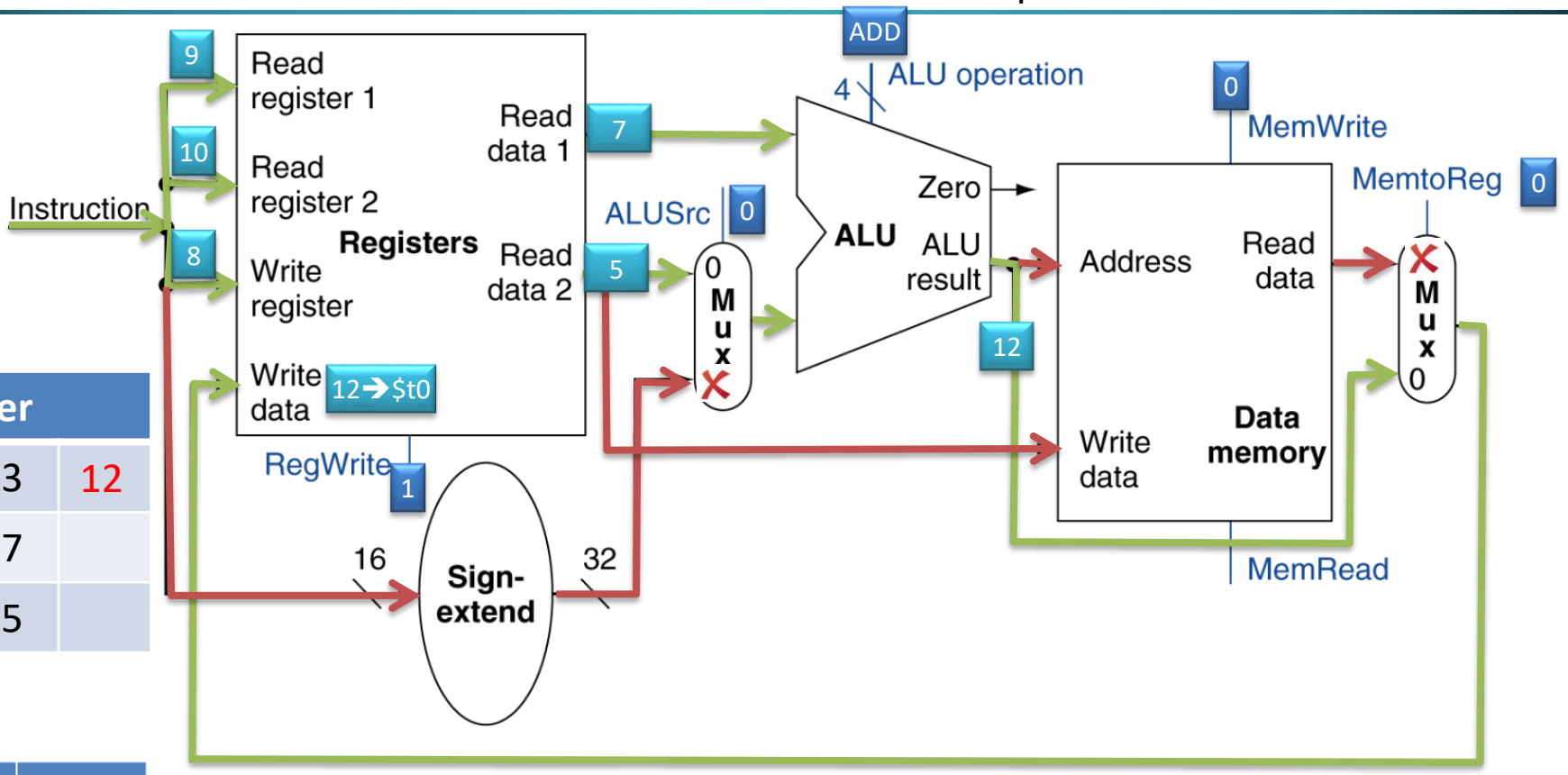


Register		
$t0	3	
$t1	7	
$t2	5	

	opcode	rs	rt	rd	shamt	funct
add $t0, $t1, $t2 ➔	000000	01001	01010	01000	00000	100000

R-Format: ADD $t0, $t1, $t2

0	9	10	8	0	32
opcode	rs	rt	rd	shamt	funct



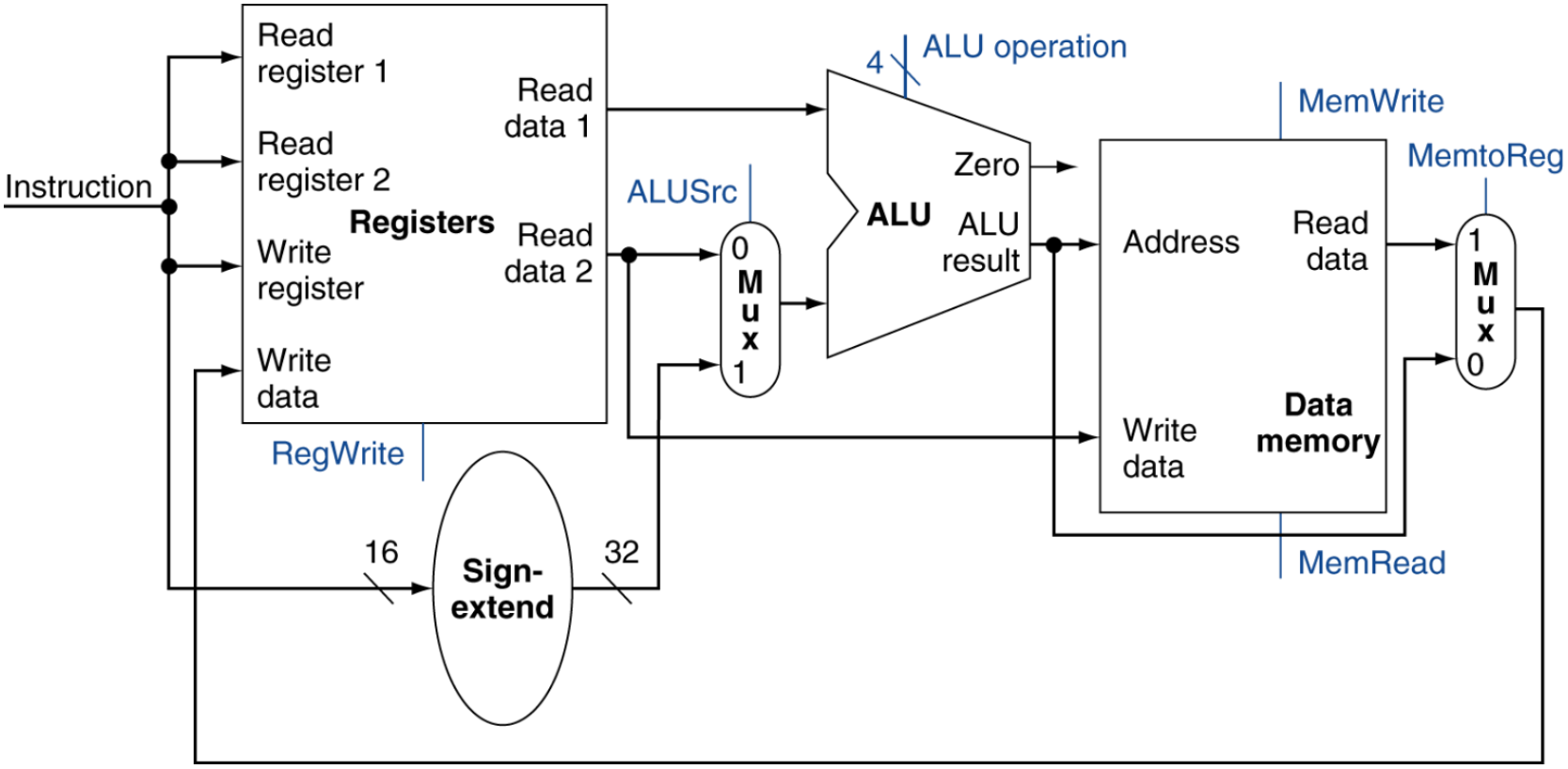
Register		
$t0	3	12
$t1	7	
$t2	5	

Name	Nr
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12

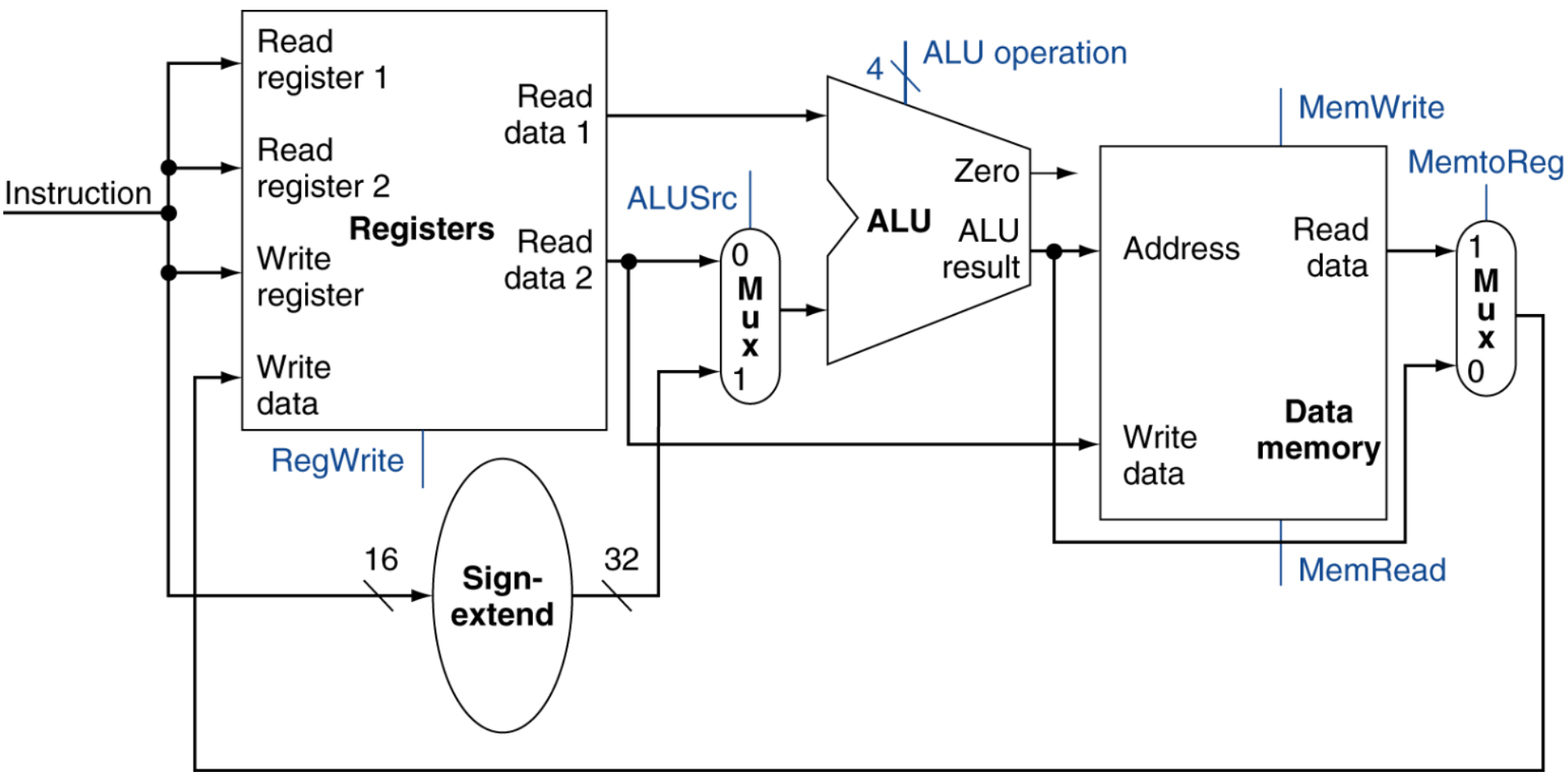
- Die Registerfelder der Instruktion bilden die Eingänge der Registernummern des Registersatzes.
  - rs=$t1 (9), rt=$t2 (10), rd=$t0 (8)
- Der Inhalt der Register $t1 und $t2 steht in den Ausgängen des Registersatzes.
- Das Steuersignal ALUSrc steht auf "0", zweiter Operand der ALU ist Read data 2 Ausgang des Registersatzes, im Beispiel also Inhalt von Register $t2.
- ALU führt dem Steuersignal entsprechend eine Berechnung durch, im Beispiel eine Addition mit Ergebnis $t1+$t2.
- Das Ergebnis der ALU geht in den Multiplexer. Das Steuersignal MemtoReg steht auf "0" (false) und das Ergebnis der ALU wird durchgeschaltet.
- Das Steuersignal RegWrite des Registersatzes steht auf "1" (true) und das Ergebnis der ALU-Operation wird in das zum Schreiben gewählte Register geschrieben, im Beispiel $t0=$t1+$t2.
- Im Datenblock wird das Steuersignal "MemWrite" auf "0" (false) gesetzt, um ungewollte Speicherzugriffe zu verhindern.
  Hinweis: Das Ergebnis der ALU-Berechnung und der Inhalt des zweiten Registers liegen auch bei einer ADD –Instruktion an den Eingängen des Speicherblocks an. Nur die Steuersignale verhindern, dass eine Operation durchgeführt wird.

I-Format: lw $t0, 4($t1)

Das zweite Beispiel ist die Instruktion „load word“ im I-Format. Hier wird die ALU genutzt, um aus Basisadresse und Offset die Speicheradresse zu berechnen, von der das Wort gelesen wir.



I-Format: lw $t0, 4($t1)



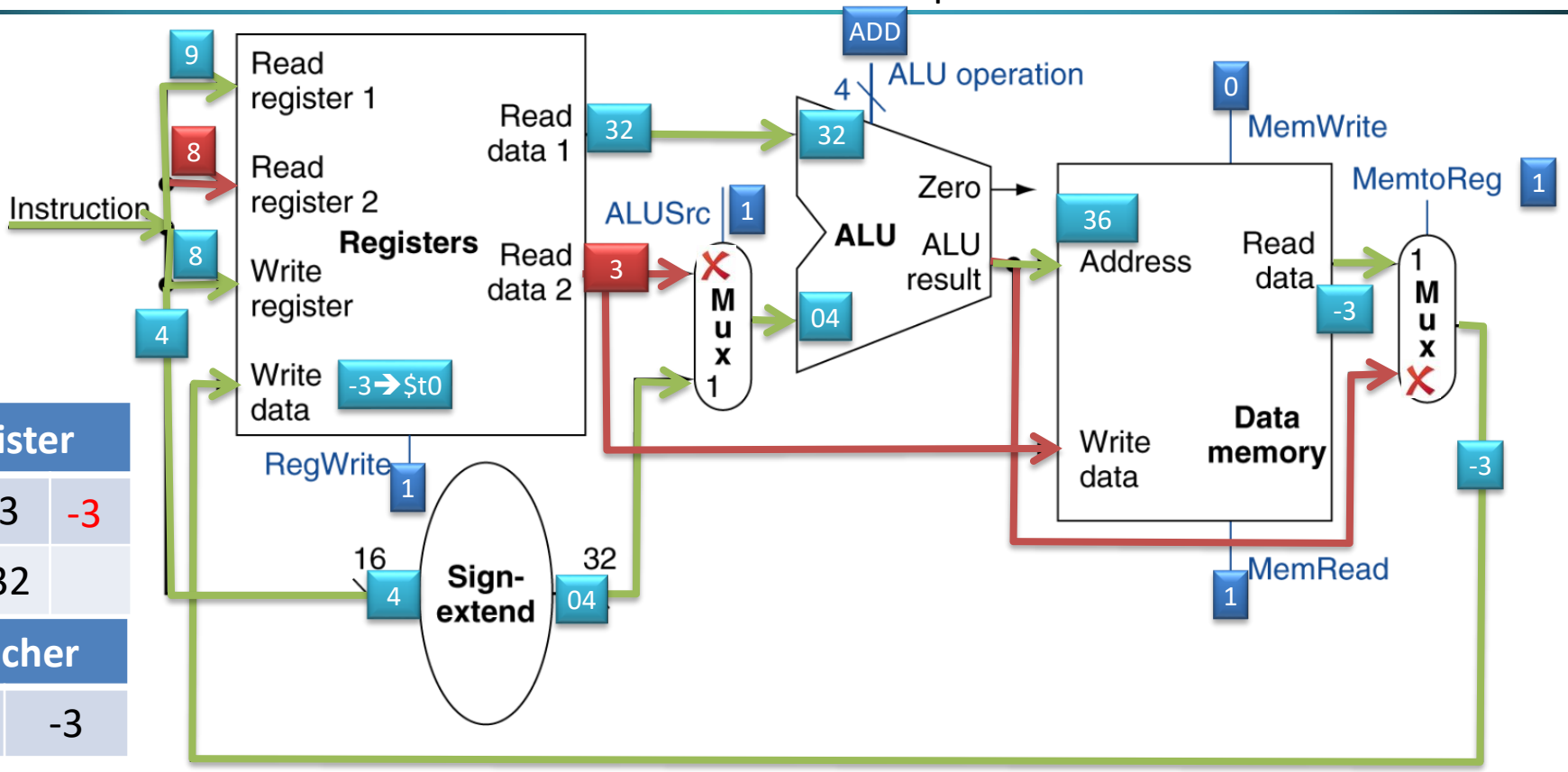
Register		
$t0	3	
$t1	32	

Speicher	
36	-3

lw $t0, 4($t1) ➔

opcode	rs	rt	const
100011	01001	01000	0000 0000 0000 0100

I-Format: lw $t0, 4($t1)



Register		
$t0	3	-3
$t1	32	

Speicher	
36	-3

- Das Registerfeld "rs" der Instruktion wird in den Eingang "Read register 1" des Registersatzes geschrieben und das Registerfeld "rt" in den Eingang "Write register"
- Der Inhalt des Register $t1 steht im "Read data 1" Ausgang des Registersatzes.
- Die Konstante (Adress-Offset) aus der Instruktion (4) wird von 16 Bit auf 32 Bit erweitert.
- Das Steuersignal ALUSrc steht auf "1", zweiter Operand der ALU ist die Konstante aus der Insturktion, im Beispiel also 4.
- ALU führt dem Steuersignal entsprechend eine Berechnung durch, im Beispiel eine Addition mit Ergebnis $t1+4.

- Das Ergebnis der ALU geht in den den "Address"-Eingang des Datenblocks.
- Im Datenblock wird das Steuersignale "MemRead" auf "1" (true) und "MemWrite" auf "0" (false) gesetzt. Am Ausgang "Read data" des Speicherblocks steht der Inhalt von Speicheradresse $t1+4.
- Das Steuersignal MemtoReg steht auf "1" (true) und der Ausgang des Speicherblocks wird zum "Write data" Eingang des Registersatzes durchgeschaltet.
- Das Steuersignal "RegWrite" des Registersatzes steht auf "1" (true) und das der Speicherinhalt wird in das zum Schreiben gewählte Register geschrieben, im Beispiel $t0=Mem($t1+4)

Kapitel 3 - Gliederung

Kapitel 3: Prozessor – Datenpfad und Steuerwerk

3.1 Aufbau einer CPU

3.2 Datenpfad (Datapath)

- 3.2.1 Komponenten
- 3.2.2 Instruktionen für Arithmetik und Datentransfer
- **3.2.3 Branch-Instruktionen**

3.3 Steuerwerk (Control)

3.4 Zusammenfassung und Bewertung

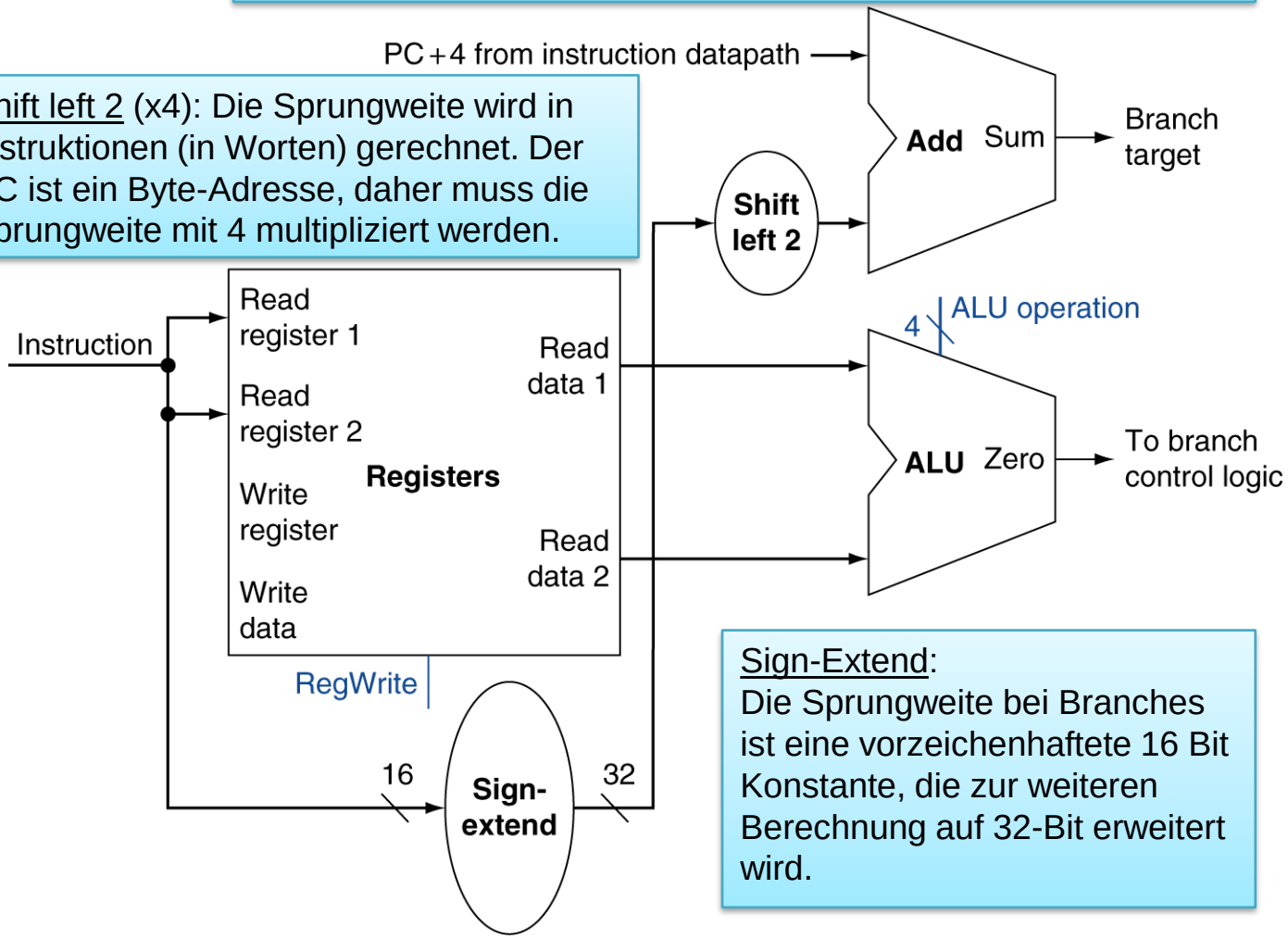
Branch-Instruktionen (Bedingte Sprünge)

- Vorgehensweise:
  - Operanden aus Register einlesen
  - Operanden vergleichen
    - in der ALU subtrahieren
    - "zero" Ausgang prüfen
  - Zieladresse bestimmen
    - 32 Bit Sprungweite (Label) in Bytes
      - Sprungweite von 16 Bit auf 32 Bit erweitern
      - Sprungweite x 4 (von Word (Instruktionen) auf Byte umrechnen)
    - zum Programm Counter (der nächsten Zeile) addieren
      - PC=PC+Sprungweite
      - PC=PC+4 wird generell berechnet

Branches

Add: Die Sprungweite (jetzt als 32-Bit Wert in Bytes) wird zu PC+4 addiert. Wir erinnern uns, dass die Sprungweite immer ab der nächsten Instruktion gerechnet wurde. Den Grund sehen wir hier: Das ist in Hardware einfacher zu realisieren.

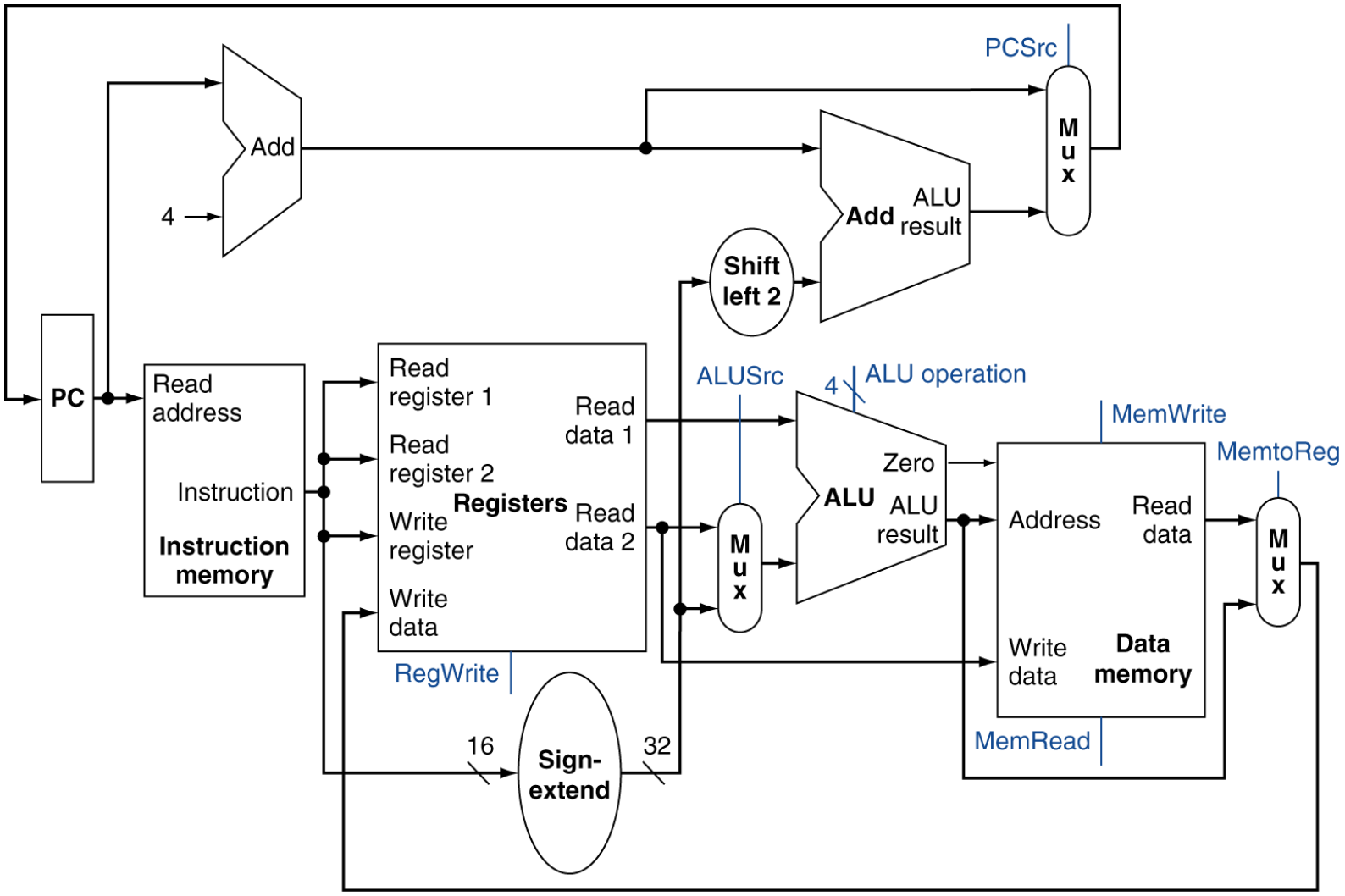
Shift left 2 (x4): Die Sprungweite wird in Instruktionen (in Worten) gerechnet. Der PC ist ein Byte-Adresse, daher muss die Sprungweite mit 4 multipliziert werden.



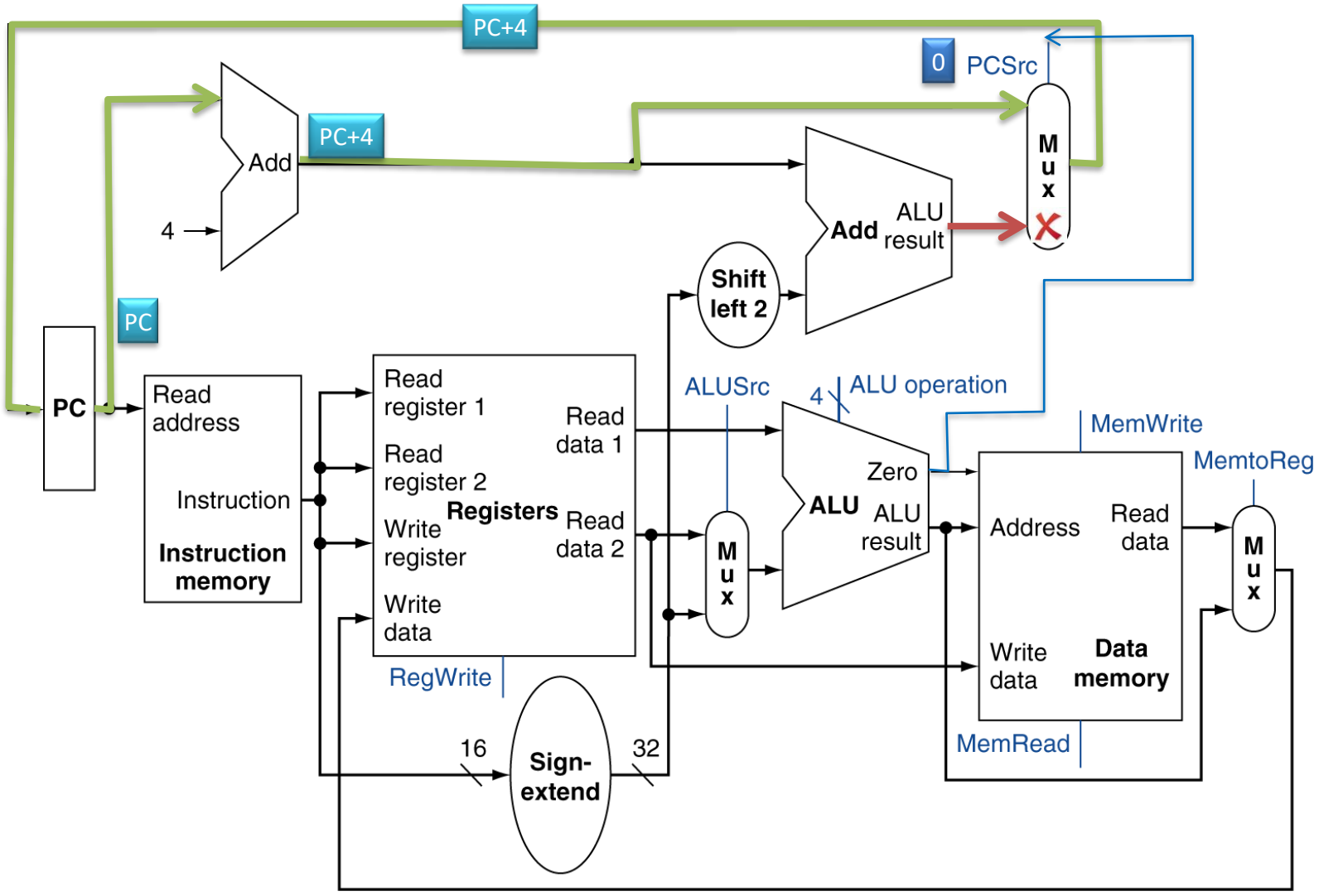
ALU:
- Die ALU führt eine Subtraktion der beiden Operanden durch. Wenn das Ergebnis 0 ist (die beiden Operanden sind gleich), liegt am Ausgang „zero“ eine „1“ an.
- Abhängig von Sprungbefehl (beq/bne) und „zero“ wird die berechnete Sprungadresse oder PC+4 in den PC geschrieben.

Sign-Extend:
Die Sprungweite bei Branches ist eine vorzeichenhaftete 16 Bit Konstante, die zur weiteren Berechnung auf 32-Bit erweitert wird.

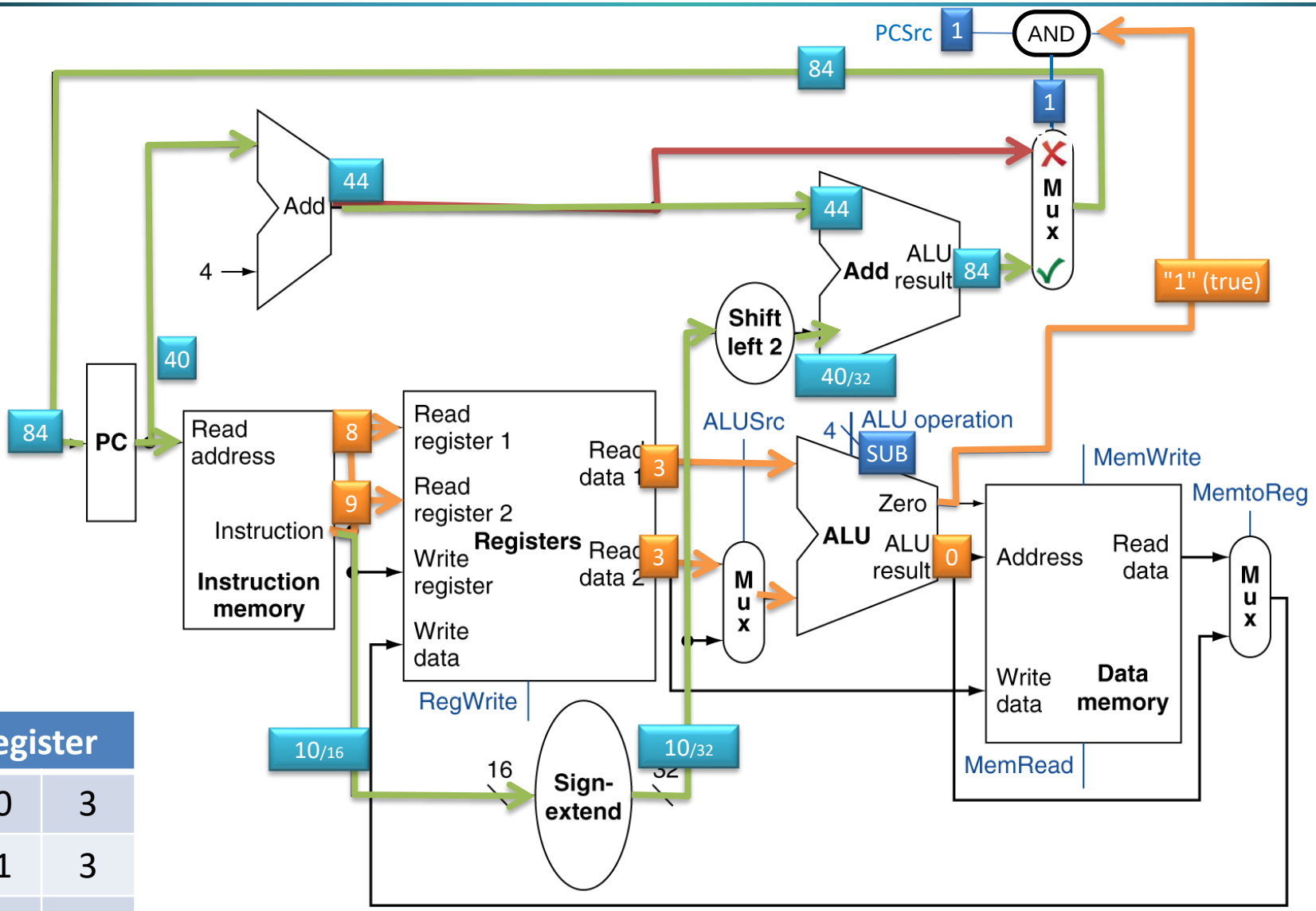
Vollständiger Datapath



Kein Branch – Nächste Instruktion



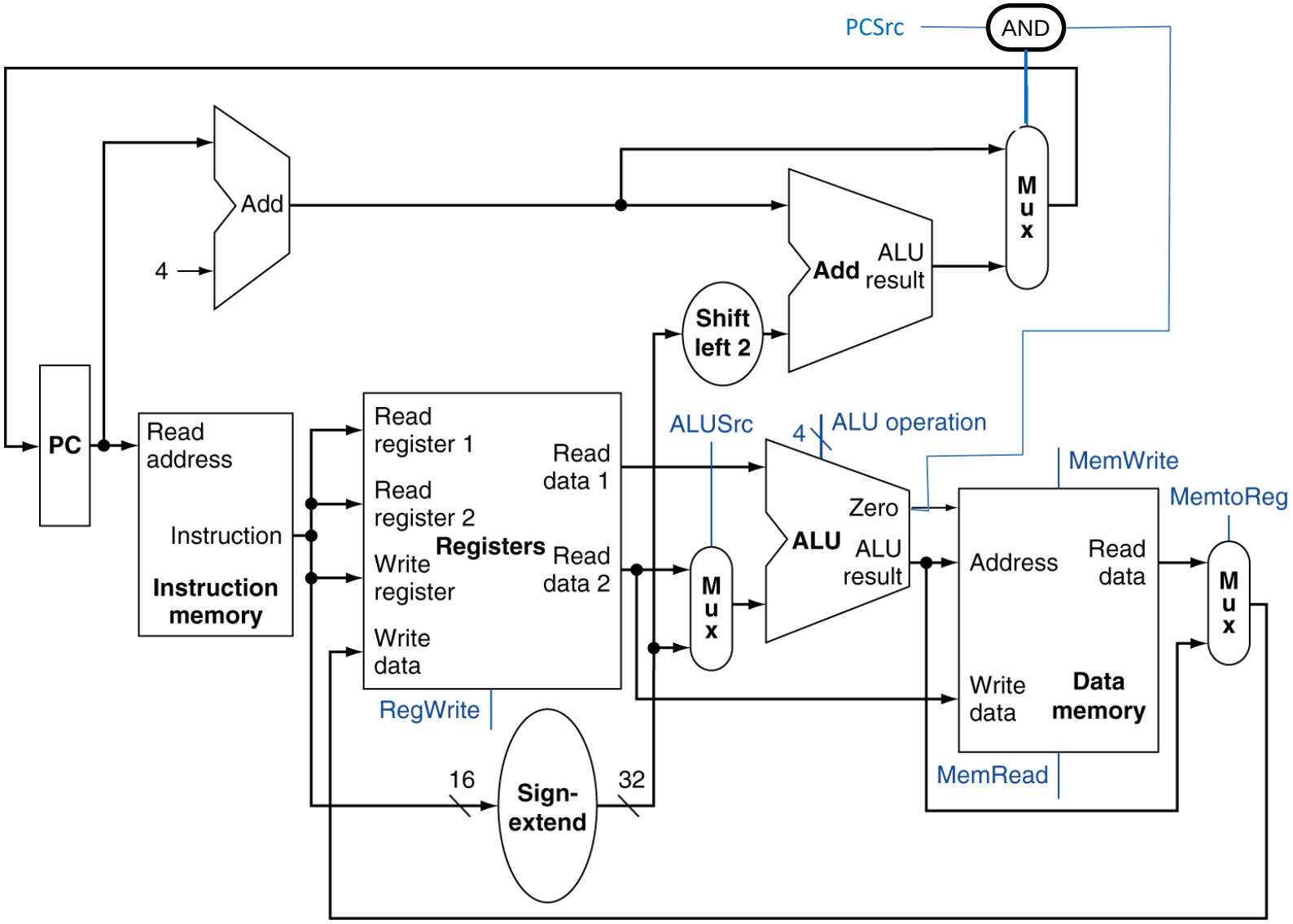
Branch – beq $t0, $t1, 10



Register	
$t0	3
$t1	3
$pc	40

beq $t0, $t1, 10 ➔ 000100 01000 01001 0000 0000 0000 1010

Datenpfad


PCSrc
AND
Add
4
M
u
x
Add
ALU
result
Shift
left 2
PC
Read
address
Instruction
Instruction
memory
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU operation
M
u
x
ALU
Zero
ALU
result
MemWrite
MemtoReg
Address
Read
data
Write
data
Data
memory
MemRead
M
u
x
16
Sign-
extend
32

Kapitel 3 - Gliederung

Kapitel 3: Prozessor – Datenpfad und Steuerwerk

3.1 Aufbau einer CPU

3.2 Datenpfad (Datapath)

3.3 Steuerwerk (Control)

3.3.1 Datapath mit Control-Komponenten und Steuerbefehlen

3.3.2 Steuerbefehle für ausgewählte Instruktionen

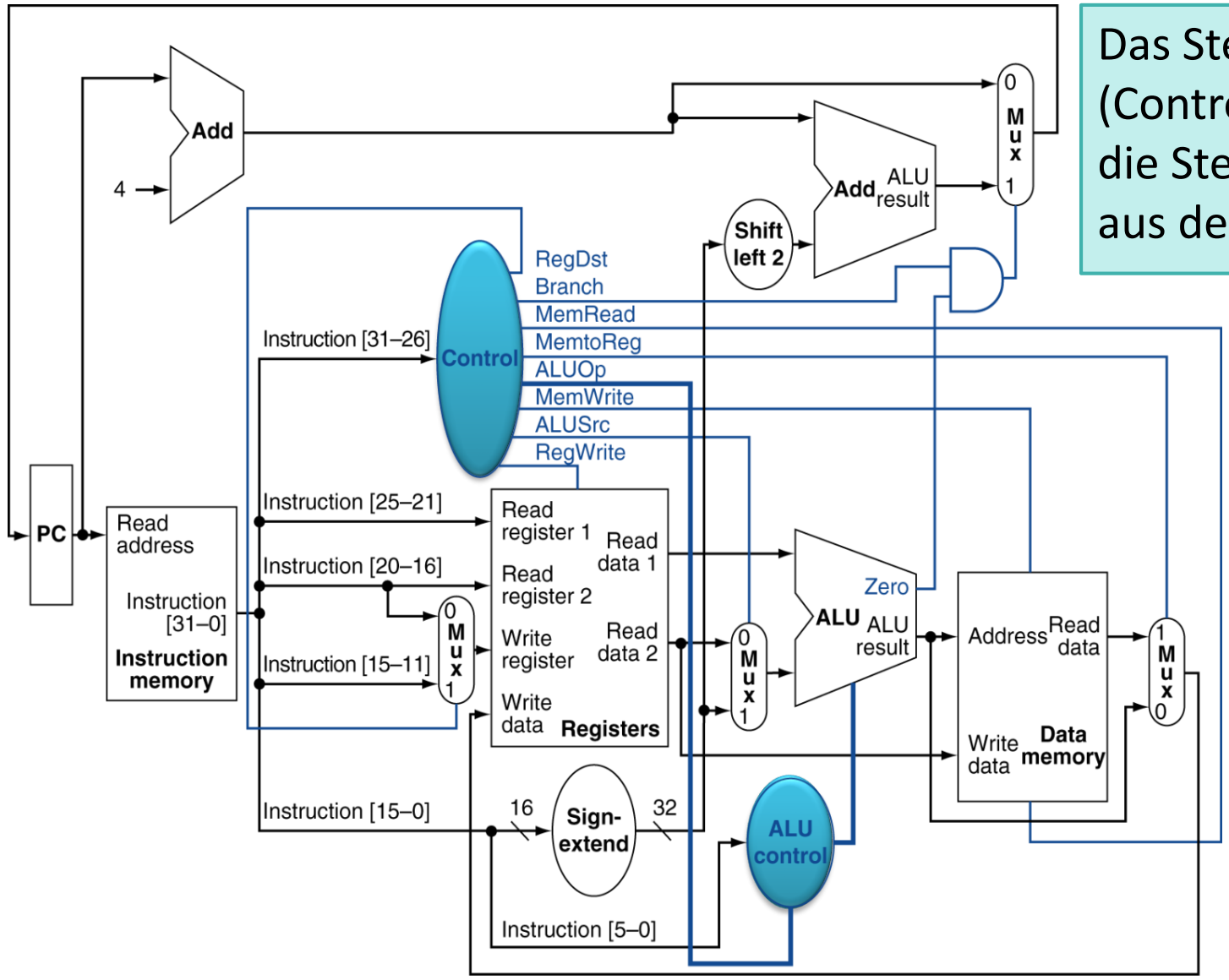
3.4 Zusammenfassung und Bewertung

Control

Wir haben bereits einige Steuerbefehle gesehen. Die Steuerbefehle werden in den Control-Elementen aus der Instruktion generiert. Die Steuerelemente sind **kombinatorische Elemente** ohne Speicher und werden dementsprechend über **logische Schaltkreise** realisiert.

- Control ist in zwei Elemente aufgeteilt
  - Main Control (Hauptsteuerung)
    - kontrolliert den Rest des Datenpfades
      - Registersatz
      - Speicherblock
      - verschiedene Multiplexer
      - ALU Operation
  - ALU Control (ALU-Steuerung)
    - erhält von der Main Control Informationen über die Instruktion
    - bildet diese in einen 4-Bit Steuerbefehl der ALU ab
      - ALU kann mit bis zu 16 verschiedenen Operationen angesteuert werden

Datapath mit Control

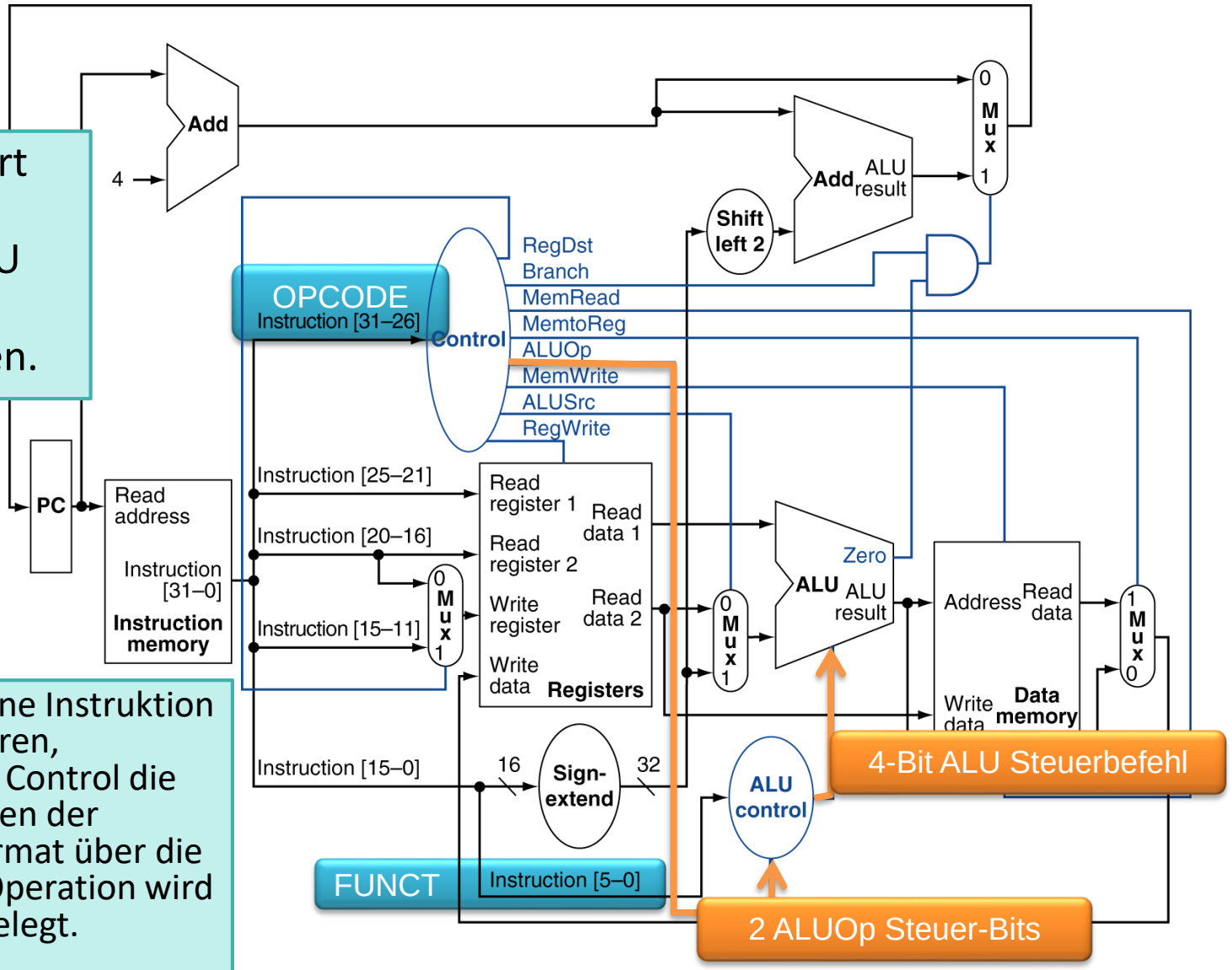


Das Steuerwerk (Control) bestimmt die Steuersignale aus dem OPCODE.

Das ALU Steuerwerk (ALU Control) bestimmt die Operation der ALU.

Datapath mit Control: ALU Control

Control komprimiert den OPCODE in 2 ALUOp Bits, die ALU Control die Art von Instruktion anzeigen.



Falls die ALUOp Bits eine Instruktion im R-Format spezifizieren, unterscheidet die ALU Control die zahlreichen Operationen der Instruktionen im R-Format über die FUNCT-Bits. Die ALU Operation wird dann über 4 Bits festgelegt.

ALU Operation

Die ALU muss abhängig von der Instruktionen unterschiedliche Operationen durchführen. Bei einem Datentransfer die Addition von Register und Offset, bei einem Branch-Befehl eine Subtraktion und bei einem R-Format-Befehl unterschiedliche Operationen je nach dem, was im FUNCT-Feld steht. Die Tabelle zeigt einige 4-Bit-Codierungen der ALU-Operationen, mit denen die ALU Control die ALU steuert.

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

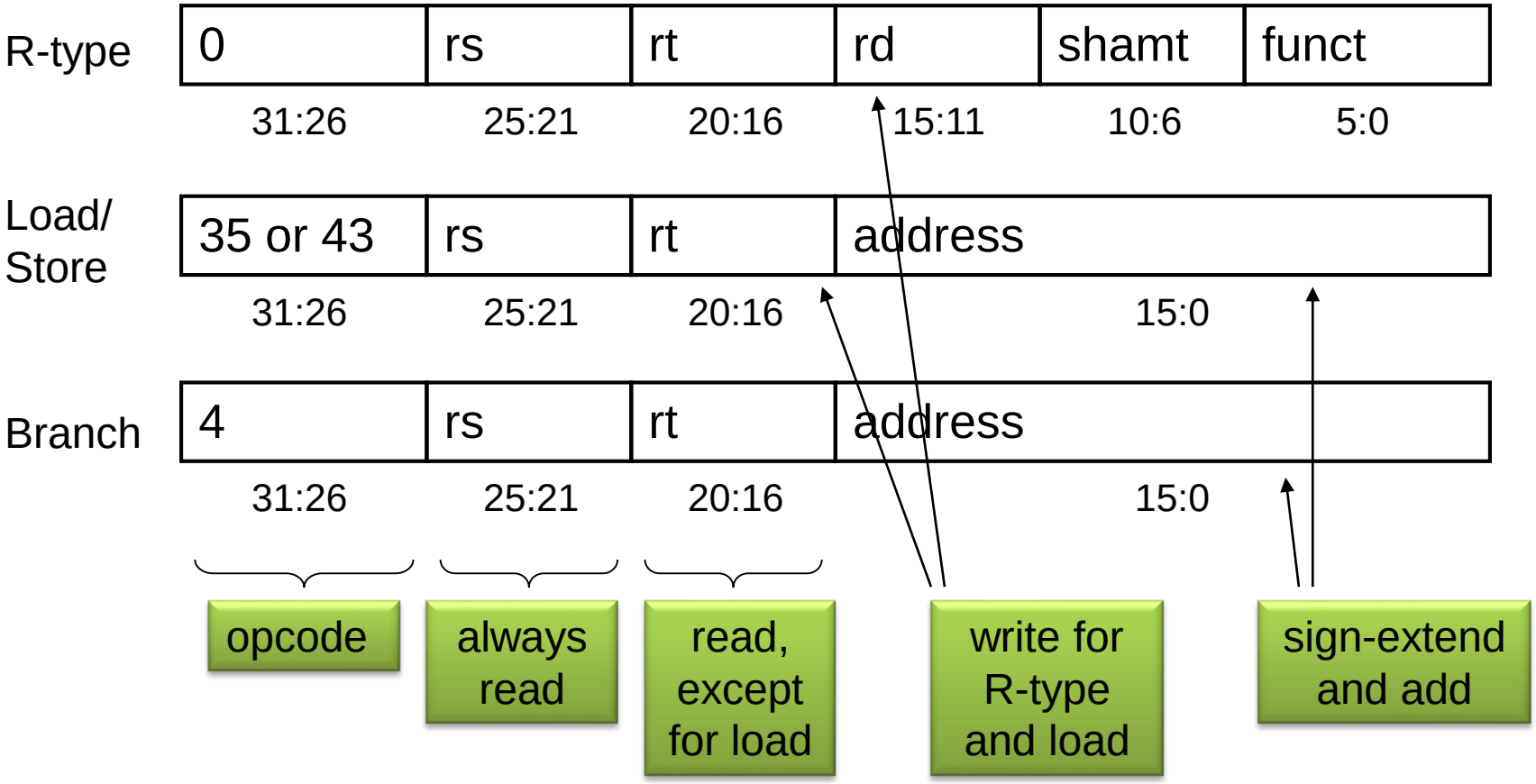
ALU Control

- ALU Control ist ein kombinatorisches Element, das aus den ALUOp-Steuerbits und den 6 Bits des FUNCT Feldes die Funktion der ALU bestimmt.
- Die Main-Control leitet die zwei ALUOp-Steuerbits aus dem OPCODE ab.
- Die ALU Control codiert die ALU Funktion in einem 4-Bit ALU Steuerbefehl.

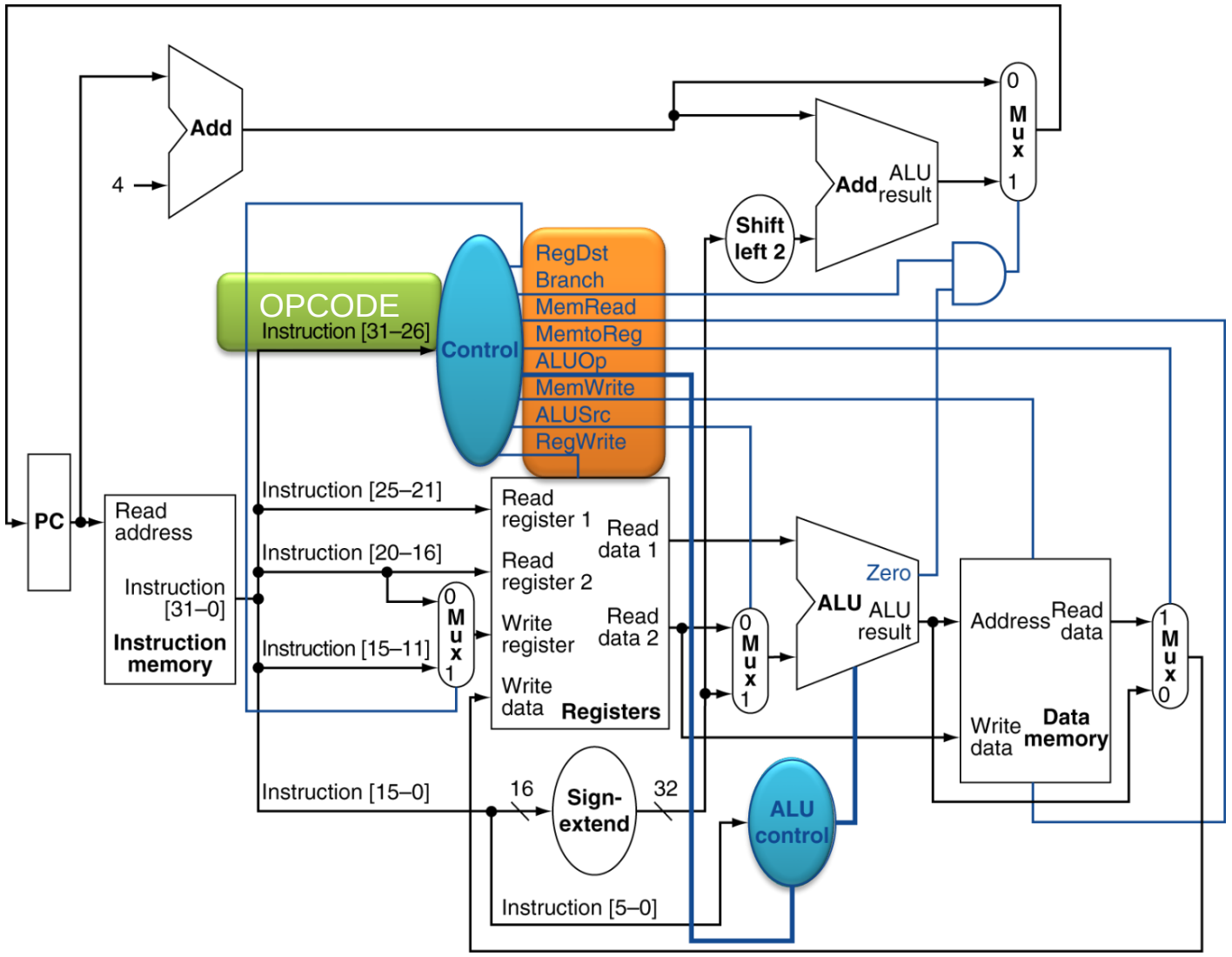
opcode	ALUOp Steuerbits	Operation	funct	ALU function	ALU Steuerbefehl
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

Main Control (Hauptsteuerungselement)

- Main Control muss die unterschiedlichen Instruktionsformate verstehen und aus den entsprechenden Bits die Steuersignale der verschiedenen Hardware-Komponenten des Datenpfades generieren. Je nach Format gibt es zwei oder drei Register und insbesondere das rt-Feld wird bei "load" als Nummer des zu schreibenden Registers verwenden und ansonsten als zu lesendes Register.
- Main Control ist ein kombinatorisches Element mit 6 Eingängen (dem OPCODE) und achte Ausgängen, wobei wie gesehen ALUOp aus zwei Bits besteht.



Datapath mit Control



Kapitel 3 - Gliederung

Kapitel 3: Prozessor – Datenpfad und Steuerwerk

3.1 Aufbau einer CPU

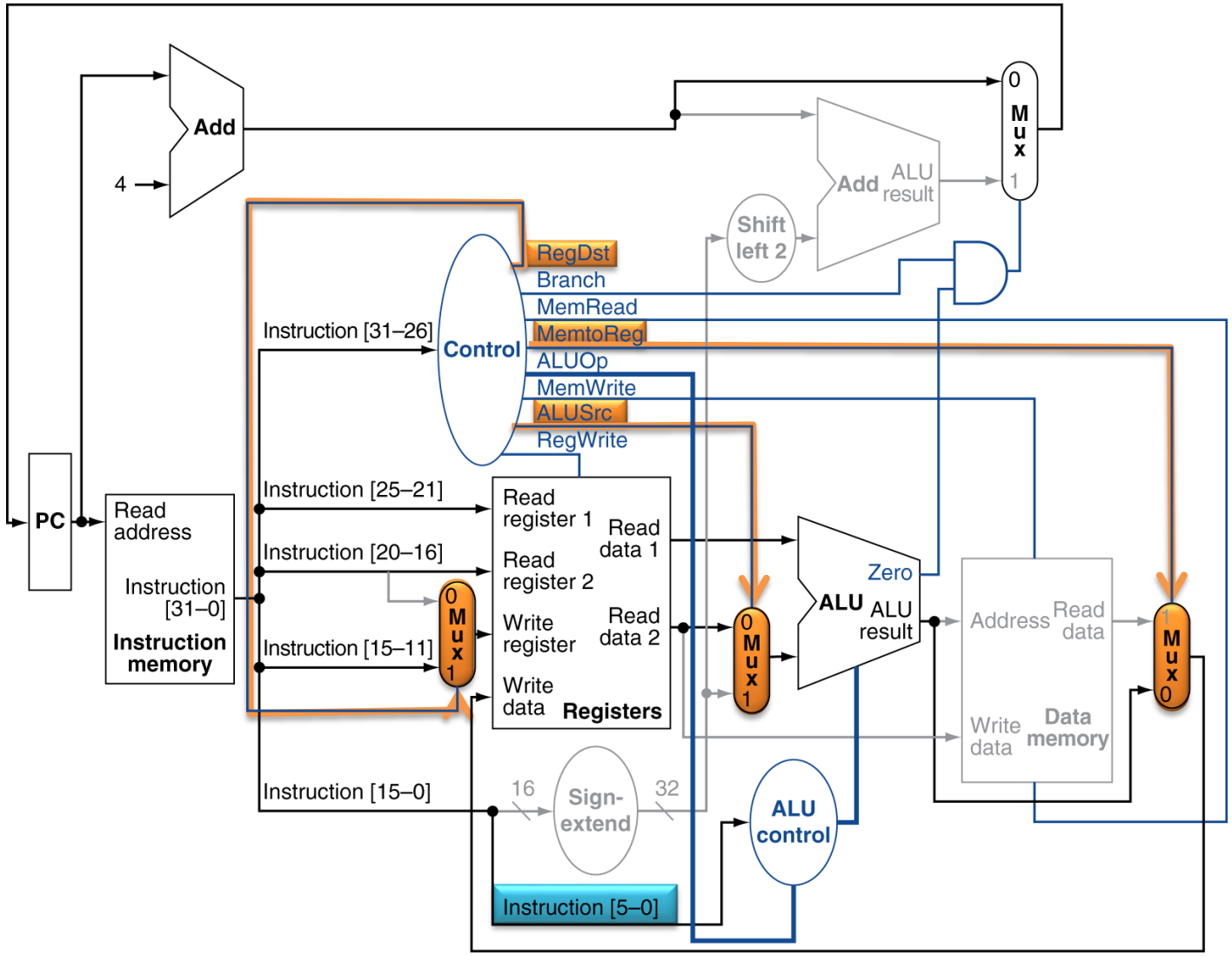
3.2 Datenpfad (Datapath)

3.3 Steuerwerk (Control)

- 3.3.1 Datapath mit Control-Komponenten und Steuerbefehlen
- **3.3.2 Steuerbefehle für ausgewählte Instruktionen**

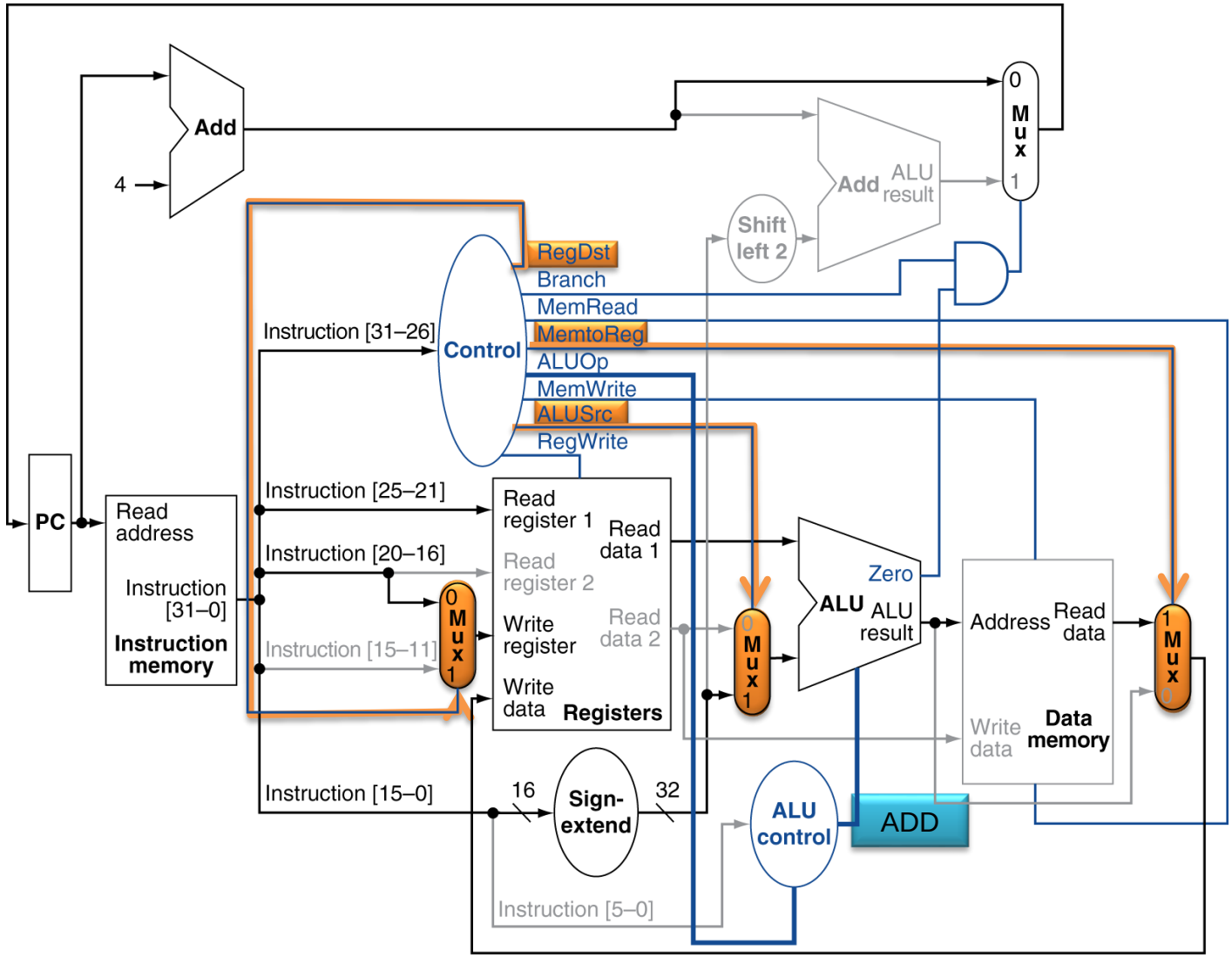
3.4 Zusammenfassung und Bewertung

R-Format Instruktionen



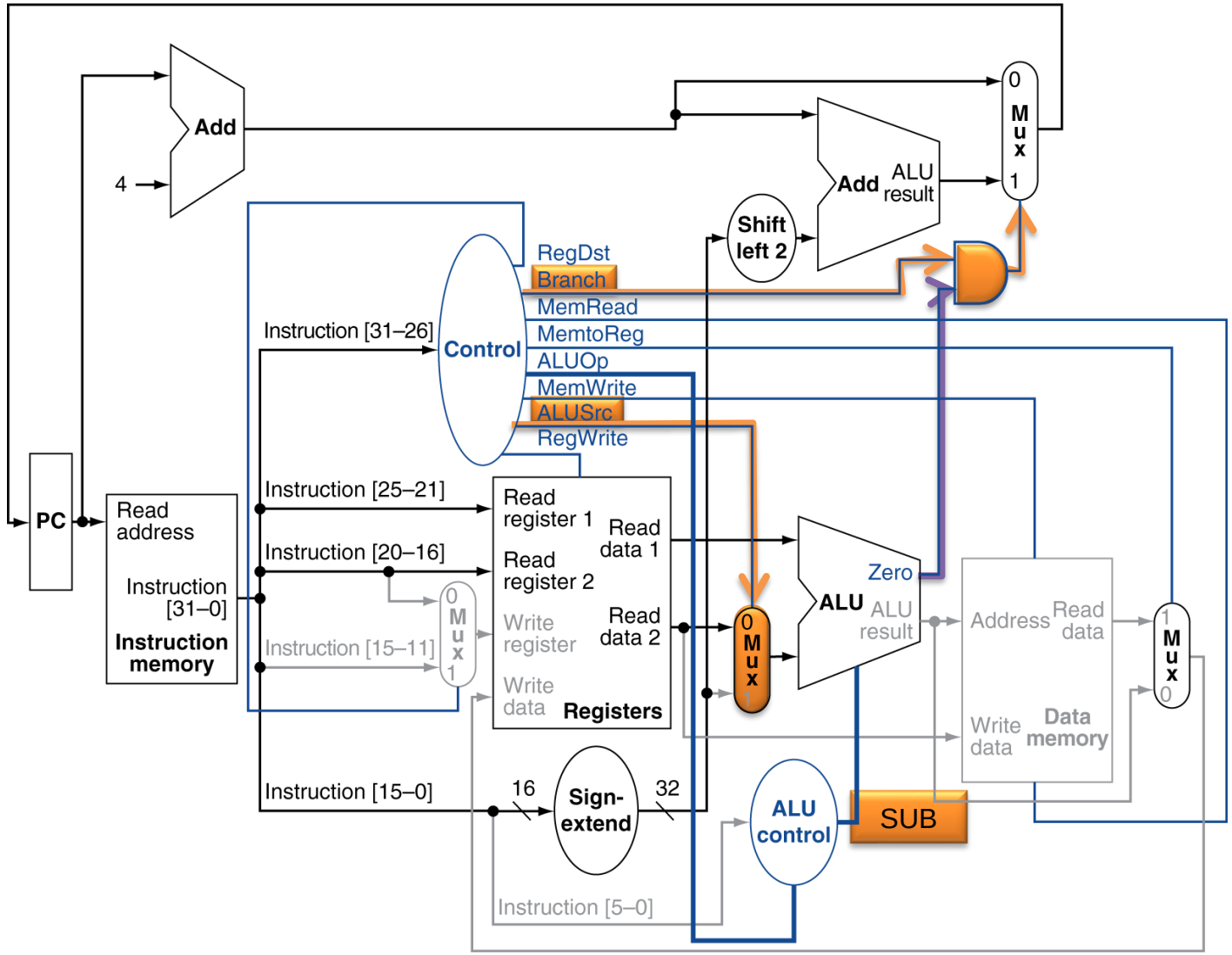
Steuer-Bits	R-Format
RegDst	1
Branch	0
MemRead	0
MemToReg	0
ALUOp	10
MemWrite	0
ALUSrc	0
RegWrite	1

Load



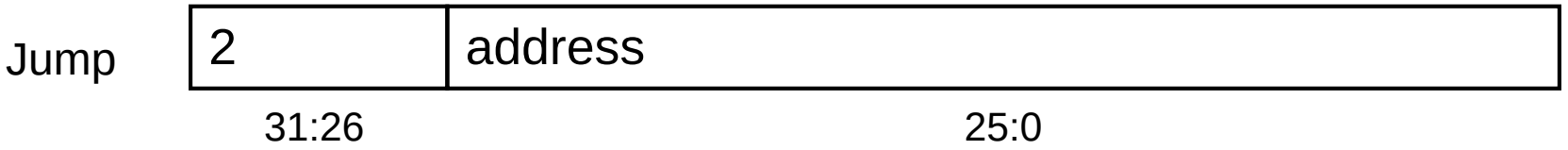
Steuer-Bits	lw
RegDst	0
Branch	0
MemRead	1
MemToReg	1
ALUOp	00
MemWrite	0
ALUSrc	1
RegWrite	1

Branch-on-equal



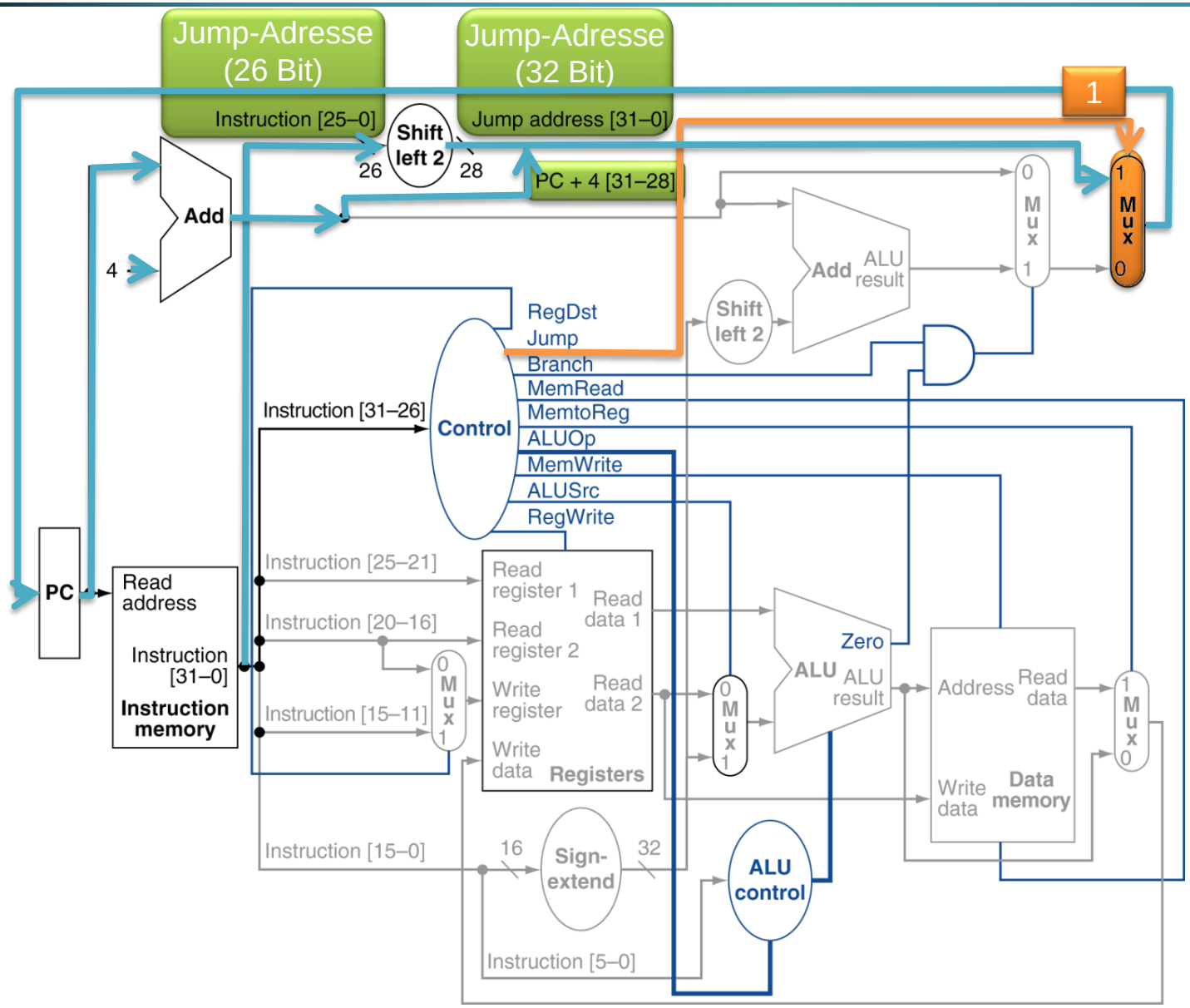
Steuer-Bits	beq
RegDst	0
Branch	1
MemRead	0
MemToReg	0
ALUOp	01
MemWrite	0
ALUSrc	0
RegWrite	0

Sprünge - Jump



- Sprungadresse:
  - in Word: nach Byte umrechnen
  - erste Bits 28-31 von Program Counter übernehmen
- Main Control:
  - mit Program Counter-"Pfad" zusammenführen

Sprünge - Jump



Steuer-Bits	R-Format
RegDst	0
Branch	0
MemRead	0
MemToReg	0
ALUOp	00
MemWrite	0
ALUSrc	0
RegWrite	0
Jump	1

Kapitel 3 - Gliederung

Kapitel 3: Prozessor – Datenpfad und Steuerwerk

3.1 Aufbau einer CPU

3.2 Datenpfad (Datapath)

3.3 Steuerwerk (Control)

3.4 Zusammenfassung und Bewertung

Bewertung der MIPS CPU

- Single-Cycle CPU
  - dargestellte CPU führt alle Instruktionen in einem Takt aus
  - Taktdauer bestimmt durch die Zeit bis sich alle Eingänge ein stabiles Ergebnis erreicht haben
    - Zeit bis die "Spannungsänderung an den Ausgänge alle elektronischen Schaltkreise (kombinatorischen Elemente) durchlaufen haben und an den Eingängen angekommen sind"
      - Ausgänge: PC, Instruktionsspeicher, Registersatz, Datenspeicher
      - Eingänge: PC, Register, Daten
    - Je komplexer die Schaltkreise desto größer die Taktdauer
  - "Load"-Instruktion hat längsten Pfad von Eingang bis Ausgang
    - Instruktion, Register, Sign Extension, ALU, Datenspeicher, Register
    - alle Instruktion brauchen so lange wie das "seltene" load
- Lösung: Pipelining