IT-Infrastrukturen – Rechnerstrukturen

Thema 4: Entwurf eines MIPS-Prozessors

Prof. Dr.-Ing. Sebastian Schlesinger
Professur für Wirtschaftsinformatik
(Infrastruktur & Security)

Dieser Foliensatz basiert auf den Folien zu „Rechnerorganisation“ von Prof. Dr. Ben Juurlink (TU Berlin) und Prof. Dr. Paula Herber (WWU Münster)

Bisheriger Inhalt der Vorlesung

- Welche Information versteht ein Rechner?
  - Zahlendarstellung
  - Rechnerarithmetik
- Wie verarbeitet man Daten und berechnet Ergebnisse?
  - Schaltwerksentwurf, Speicherelemente, ALU
- Wie kommt man von Hochsprachen (C, C++, Java…) zur Maschinensprache?
  - Befehlssatz, MIPS Assembler
  - Maschinensprache

Frage dieser Vorlesung:

- Wie wird Maschinensprache in Hardware ausgeführt?

Lernziele

- Nach diesem Kapitel sind Sie in der Lage
  - Einen Datenpfad zu entwerfen, der einen Teil des MIPS-Befehlssatzes implementiert.
  - Die Steuersignale anzugeben, die für die Ausführung bestimmter Befehle gebraucht werden.
  - Befehle zu dem unterstütztem Befehlssatz hinzuzufügen.
  - Zu erklären, warum Eintakt-Implementierungen ineffizient sind.
  - Die verschiedenen Ausführungsschritte von Befehlen zu erklären.
  - Befehle in Ausführungsschritte zu zerlegen und die Hardware zur Umsetzung einer Mehrzyklen-Implementierung zu entwerfen.

Überblick

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

1. Einleitung
2. Vorbereitung: ALU-Erweiterung für **`slt`** und **`beq`**
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

Überblick

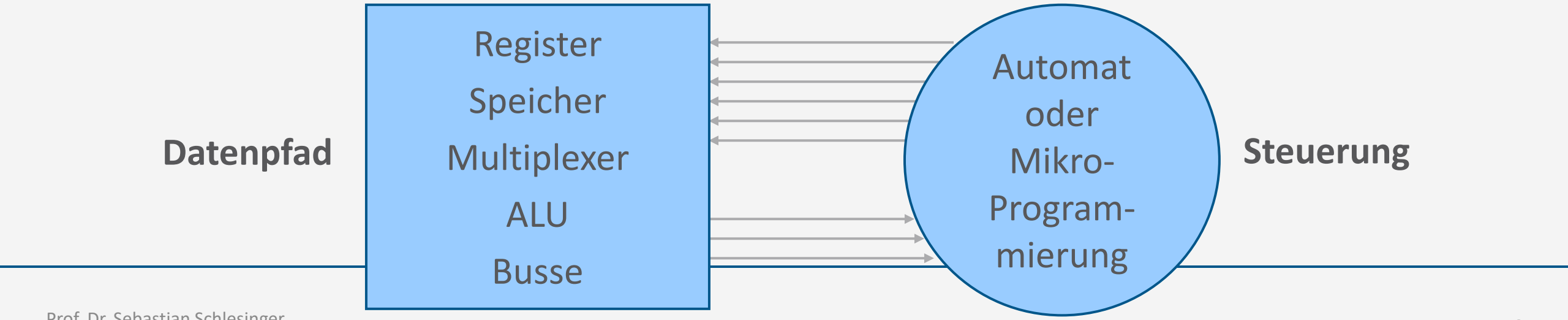
Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

1. Einleitung
2. Vorbereitung: ALU-Erweiterung für `slt` und `beq`
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

Datenpfad und Steuerung

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Datenpfad (*datapath*)
  - Hardwarekomponenten, die Operationen ausführen, und deren Verbindungen
  - Muskeln des Prozessors
- Steuerung (*control*):
  - Teil des Prozessors, der den Ablauf der Befehlsverarbeitung durch Steuersignale an den Datenpfad steuert
  - Gehirn des Prozessors



Eine einfache MIPS-Implementierung

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Zur Vereinfachung nur folgende Befehle:
  - Datentransport-Befehle: `lw, sw`
  - arithmetisch-logische Befehle: `add,sub,and,or,slt`
  - Kontrollfluss-Befehle: `beq, j`
- **Was ist zu tun um die Befehle auszuführen? Welche Bauteile werden benötigt?**

Eine einfache MIPS-Implementierung

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Zur Vereinfachung nur folgende Befehle:
  - Datentransport-Befehle: `lw, sw`
  - arithmetisch-logische Befehle: `add,sub,and,or,slt`
  - Kontrollfluss-Befehle: `beq, j`
- **Was ist zu tun um die Befehle auszuführen? Welche Bauteile werden benötigt?**
  - Hole Befehl aus dem Speicher
  - Dekodiere den Befehl, um zu entscheiden was zu tun ist (Multiplexer)
  - Lese Register und/oder Speicherinhalte
  - Verwende ALU zur Ausführung von arithmetisch/logischen Befehlen oder zur Berechnung von Adressen
  - Schreibe Ergebnisse (in den Speicher oder in Register)

Überblick

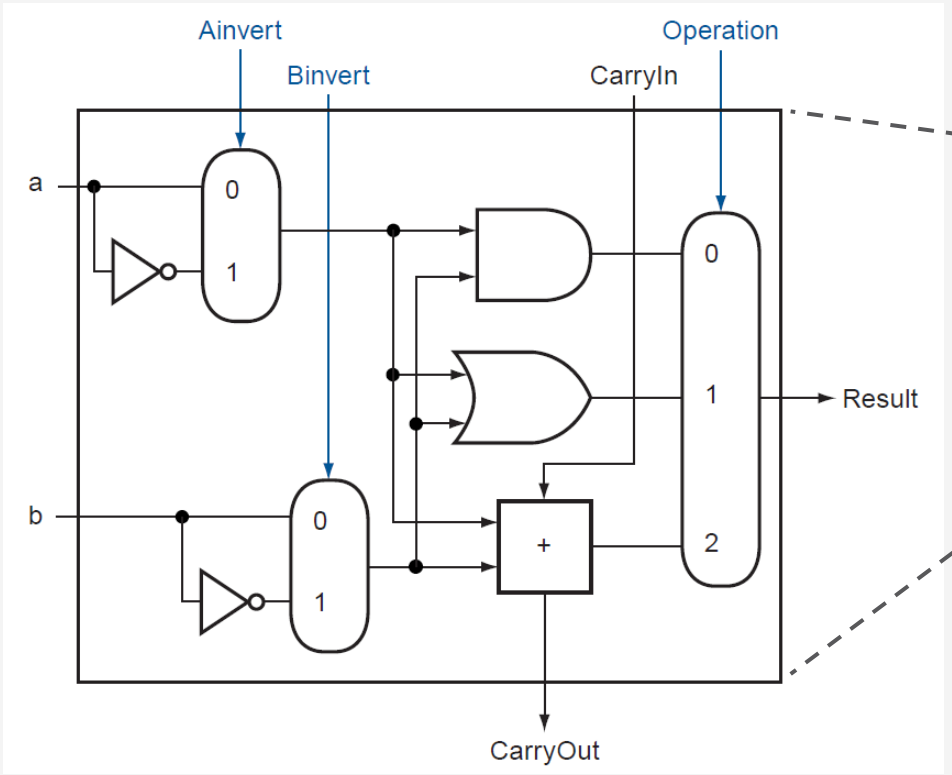
Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

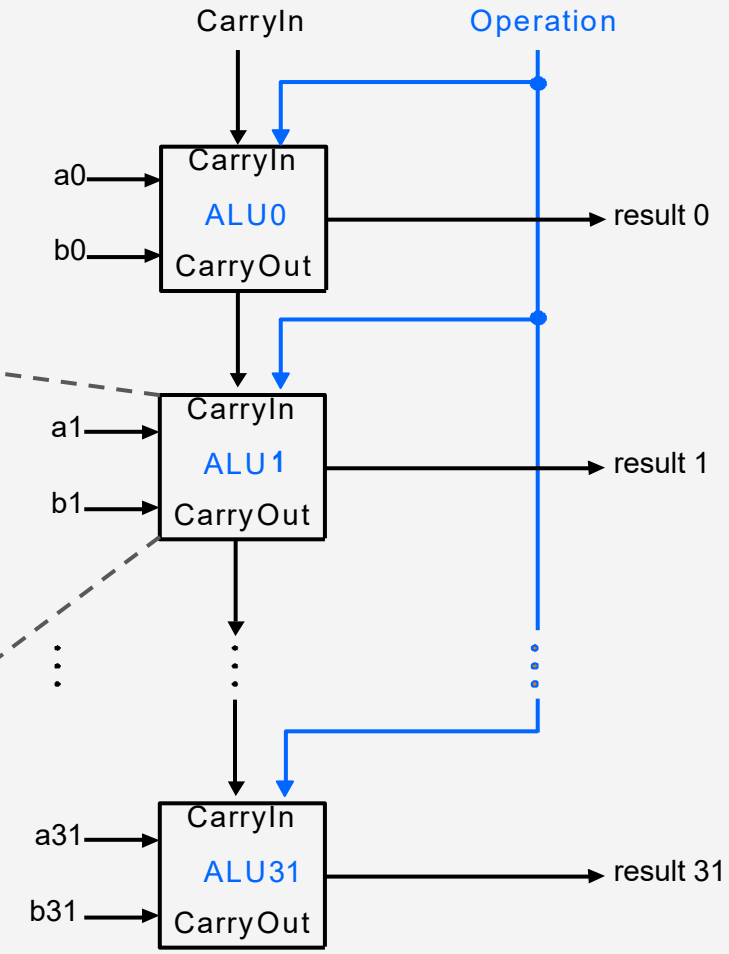
1. Einleitung
2. Vorbereitung: ALU-Erweiterung für **`slt`** und **`beq`**
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

ALU-Erweiterung

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- **Wie muss die ALU erweitert werden, damit die Instruktionen `slt` und `beq` umgesetzt werden können?**





Erweiterung der ALU für slt und beq

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Für **`slt`** muss getestet werden, ob der Inhalt eines Registers kleiner als der Inhalt eines anderen Registers ist.
  - **`slt rd,rs,rt`** produziert 1, wenn **`rs < rt`**, ansonsten 0
  - verwende Subtraktion: (a-b) < 0 → a < b

  berechnetes Ergebnis negativ: wenn MSB = 1 (2-Komplement)
- Für **`beq`** muss getestet werden, ob der Inhalt eines Registers gleich dem Inhalt eines anderen Registers ist.
  - **`beq rs,rt,offset`** springe zu PC + 4 + 4 · offset, wenn **`rd = rs`**
  - verwende wieder Subtraktion: (a-b) = 0 → a = b

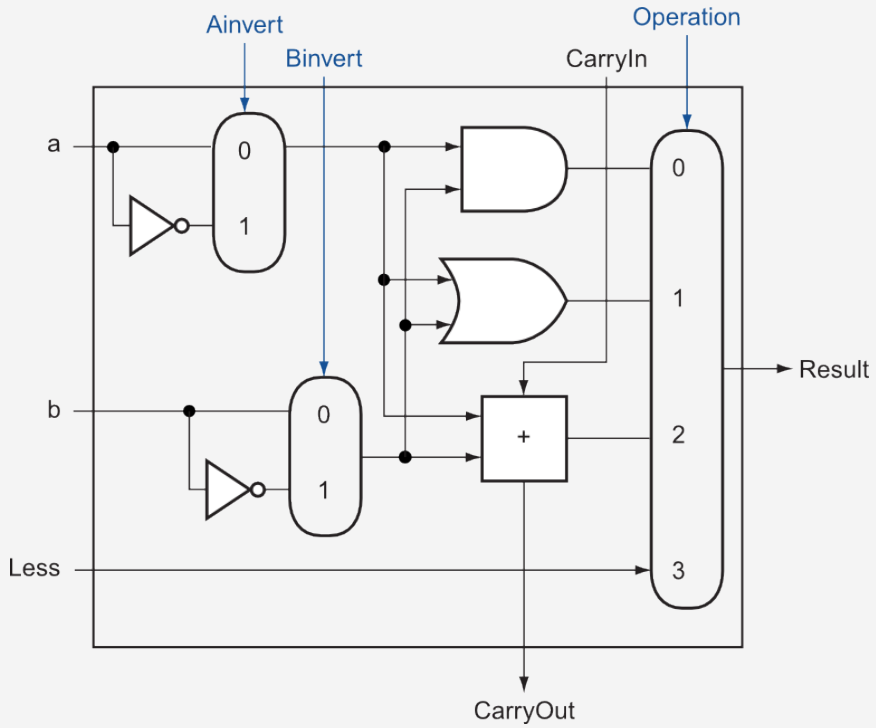
  berechnetes Ergebnis Null: wenn alle Bits 0 sind

Erweiterung der ALU für slt: Less und Set

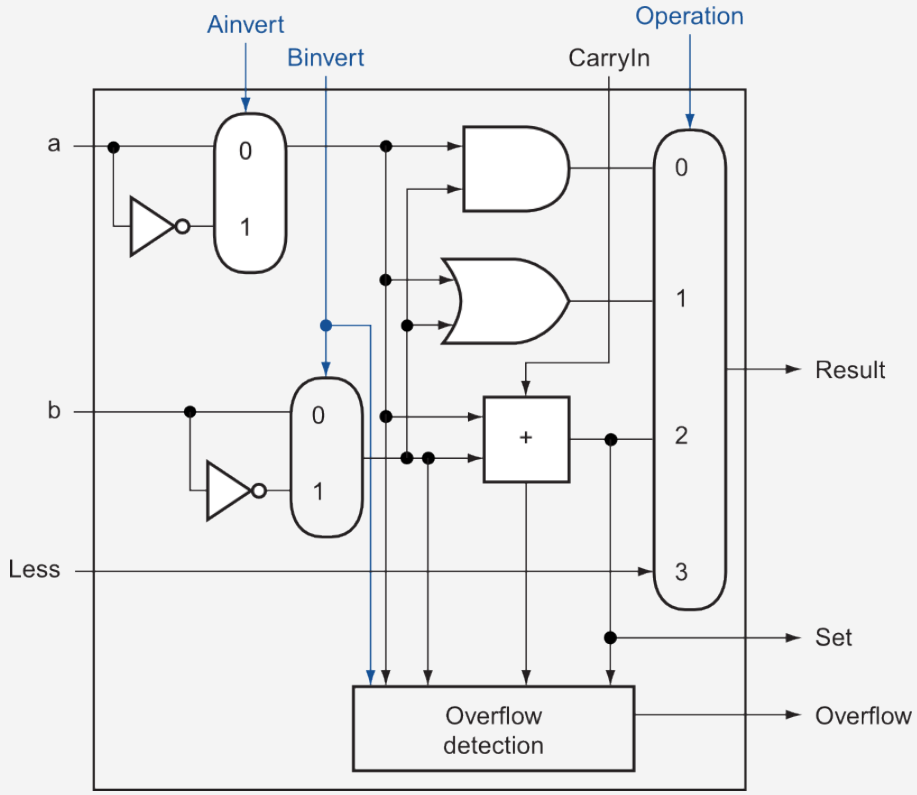
Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Idee: LSB soll auf Vorzeichenbit des Ergebnisses der Subtraktion gesetzt werden, Rest 0
- zusätzlicher Eingang *Less*, wird auf 0 gesetzt für alle 1-Bit ALUs außer ALU_0
- Für ALU_0: *Less* = *Set*-Output von ALU_{31}

ALU_0..ALU_{30}:

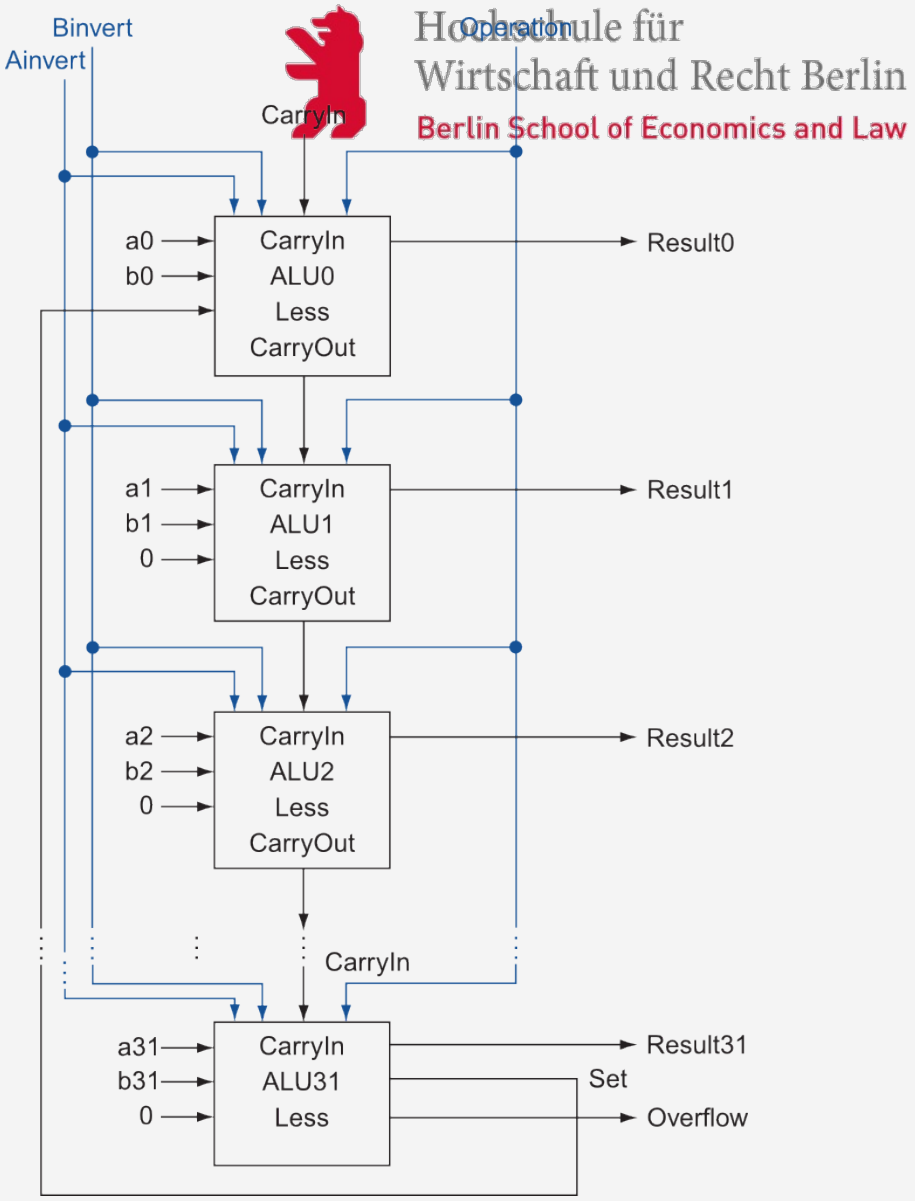


ALU_{31}:



Erweiterung der ALU für slt

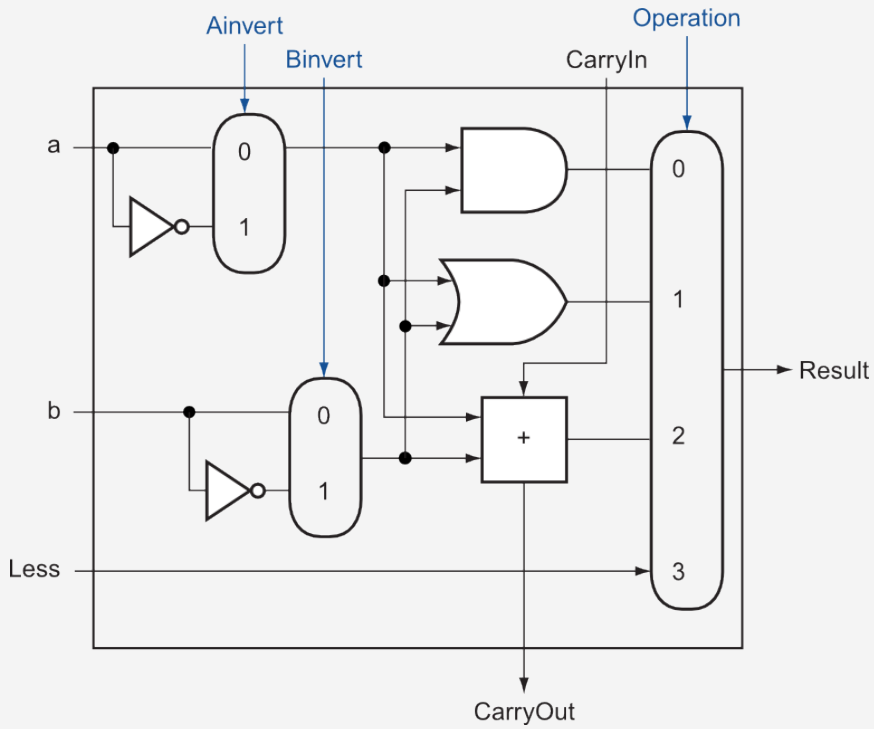
- zusätzlicher Eingang *Less*, wird auf 0 gesetzt für alle 1-Bit ALUs außer ALU_0
- Für ALU_0: *Less* = *Set*-Output von ALU_{31} = Vorzeichenbit des Ergebnis der Subtraktion



Erweiterung der ALU für slt: Steuersignale

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- zusätzlicher Eingang *Less*, wird auf 0 gesetzt für alle 1-Bit ALUs außer ALU_0
- Für ALU_0: *Less* = *Set*-Output von ALU_{31} = Vorzeichenbit des Ergebnis der Subtraktion



ALU-Steuersignale für slt:

- Ainvert: 0
- Binvert: 1
- CarryIn: 1

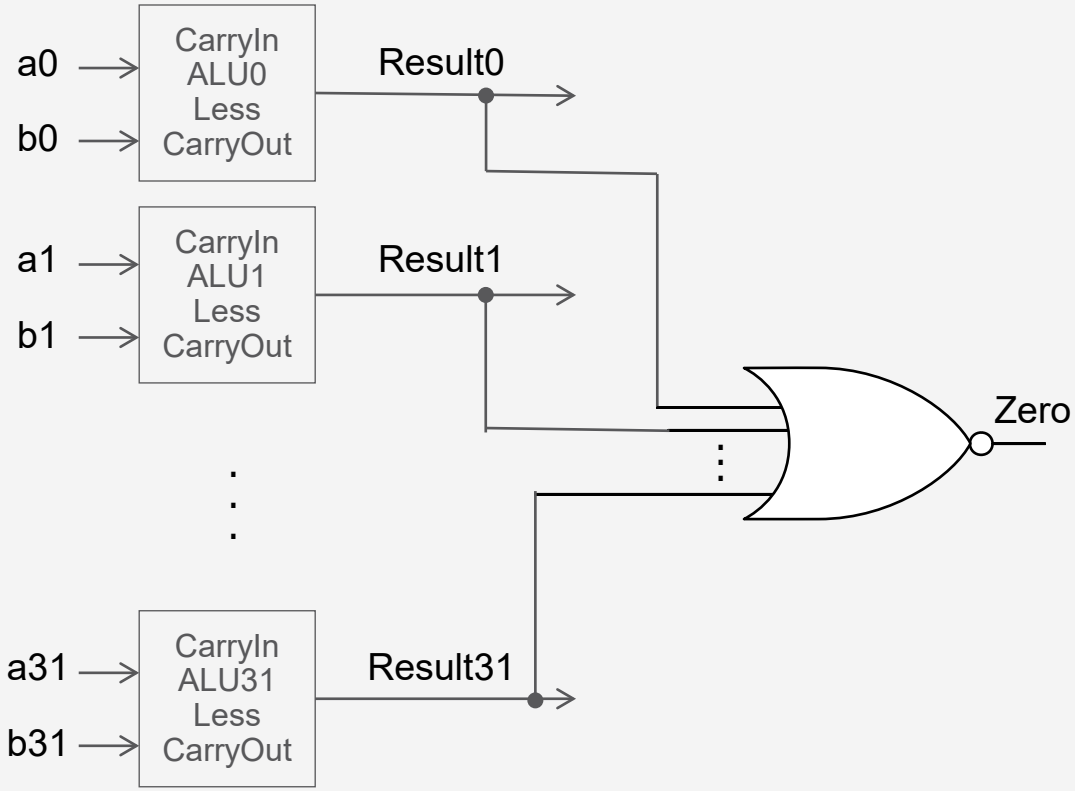
} subtrahiere b von a

- Operation: 11 — selektiere *Less*

Erweiterung der ALU für beq


```
beq rs,rt,offset # if (Reg[rs]==Reg[rt]) PC += 4+4·offset
```

- Benutze Subtraktion:
  - Setze Zero-Ausgang falls alle Result-Bits = 0
  - Steuersignale wie Subtraktion:
    Ainvert = 0;
    Binvert = CarryIn = 1;
    Operation = 10 (*Adder*-Output)



ALU Steuersignale

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Gewünschte ALU-Aktion	*Ainvert*	*Binvert*	*CarryIn*	*Operation*
and	0	0	*	00
or	0	0	*	01
add	0	0	0	10
sub	0	1	1	10
slt	0	1	1	11
nor	1	1	*	00
nand	1	1	*	01

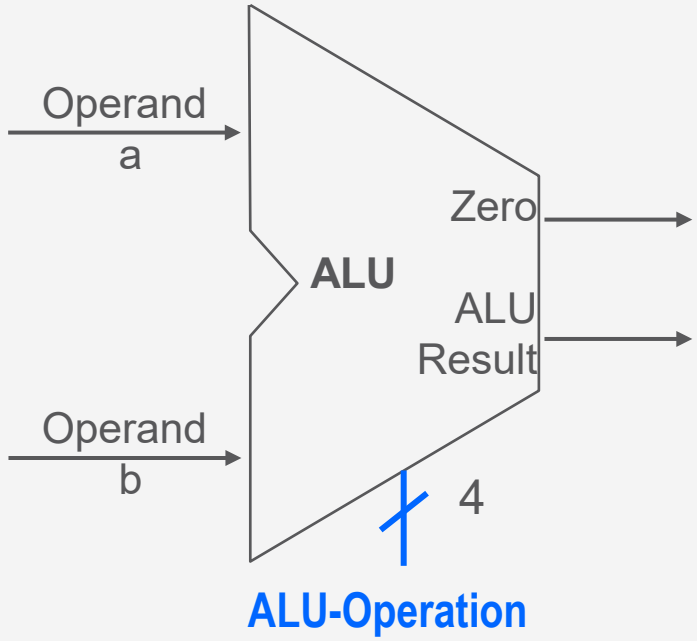
Gewünschte ALU-Aktion	*Ainvert*	*Bnegate*	*Operation*
and	0	0	00
or	0	0	01
add	0	0	10
sub	0	1	10
slt	0	1	11
nor	1	1	00
nand	1	1	01

- *Binvert* und *CarryIn* der ALU0 können zu einem Steuersignal *Bnegate* zusammengefasst werden

ALU Blockschaltbild

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Gewünschte ALU-Aktion	*Ainvert*	*Bnegate*	*Operation*
and	0	0	00
or	0	0	01
add	0	0	10
sub	0	1	10
slt	0	1	11
nor	1	1	00
nand	1	1	01



Überblick

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

1. Einleitung
2. Vorbereitung: ALU-Erweiterung für `slt` und `beq`
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

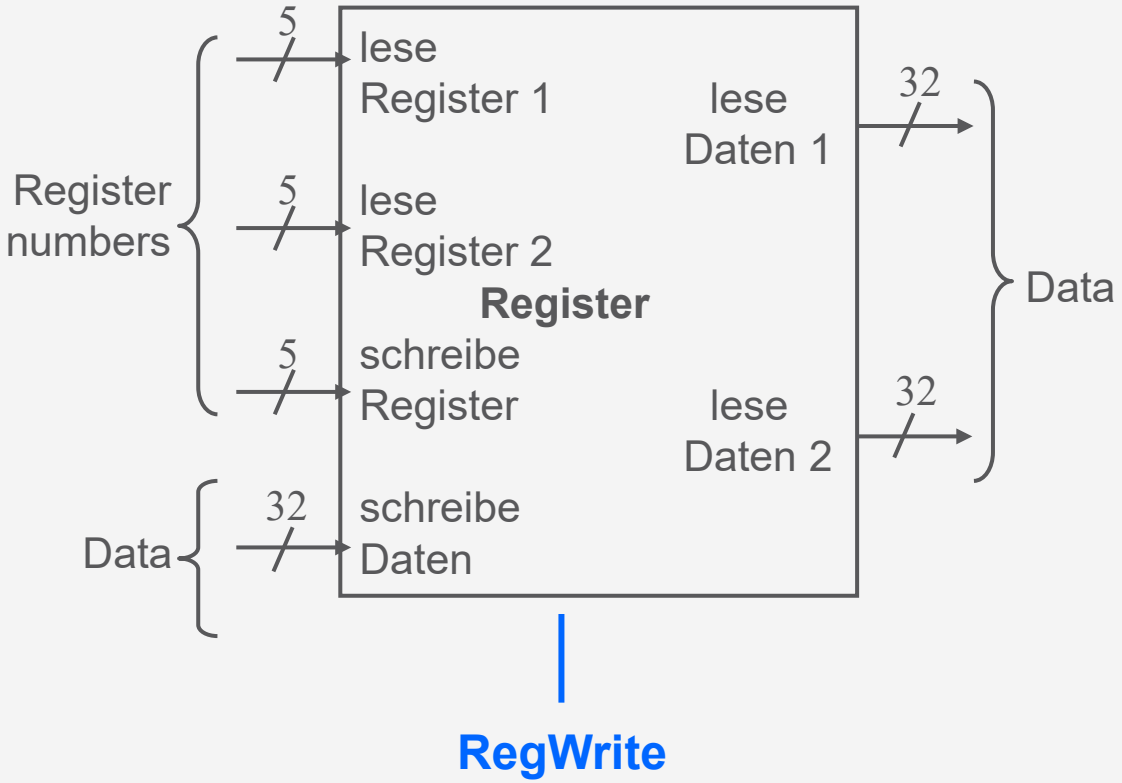
Eine einfache MIPS-Implementierung

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Zur Vereinfachung nur folgende Befehle:
  - Datentransport-Befehle: `lw, sw`
  - arithmetisch-logische Befehle: `add,sub,and,or,slt`
  - Kontrollfluss-Befehle: `beq, j`
- **Ausführung von Befehlen:**
  - Hole Befehl aus dem Speicher
  - Dekodiere den Befehl, um zu entscheiden was zu tun ist
  - Lese Register und/oder Speicherinhalte
  - Verwende ALU zur Ausführung von arithmetisch/logischen Befehlen oder zur Berechnung von Adressen
  - Schreibe Ergebnisse (in den Speicher oder in Register)

Rückblick: Registersatz (*register file*)

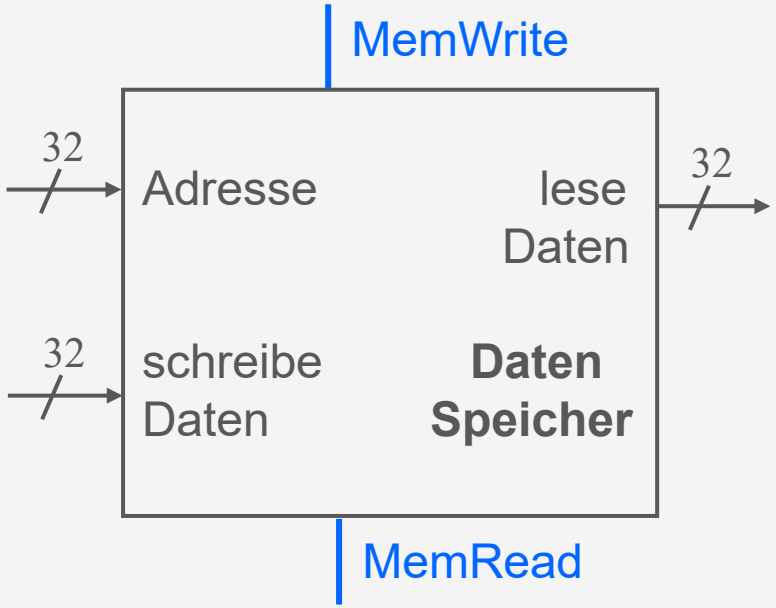
- LeseDaten1 = Register[LeseRegister1]
- Lesedaten2 = Register[LeseRegister2]
- Register[SchreibeRegister] = SchreibeDaten
  - nur wenn RegWrite (==1)
  - am Ende des Taktzyklus (bei fallender Taktflanke)
  - ➢ Es kann das gleiche Register gelesen und geschrieben werden.
  - ➢ also z. B. Reg[5] = Reg[5]+Reg[6] wohldefiniert



Rückblick: Speicher (*memory*)

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Analog zum Registersatz kann auch ein Speicher implementiert werden
- Da die Befehle **`lw`** und **`sw`** jeweils nur auf eine Adresse zugreifen und nicht gleichzeitig ausgeführt werden, genügt ein Speicher mit nur einem Adress-Eingang, einem Schreib-Eingang und einem Lese-Ausgang:



Rückblick: MIPS-Befehlsformate

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- MIPS Befehlsformate:

	6 Bit	5 Bit	5 Bit	5 Bit	5 Bit	6 Bit
R-Format	op	rs	rt	rd	shamt	func
I-Format	op	rs	rt	16-Bit Konstante		
J-Format	op	26-Bit Wort-Adresse				

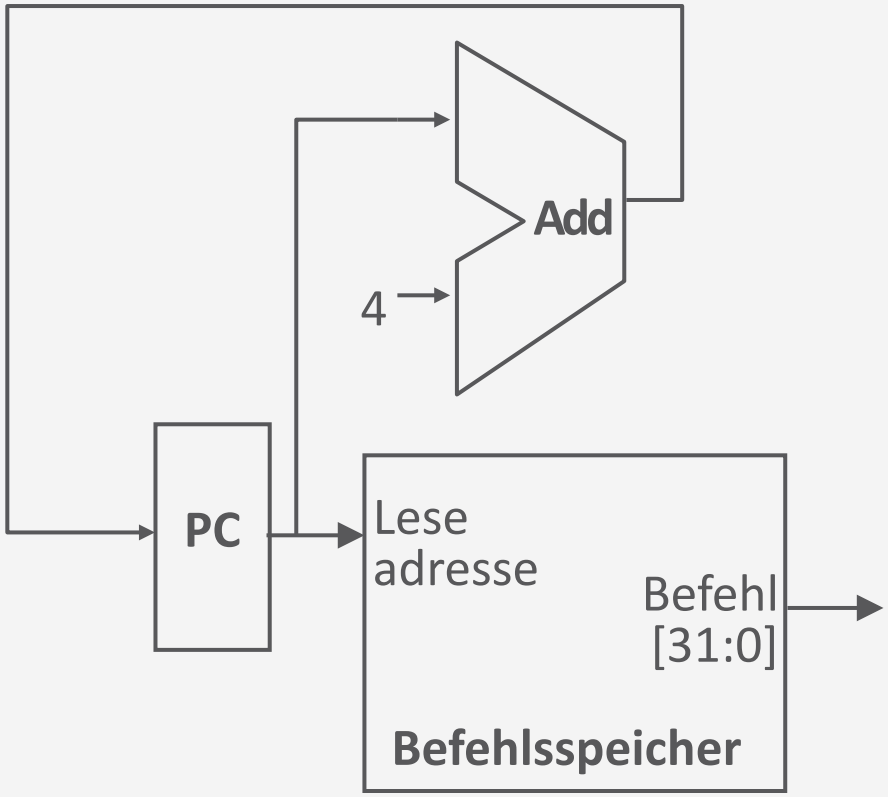
- R-Format: `add $1,$2,$3;  sub, and, or, slt`
- I-Format: `addi $1,$2,100;  lw $1,40($2);  beq $1,$2,100`
- J-Format: `j 100`

Befehle laden

- Welche Bausteine benötigen wir, um den nächsten Befehl zu laden und die nächste Befehlsadresse zu berechnen?
  (hier noch ohne Berücksichtigung von Sprungbefehlen)

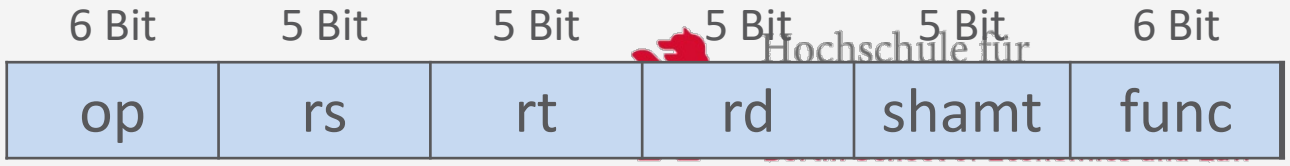
Befehle laden

- Welche Bausteine benötigen wir, um den nächsten Befehl zu laden und die nächste Befehlsadresse zu berechnen?

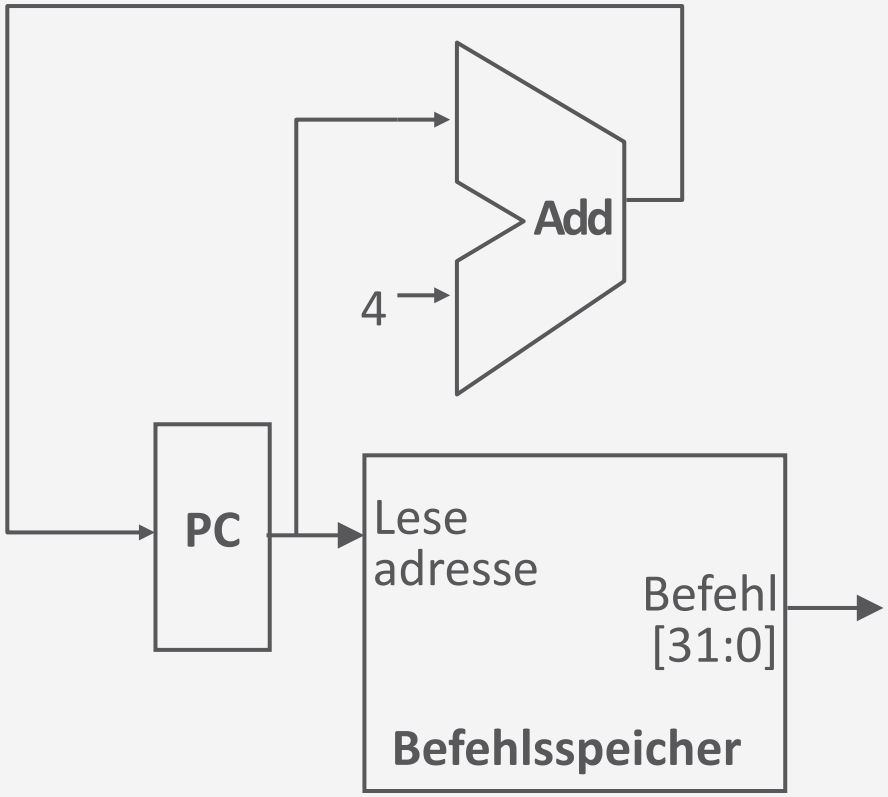


- PC = Program Counter (Adresse der auszuführenden Programmzeile)
- Neuer PC wird zur fallenden Taktflanke gespeichert.

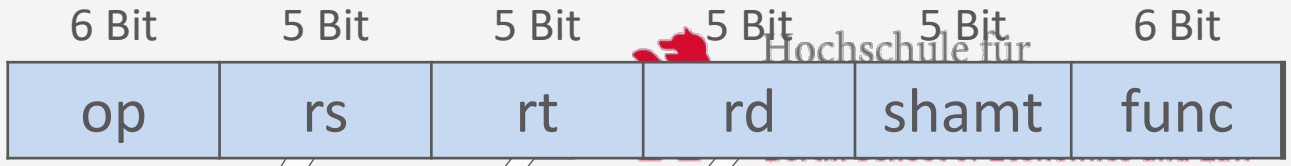
R-Typ Befehle



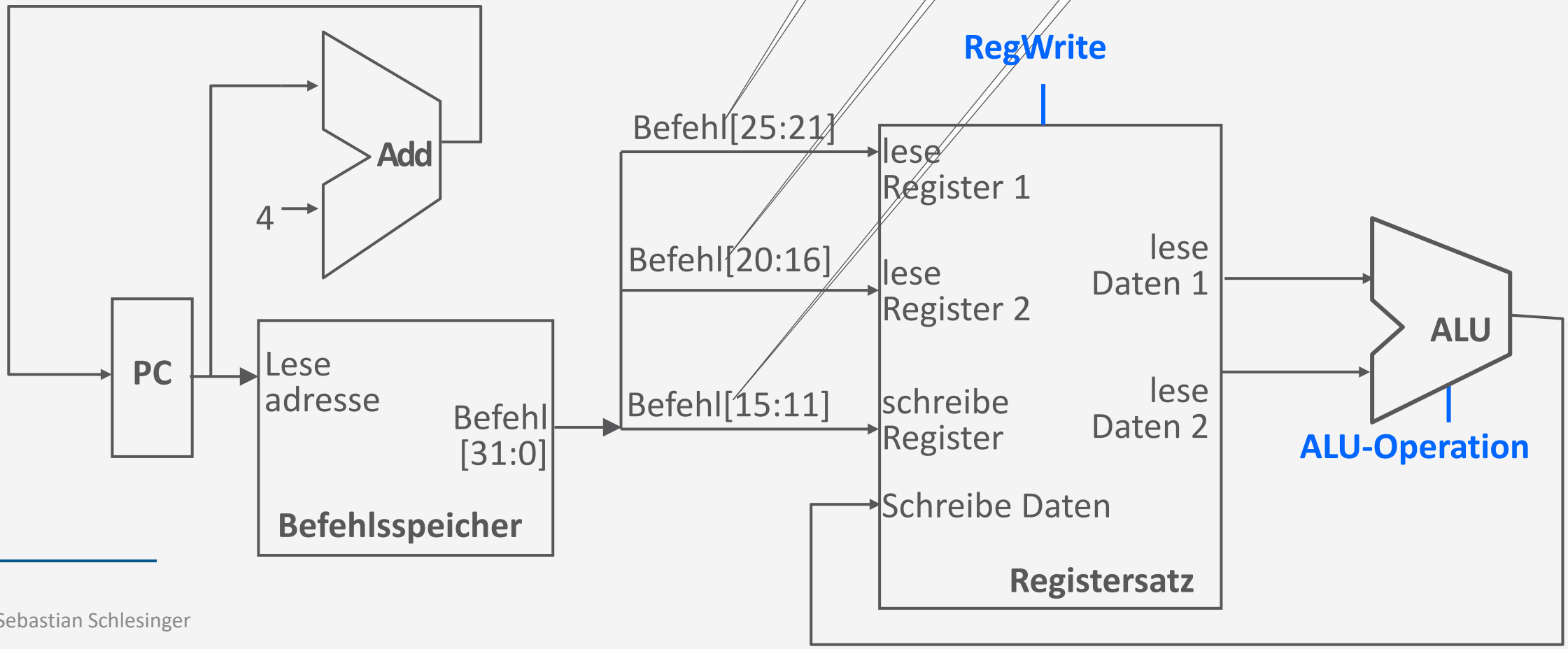
- Was brauchen wir (zusätzlich) um R-Typ Befehle zu realisieren?



R-Typ Befehle



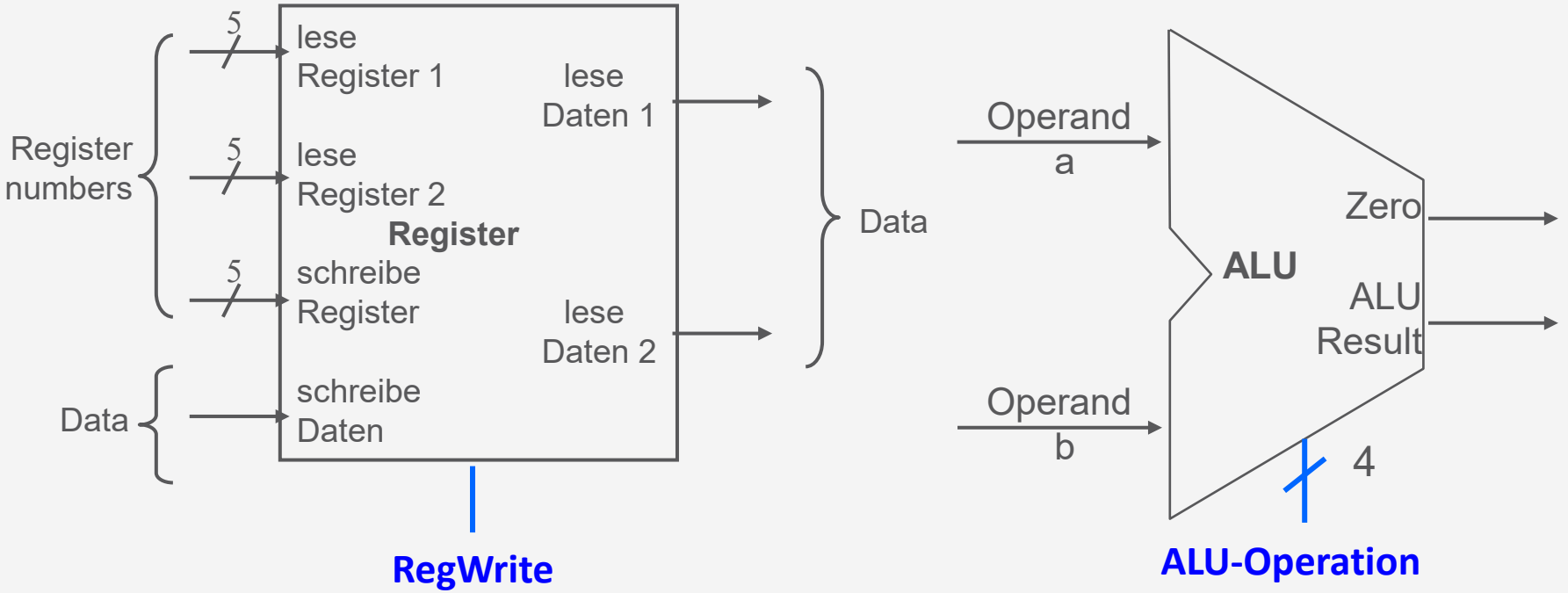
- Was brauchen wir (zusätzlich) um R-Typ Befehle zu realisieren?



Steuersignale (bisher)

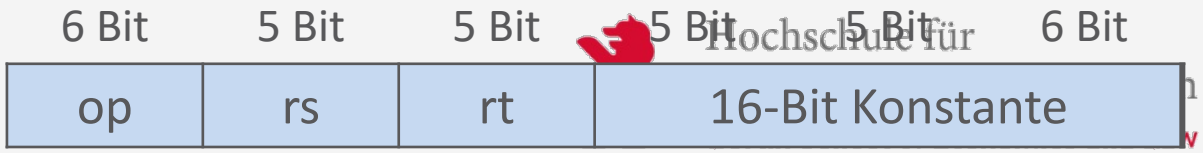
Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Registerspeicher hat das Steuersignal RegWrite
- ALU hat 4 Steuersignale, die die ALU-Operation spezifizieren



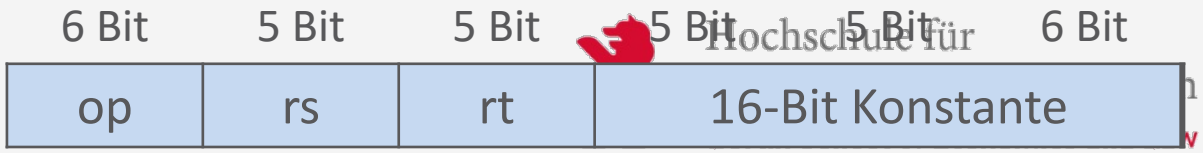
Operation	Funktion
0000	AND
0001	OR
0010	ADD
0110	SUB
0111	SLT
1100	NOR

I-Typ Befehle

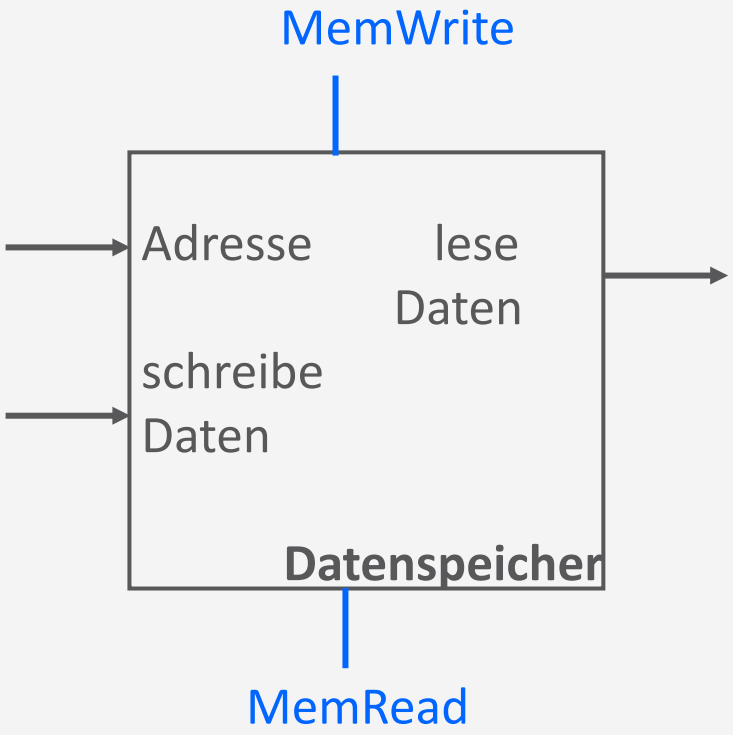


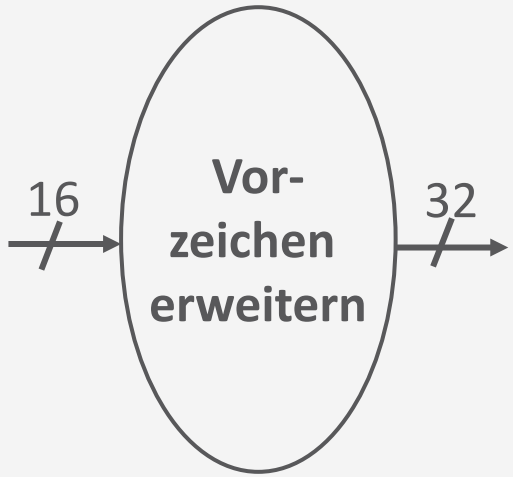
- `lw` und `sw` verwenden die ALU um die Adresse zu berechnen.
- **Was brauchen wir noch zusätzlich zum Registersatz und zur ALU?**

I-Typ Befehle



- `lw` und `sw` verwenden die ALU um die Adresse zu berechnen.
- **Was brauchen wir noch zusätzlich zum Registersatz und zur ALU?**



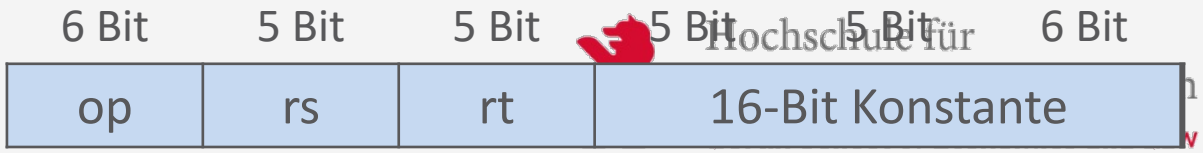


Vorzeichenerweiterung:

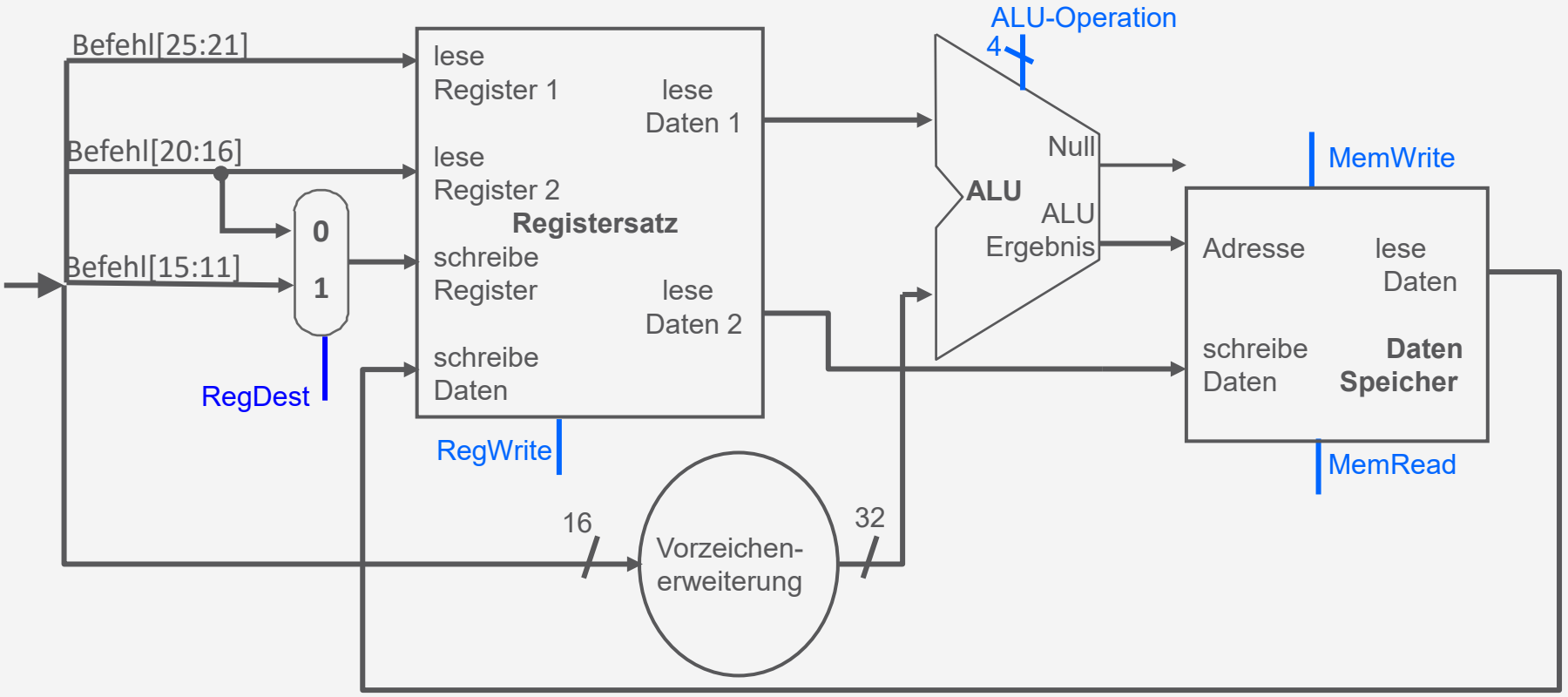
FF FF wird zu FF FF FF FF

7F FF wird zu 00 00 7F FF

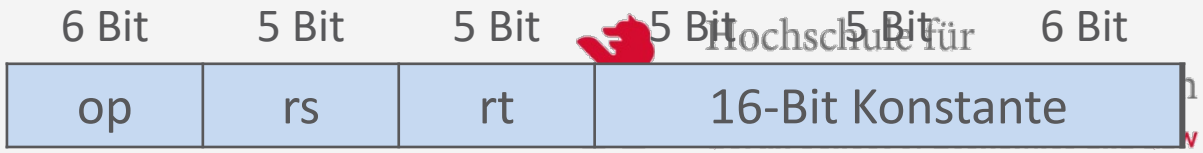
Speicherzugriffe



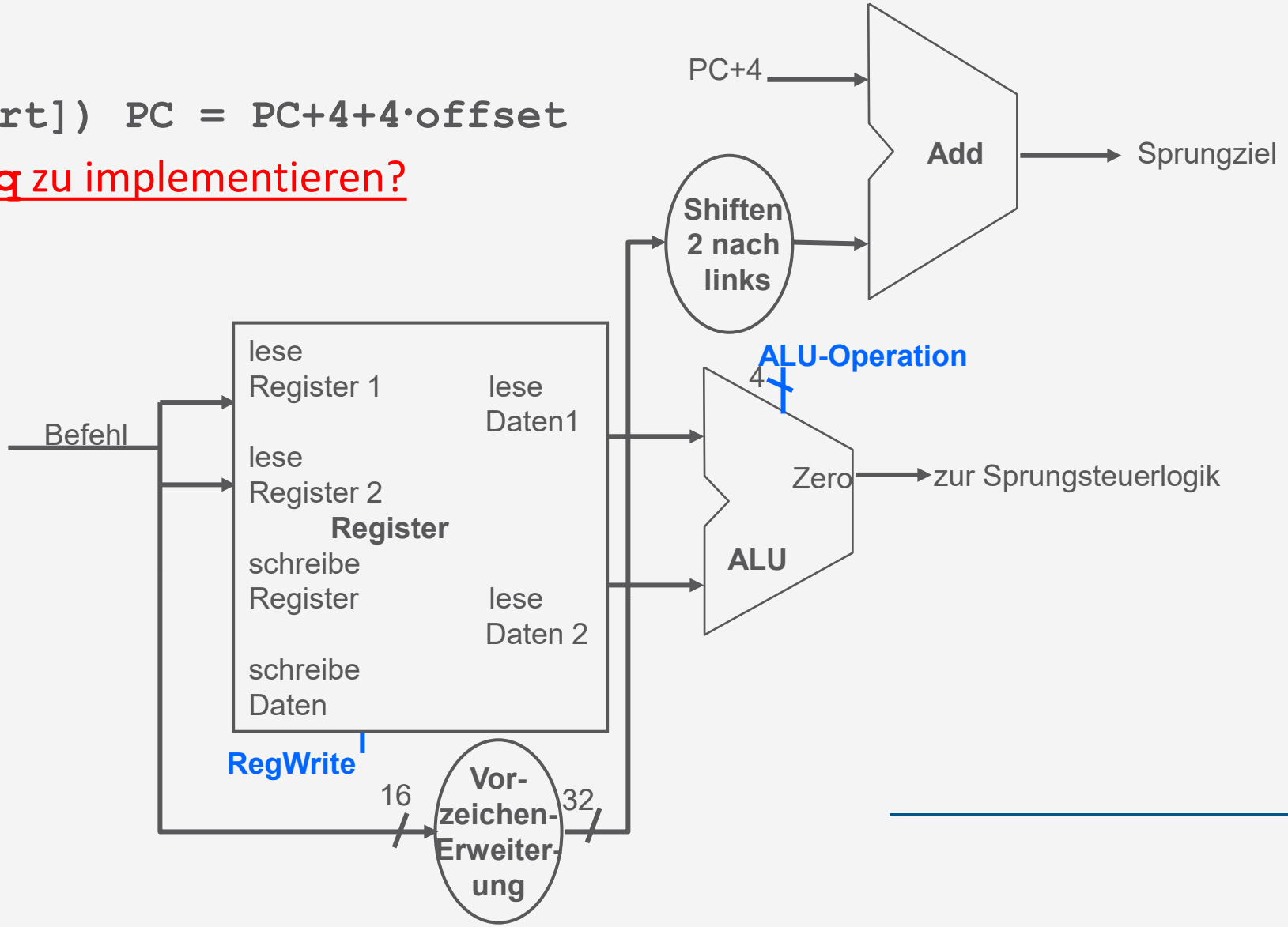
- Teil des Datenpfads für Lade- und Speicherbefehle:



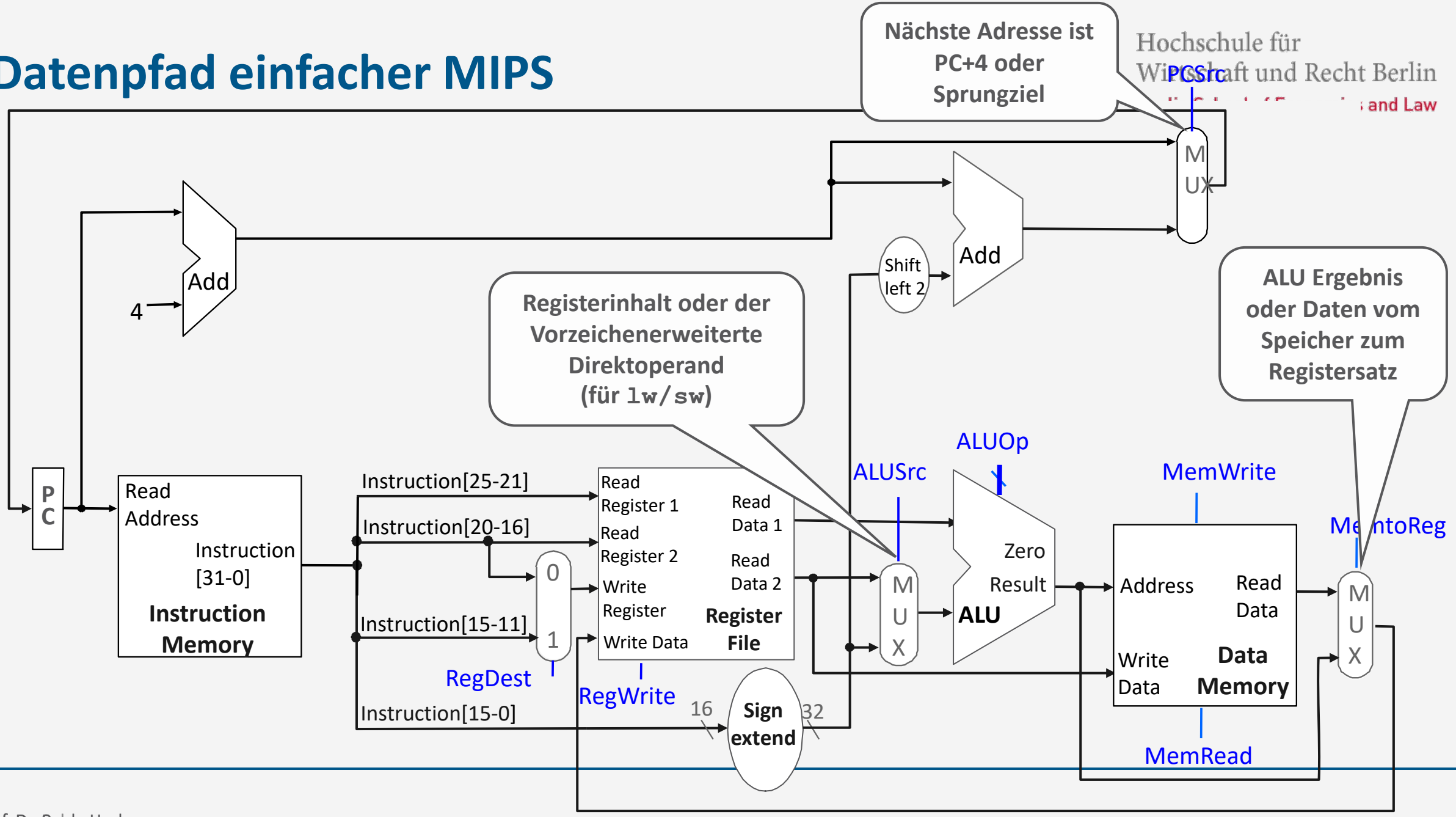
Verzweigung



- `beq rs,rt,offset`
  `# if (Reg[rs]==Reg[rt]) PC = PC+4+4·offset`
- Was brauchen wir um **`beq`** zu implementieren?



Datenpfad einfacher MIPS

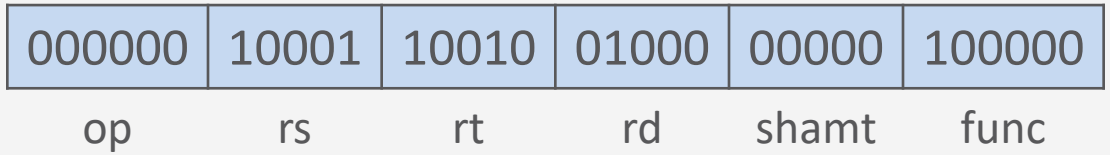


Überblick

1. Einleitung
2. Vorbereitung: ALU-Erweiterung für `slt` und `beq`
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

Steuerung

- Wählt die auszuführende Operation der ALU
- entscheidet, welche Register geschrieben werden
- Kontrolliert den Datenfluss (Multiplexer)
- Die Information kommt von den obersten und den untersten 6 Bit des Befehls (Opcode *op* und Funktionscode *func*)
- Beispiel: **`add $8, $17, $18`**
  - Befehlsformat:



- Zuerst legen wir die ALU-Steuerung fest und dann die Hauptsteuereinheit

ALU-Steuerung

- Was muss die ALU machen für
  - Lade/Speicher-Befehle?
  - `beq`?
  - R-Typ Befehle?

ALU-Steuerung

- Was muss die ALU machen für
  - Lade/Speicher-Befehle?
  - `beq`?
  - R-Typ Befehle?

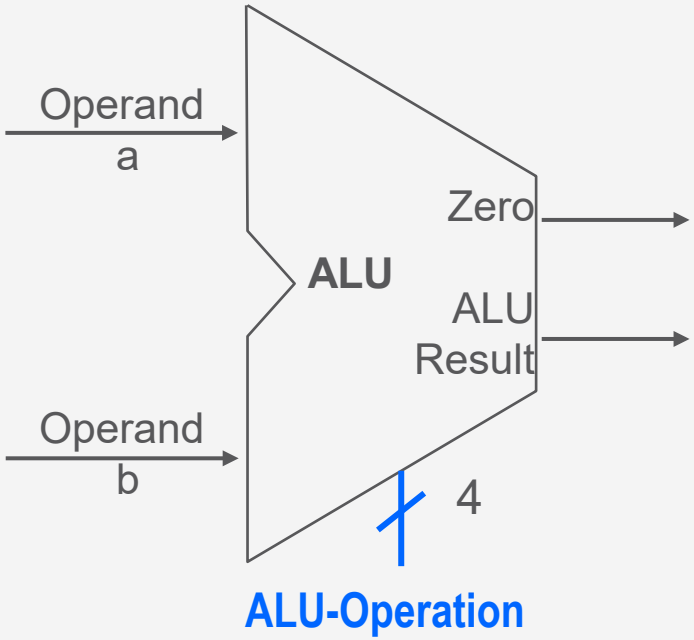
➢ addieren

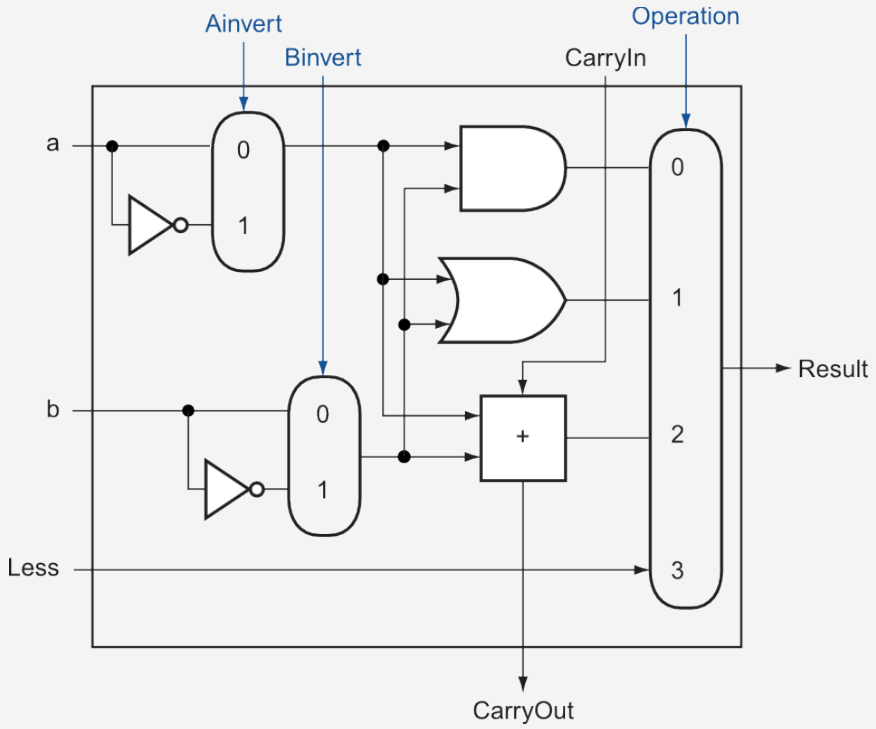
➢ subtrahieren

➢ hängt vom Funktionscode ab

ALU Steuersignale

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

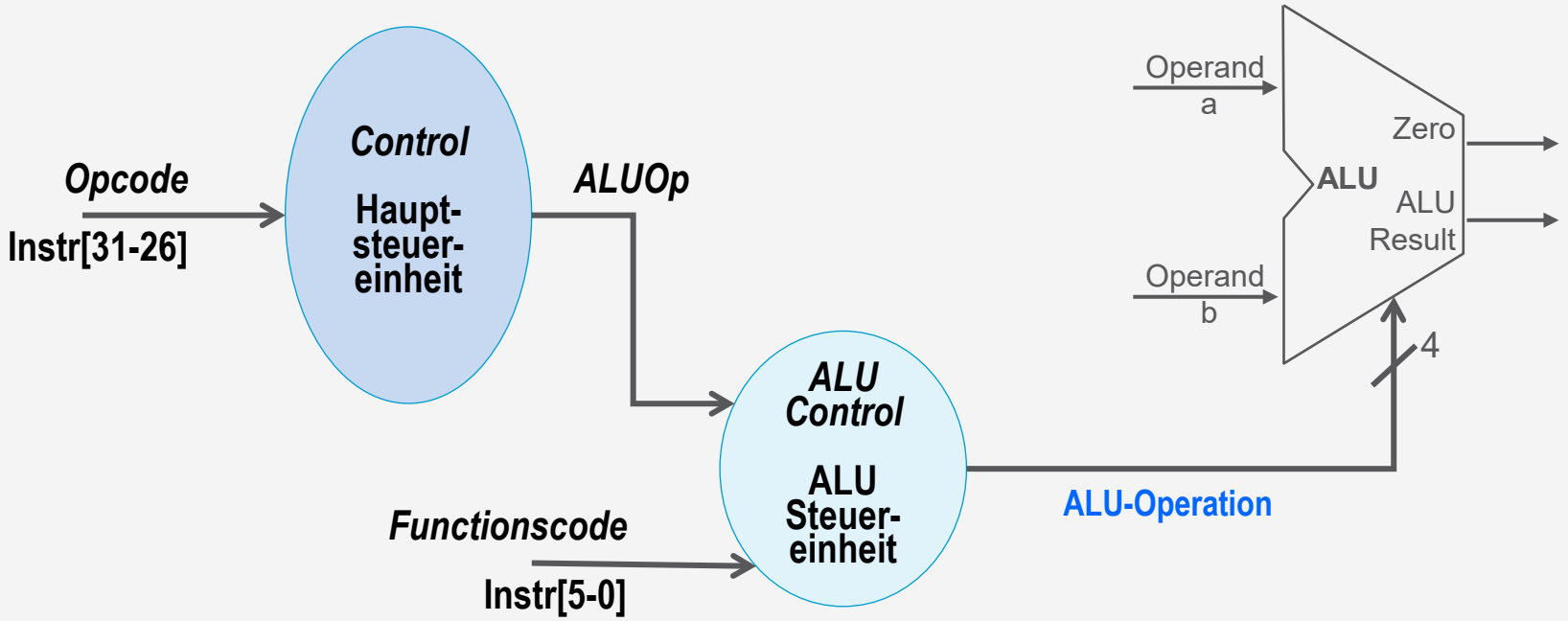




ALU Operation	Steuer-signale
and	0000
or	0001
add	0010
sub	0110
slt	0111
nor	1100

ALU Steuereinheit

- Die 4 ALU Steuersignale werden aus Opcode und Funktionscode berechnet.
- Achtung: `beq`, `sw`, `lw` haben keinen Funktioncode
- Steuereinheit ***Control*** berechnet aus den 6 Bit des Opcode ein 2-Bit Steuersignal ALUOp, ***ALU Control*** berechnet ALU-Operation aus ALUOp und Funktionscode
  - Addition für `lw`, `sw`
    → ALUOp = 00
  - Subtraktion für `beq`
    → ALUOp = 01
  - Bestimmt durch Funktionscode
    → ALUOp = 10



Spezifikation ALU Steuereinheit

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

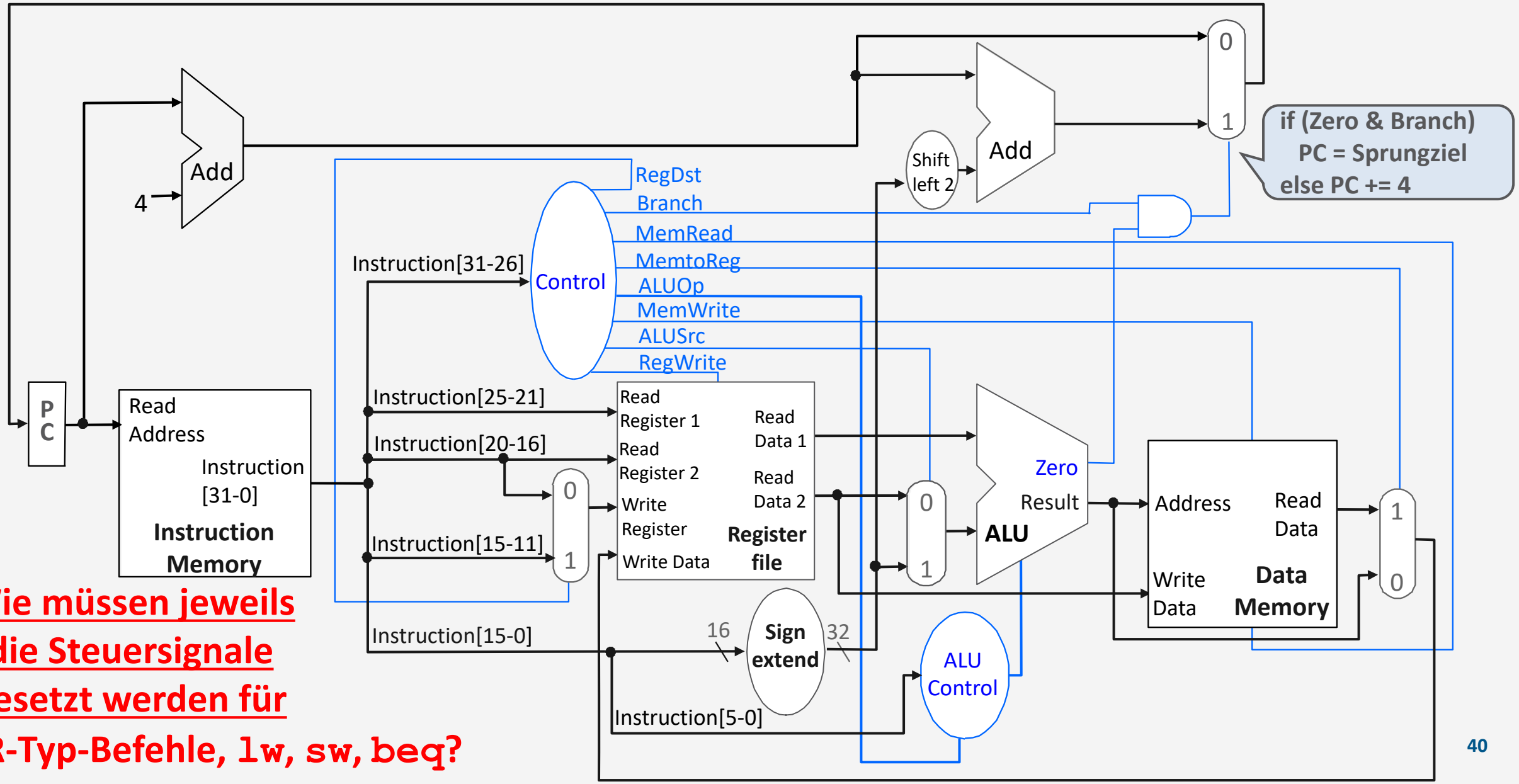
- ALU Steuereinheit kann durch eine Wahrheitstabelle spezifiziert werden:

opcode	funct	ALUOp		Funct field						ALU control signals	Desired ALU action
		ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0		
lw/sw	-	0	0	X	X	X	X	X	X	0010	add
beq	-	0	1	X	X	X	X	X	X	0110	subtract
R-type	add	1	0	X	X	0	0	0	0	0010	add
R-type	sub	1	0	X	X	0	0	1	0	0110	subtract
R-type	and	1	0	X	X	0	1	0	0	0000	and
R-type	or	1	0	X	X	0	1	0	1	0001	or
R-type	slt	1	0	X	X	1	0	1	0	0111	slt

- Ableitung des Schaltnetzes aus der Wahrheitstabelle: Übung für zu hause

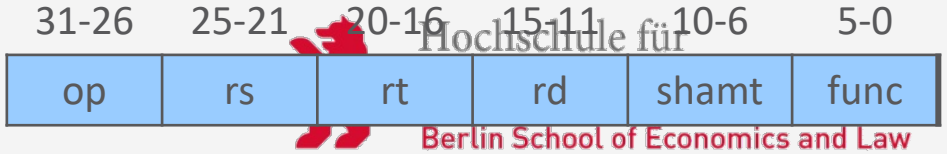
Datenpfad mit Steuerung

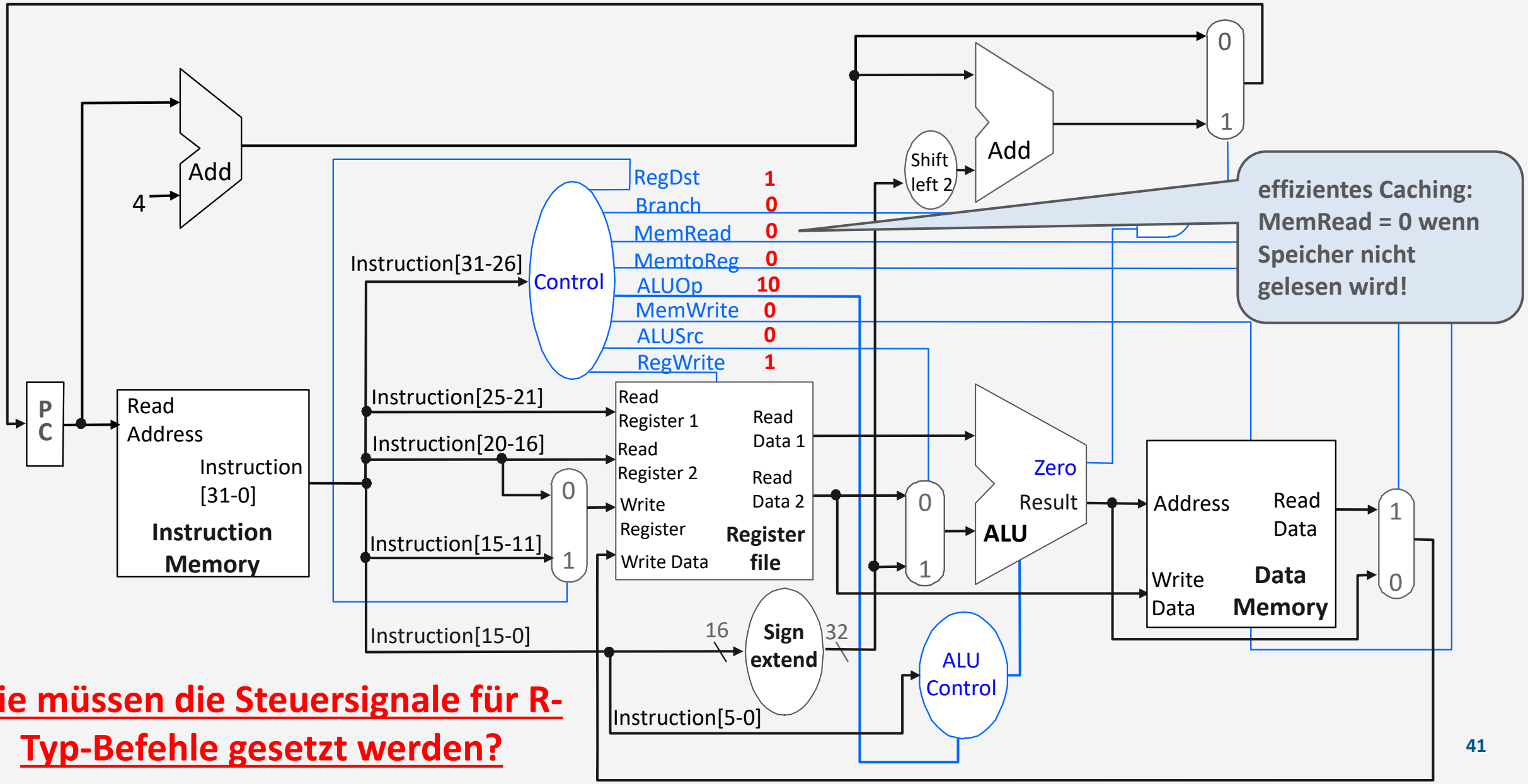
Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Wie müssen jeweils die Steuersignale gesetzt werden für R-Typ-Befehle, `lw`, `sw`, `beq`?

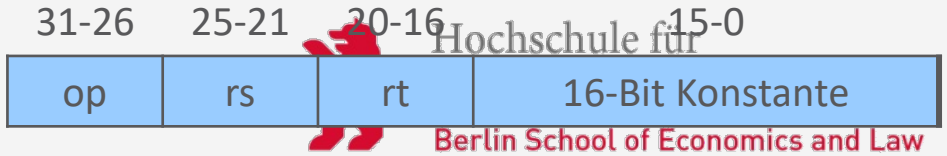
Steuersignale für R-Format

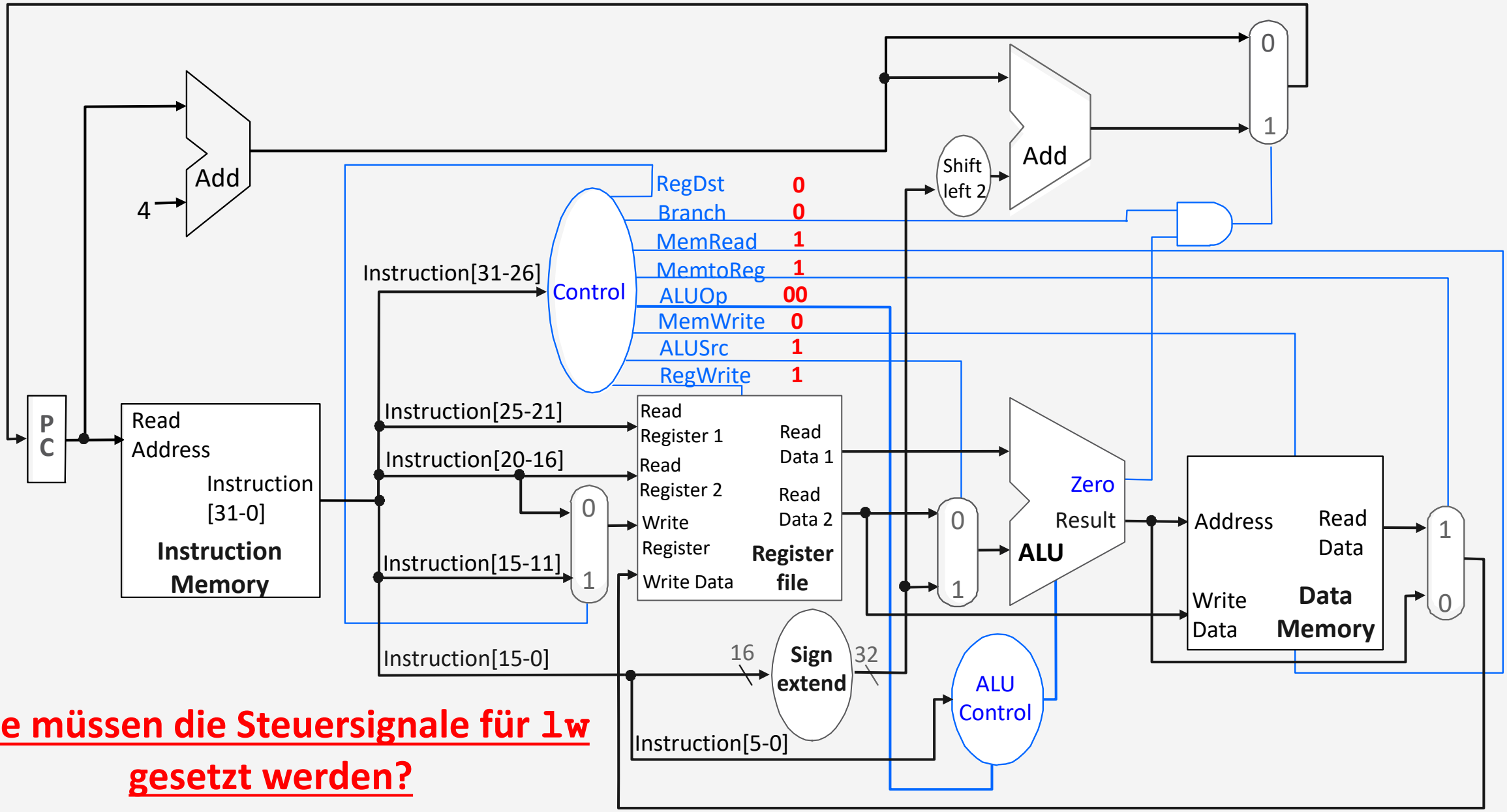




Wie müssen die Steuersignale für R-Typ-Befehle gesetzt werden?

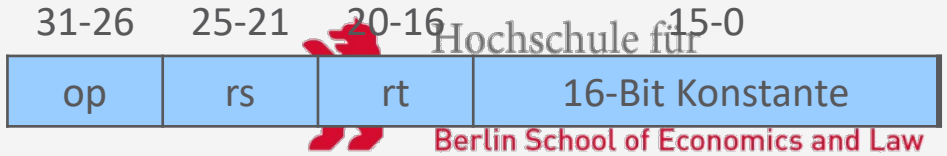
Steuersignale `lw`



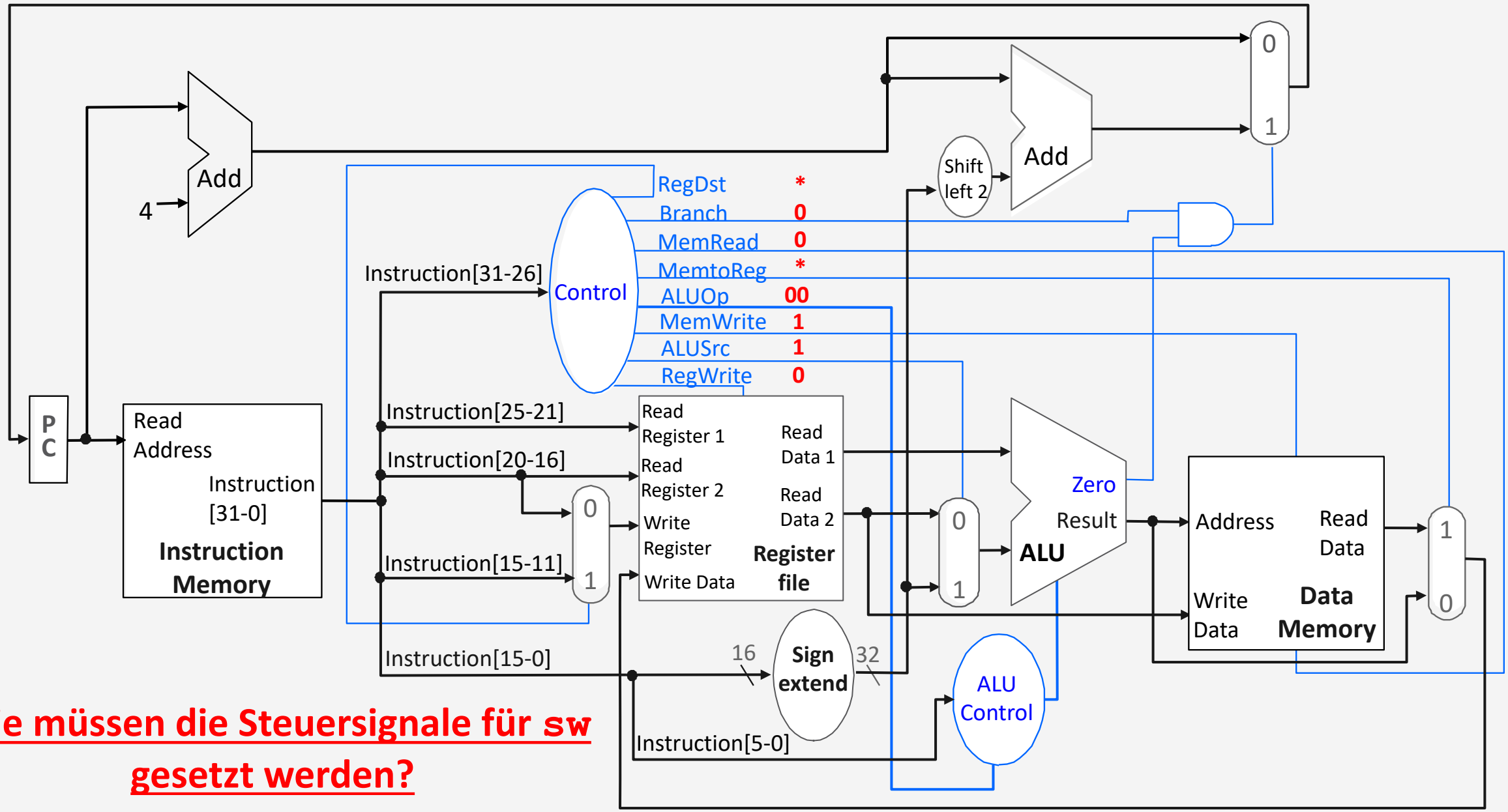


Wie müssen die Steuersignale für `lw` gesetzt werden?

Steuersignale sw

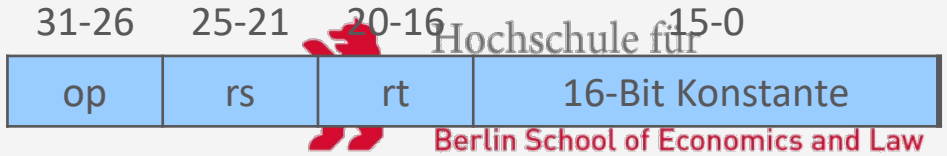


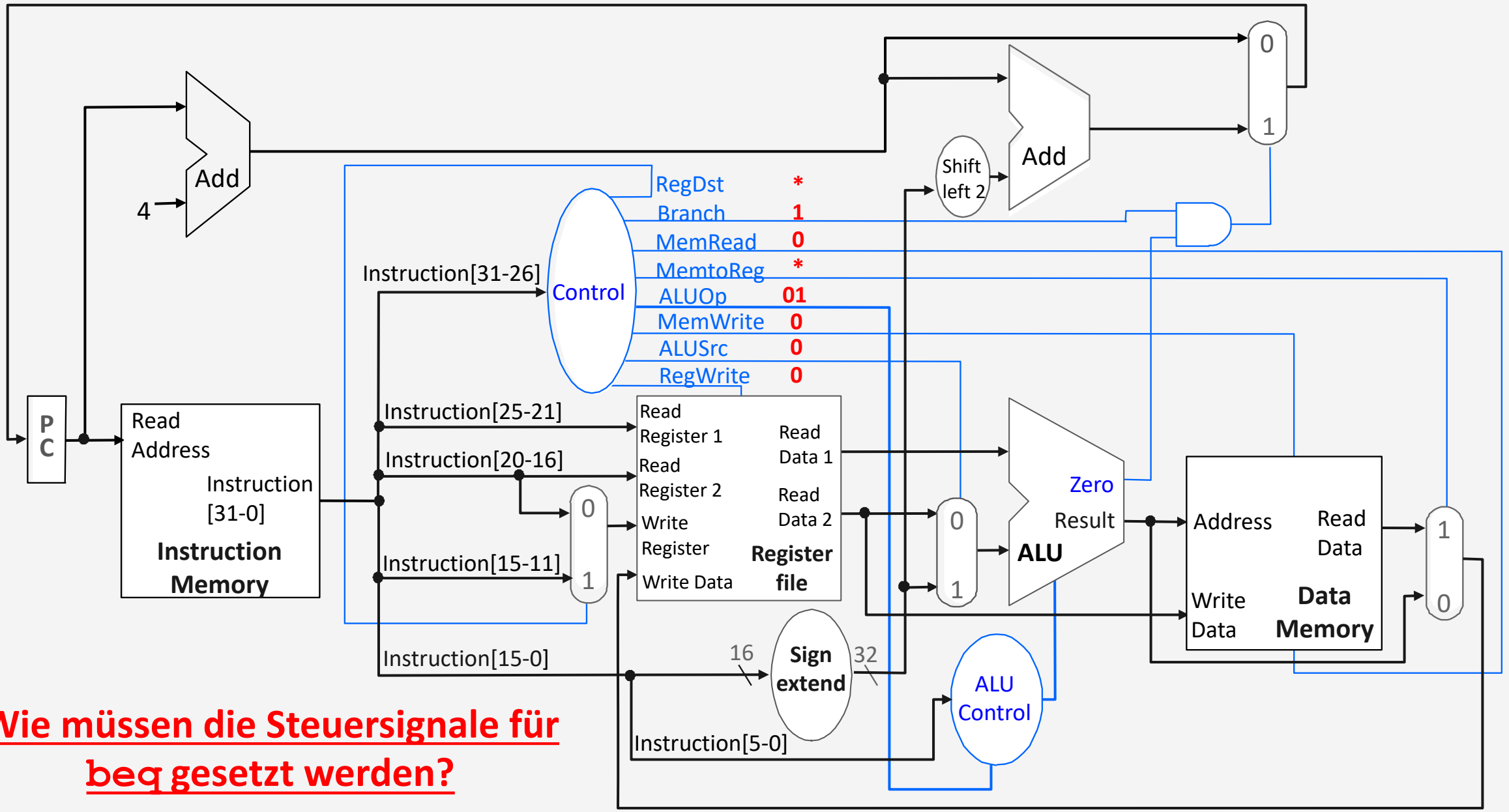
Hochschule für Berlin School of Economics and Law



Wie müssen die Steuersignale für sw gesetzt werden?

Steuersignale beq





Wie müssen die Steuersignale für beq gesetzt werden?

Spezifikation der Hauptsteuereinheit

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Belegung der Steuerleitungen wird ausschließlich durch *Opcode* bestimmt
  - R-Typ Befehle
  - `lw`
  - `sw`
  - `beq`

0	rs	rt	rd	shamt	func
35	rs	rt	16-Bit Konstante		
43	rs	rt	16-Bit Konstante		
4	rs	rt	16-Bit Konstante		

- Steuersignale:

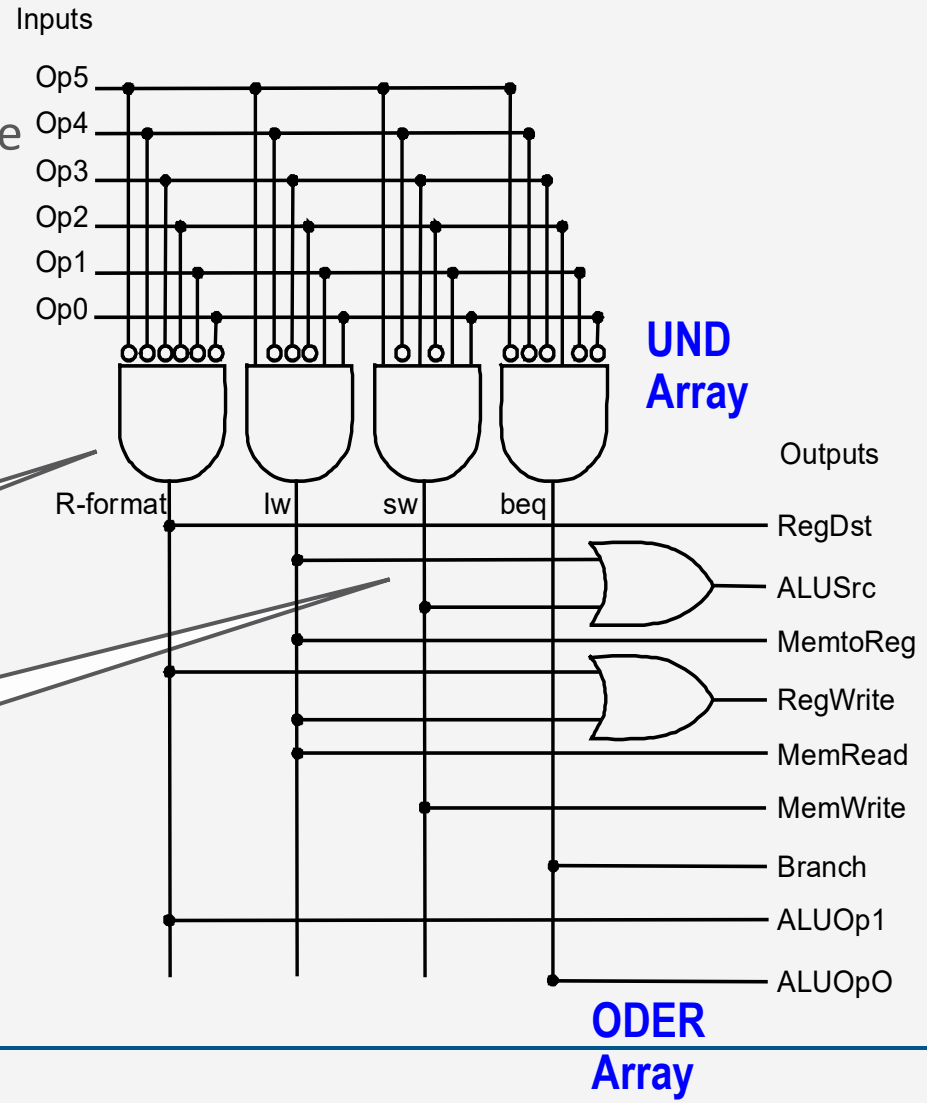
Befehl	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-Format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Mögliche Implementierung der Hauptsteuereinheit

- Tabelle kann auf eine *Programmable Logic Array* (PLA, programmierbare logische Anordnung) abgebildet werden, die die Steuersignale berechnet.

Befehl	Opcode	RegDst	ALUSrc	…
R-format	000000	1	0	.
lw	100011	0	1	.
sw	101011	X	1	.
beq	000100	X		.

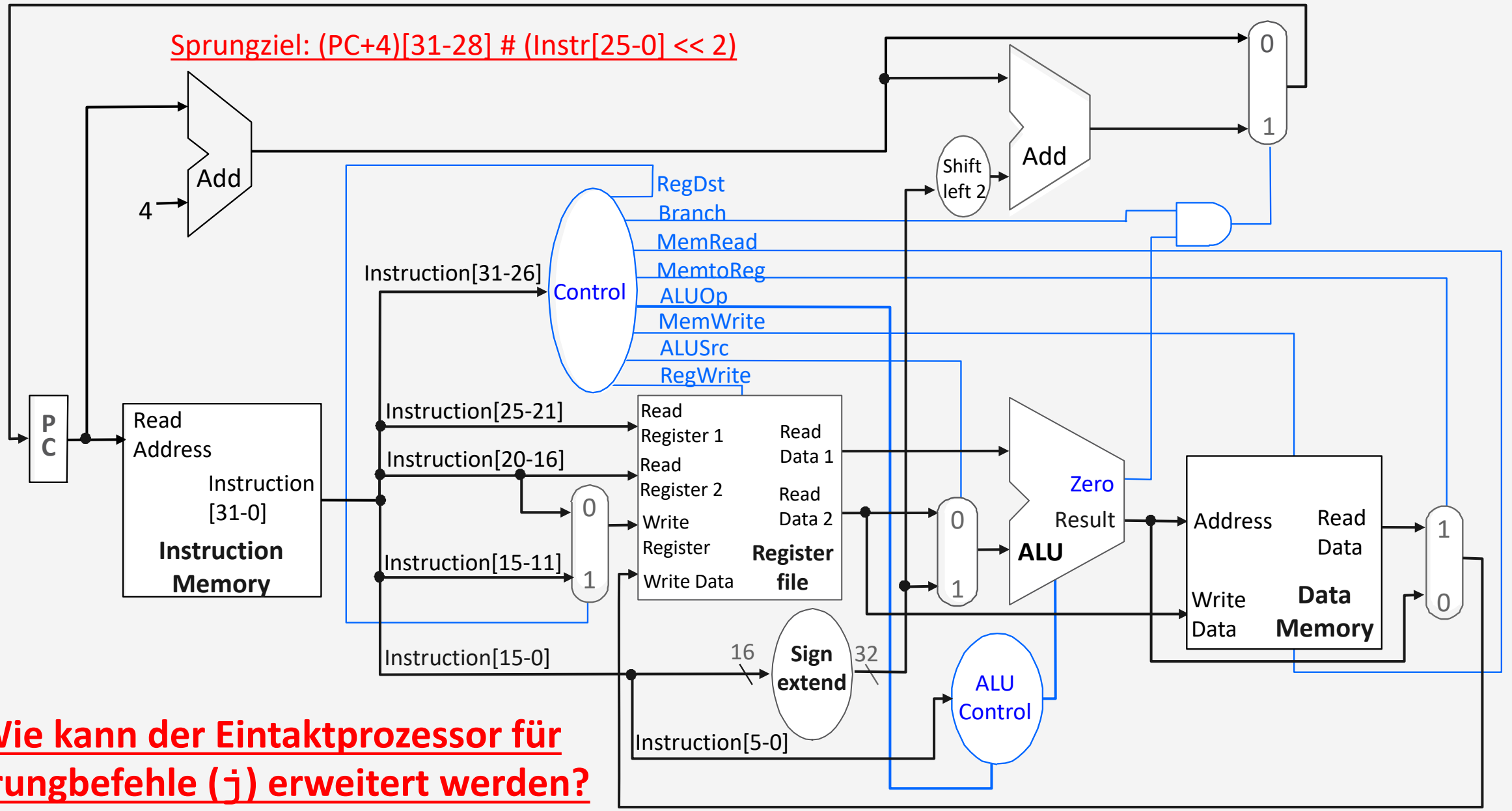
- Beispiele:
  - **RegDst=1 gdw Opcode=000000 (R-format)**
  - **ALUSrc=1 gdw Opcode=100011 (lw) or 101011 (sw)**



Sprungbefehl (j)

2	26-Bit Adresse

Hochschule für Wirtschaft und Recht
Berlin School of Economics and Law



Wie kann der Eintaktprozessor für Sprungbefehle (j) erweitert werden?

Sprungbefehl

- Wie kann der Eintaktprozessor erweitert werden, um den Sprungbefehl (`j`) zu implementieren?
- Sprung-Befehlsformat:
- Zieladresse ist eine Verknüpfung von:
  - Höchstwertigsten 4 Bits von PC+4
  - 26-Bit Adresse im Befehl
  - die Bits 00 (da Wortadresse)
- Was müssen wir können?
  - Zieladresse berechnen: (PC+4)[31-28] # (Instr[25-0] << 2)
  - Selektieren zwischen nächstem PC und Zieladresse

    ➢wird gesteuert von einem neuen Steuersignal

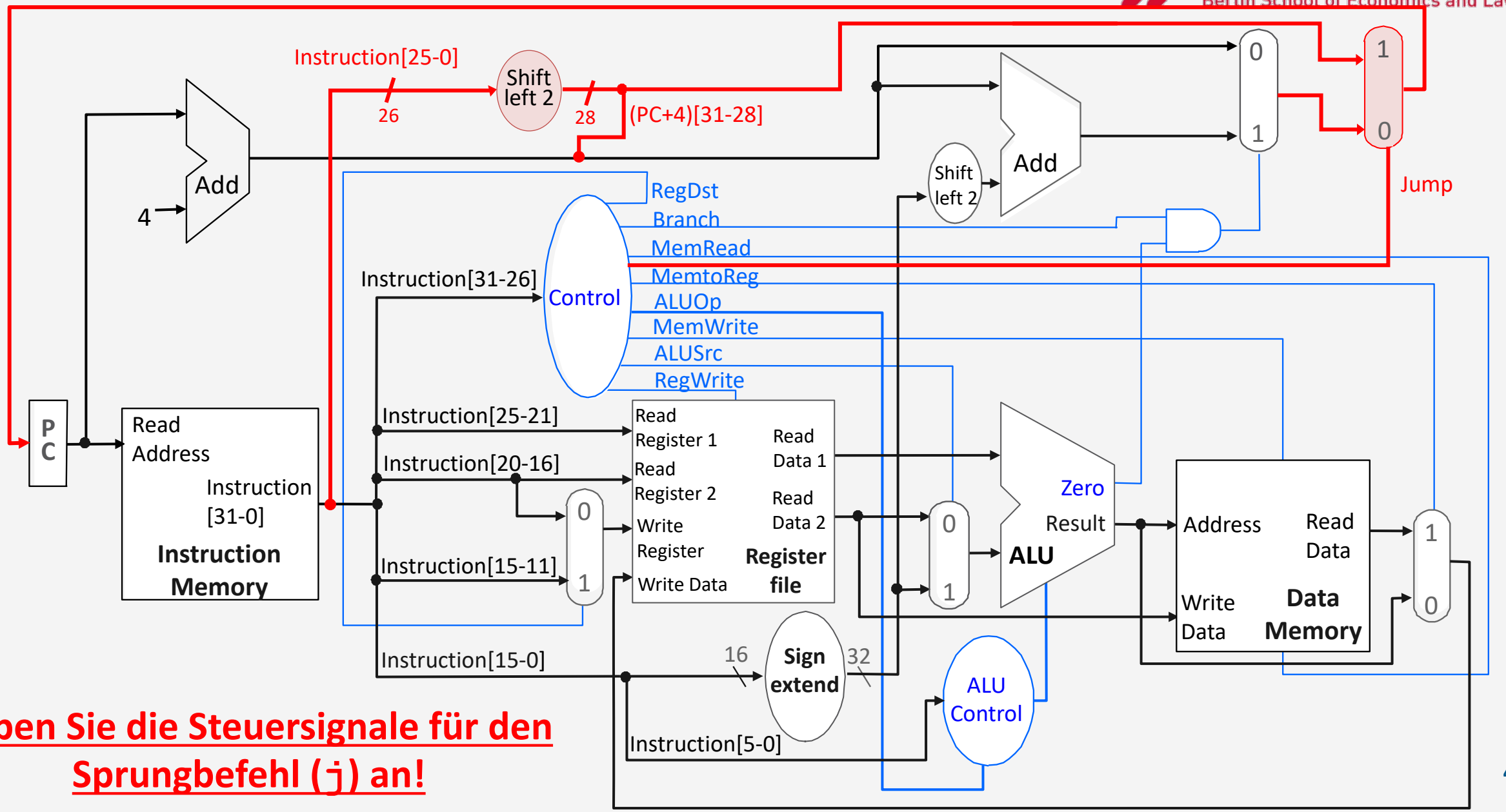
Verknüpfung der Leitungen

„2-Stellen-nach-links-Shifter“

MUX

Eintaktprozessor mit Jump

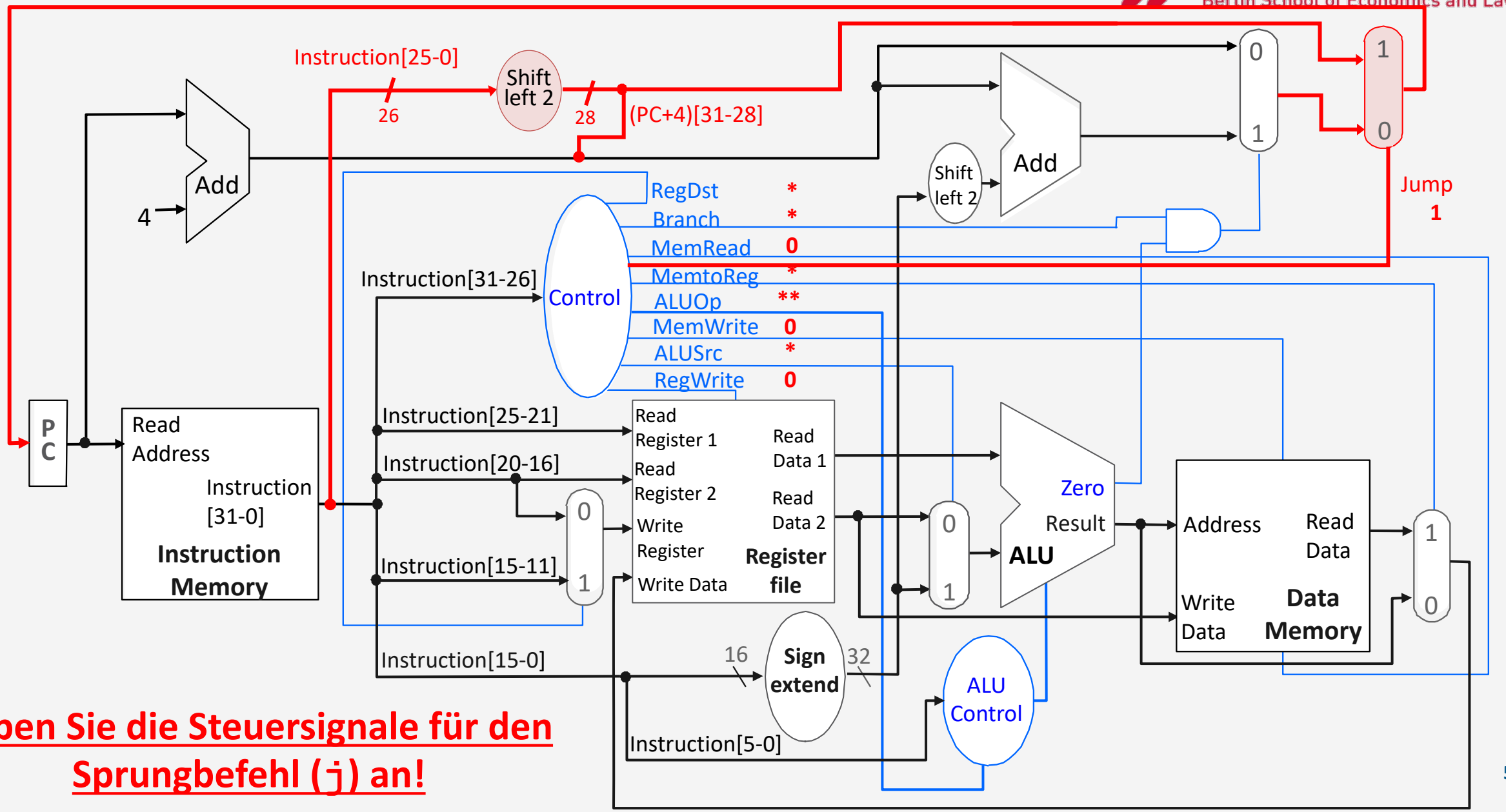
Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Geben Sie die Steuersignale für den Sprungbefehl (j) an!

Eintaktprozessor mit Jump

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Geben Sie die Steuersignale für den Sprungbefehl (j) an!

Steuersignale für Jump

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- RegDst=✶; da `j` nicht in den Registerspeicher schreibt, kann RegDst=✶ sein
- Branch=✶; für `j` wird der obere Eingang des neuen MUX verwendet
- RegWrite=0; `j` schreibt nicht in den Registerspeicher
- Jump=1; selbstverständlich
- MemRead=0; selbst wenn die geladenen Daten verworfen werden, soll kein *cache miss* oder *page fault* provoziert werden
- Mem2Reg= ✶; `j` schreibt nicht in den Registerspeicher
- ALUOp=✶✶; Ergebnis der ALU wird verworfen, Operation unerheblich
- MemWrite=0; Befehl sollte nicht in den Speicher schreiben (Achtung: auch wenn die ALU-Operation außer Acht gelassen wird, wird von ihr ein Ergebnis produziert)
- ALUSrc=✶; Ergebnis der ALU wird verworfen, ALU Operation und Input irrelevant
- RegWrite=0; `j` schreibt nicht in den Registerspeicher

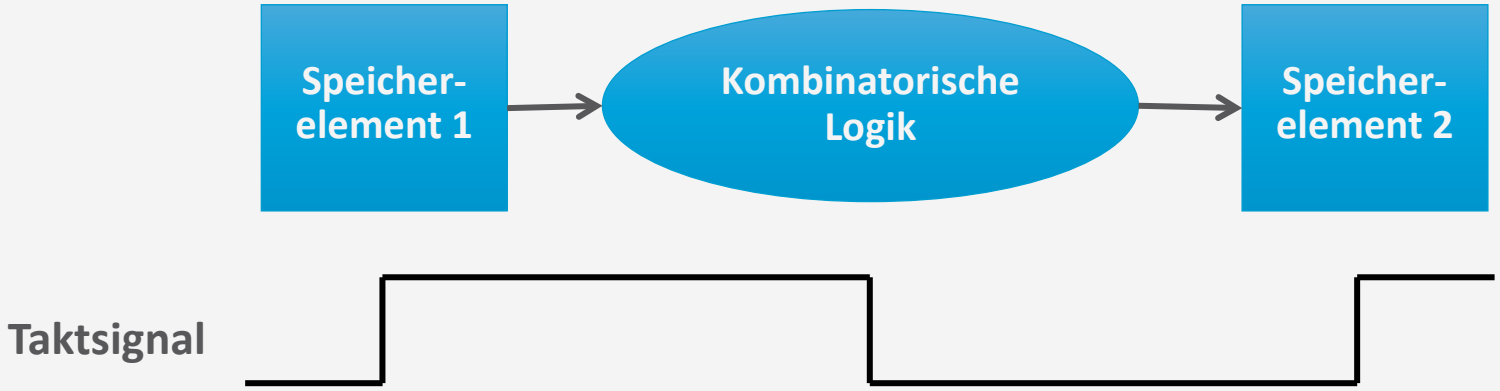
Überblick

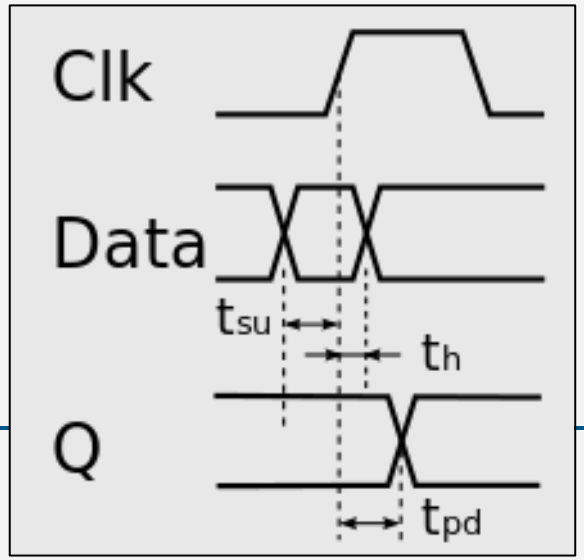
1. Einleitung
2. Vorbereitung: ALU-Erweiterung für `slt` und `beq`
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

Zeitverhalten des Eintaktprozessors

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

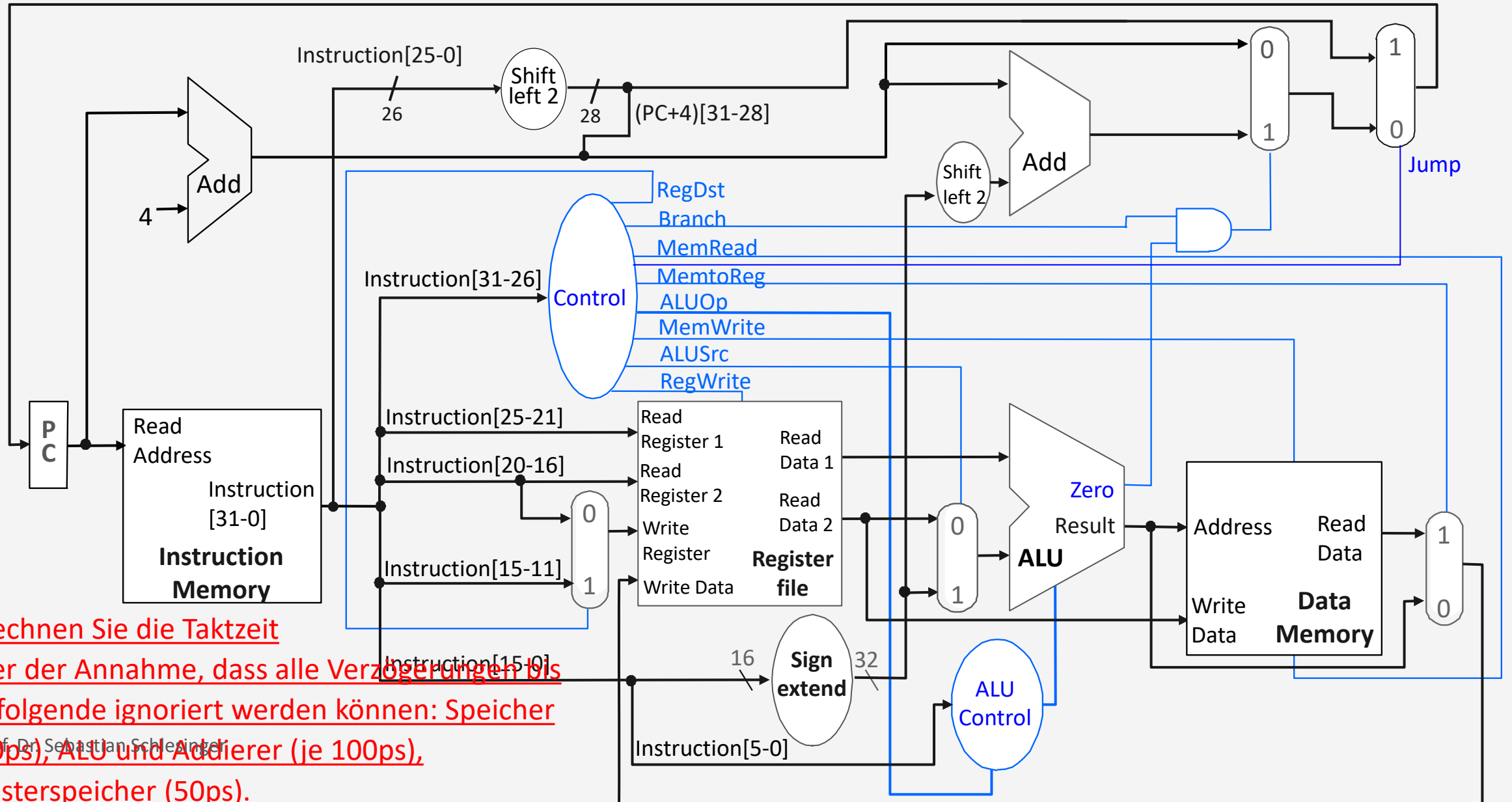
- alle kombinatorischen Schaltungen (Steuerung, ALU, Vorzeichenerweiterung, …) brauchen Zeit, bis sich das Ergebnis am Ausgang stabilisiert hat
- Schreib-Signale bestimmen, ob geschrieben wird
- Taktsignal (*Clock*) bestimmt, wann geschrieben wird
- Taktzeit wird durch den längsten Pfad der kombinatorischen Logik bestimmt
- wir vernachlässigen *setup time* t_{su}, *hold time* t_h und *propagation delay* t_{pd}





Taktzeit der Eintakt-Implementierung

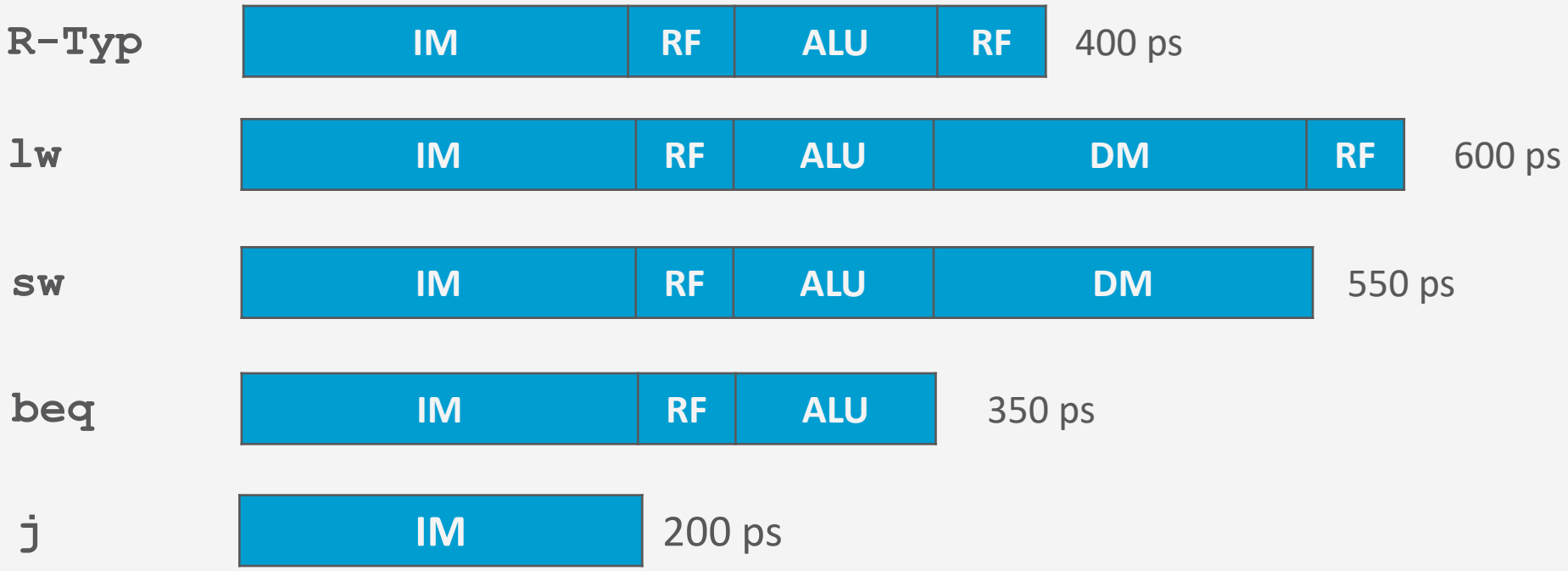
Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Berechnen Sie die Taktzeit unter der Annahme, dass alle Verzögerungen bis auf folgende ignoriert werden können: Speicher (200ps), ALU und Addierer (je 100ps), Registerspeicher (50ps).

Taktzeit der Eintakt-Implementierung

- Speicher (200ps), ALU & Addierer (100ps), Registerspeicher (50ps)



  - langsamster Befehl ist **`lw`** mit 600 ps

Leistung des Eintaktprozessors

- Bewertung der Leistung: Gedankenexperiment
- Wenn eine variable Taktzeit möglich wäre, würden die Instruktionen folgende Zeiten benötigen:
  - R-type: 400 ps, Load: 600 ps, Store: 550 ps, Branch: 350 ps, Jump: 200 ps
- Gegeben sei der folgende Instruktionsmix:
  - 25% loads, 10% stores, 45% R-type, 15% branches, 5% jumps
- **Wie hoch ist die „durchschnittliche Befehlszeit" mit a) fester oder b) variabler Taktzeit?**

Leistung des Eintaktprozessors

- Bewertung der Leistung: Gedankenexperiment
- Wenn eine variable Taktzeit möglich wäre, würden die Instruktionen folgende Zeiten benötigen:
  - R-type: 400 ps, Load: 600 ps, Store: 550 ps, Branch: 350 ps, Jump: 200 ps
- Gegeben sei der folgende Instruktionsmix:
  - 25% loads, 10% stores, 45% R-type, 15% branches, 5% jumps
- **Wie hoch ist die „durchschnittliche Befehlszeit" mit a) fester oder b) variabler Taktzeit?**

a) DB_{fest} = 600 ps

b) $DB_{var} = 0.25 \cdot 600 + 0.1 \cdot 550 + 0.45 \cdot 400 + 0.15 \cdot 350 + 0.05 \cdot 200 =$ 447.5 ps

- Implementierung mit variabler Taktzeit wäre 600/447.5 = 1.34 Mal schneller

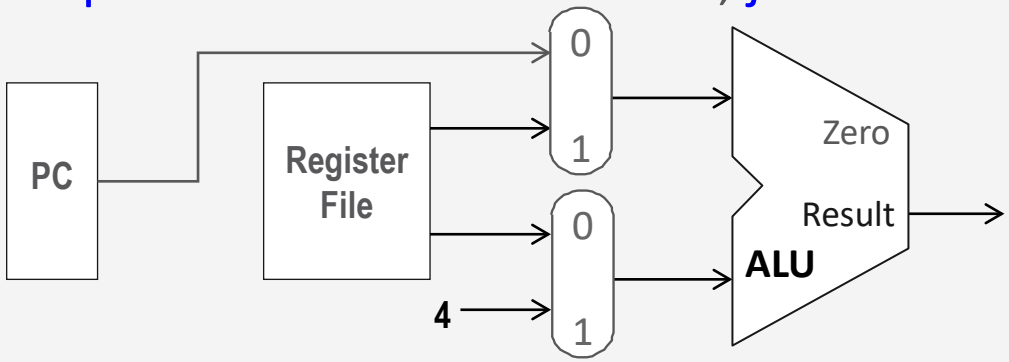
➢ theoretischer (!) Performanzverlust von 34 % durch die Eintakt-Implementierung

Nachteile der Eintaktausführung

- Probleme des Eintaktprozessors
  - Komplexere Befehle wie Gleitkomma-Instruktionen würden Taktzeit noch weiter erhöhen
  - Verschwendung von Chipfläche: Hardware-Ressourcen (z. B. ALU + Addierer) müssten repliziert werden
- Mögliche Lösung
  - Kürzere Taktzeit (aber wie?)
  - Unterschiedliche Befehle dauern unterschiedliche Anzahl von Taktzyklen

Eine mögliche Lösungsidee

- kürzere Taktzeit wählen
- verschiedene Befehle mit unterschiedlicher Anzahl von „Schritten“
  - ➢ jeder Schritt 1 Takt.
- Beispiel:
  - Schritt 1: Lese Befehl aus dem Speicher und ALU inkrementiert PC
  - Schritt 2: Lese Registerdatei und ALU berechnet Sprungadresse
  - Schritt 3: . . .
- Eine Einheit kann mehrmals pro Befehl benutzt werden, jedoch zu verschiedenen Takten.



Überblick

1. Einleitung
2. Vorbereitung: ALU-Erweiterung für `slt` und `beq`
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

Was sind die Schritte?

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Welche Schritte werden jeweils für einen Befehl benötigt? Was geschieht genau in den einzelnen Schritten?

Was sind die Schritte?

Welche Schritte werden jeweils für einen Befehl benötigt? Was geschieht genau in den einzelnen Schritten?

- Ausführung auf Eintaktprozessor:

R-type	IM	RF	ALU	RF	
`lw`	IM	RF	ALU	DM	RF
`sw`	IM	RF	ALU	DM	
`beq`	IM	RF	ALU		
`j`	IM				

R-Type	Befehl Holen	Reg. lesen	Ausführung	Register schreiben	
`lw,sw`			Adress-berechnung	Speicherzugriff	Register schreiben
`beq`			Sprung		

Register-Transfer-Ebene

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Register-Transfer-Ebene (*Register Transfer Level*, RTL)
  - sehr wichtige Abstraktionsebene in der Modellierung von integrierten Schaltkreisen.
  - Beim Entwurf auf dieser Ebene wird das System durch die Transfers zwischen den Registern spezifiziert.
  - Register können i. Allg. sowohl für (Assembly)-Programmierer sichtbare Register (wie `$0`, `$1`, ...) als auch Prozessor-interne Register (PC, ALUOut, ...) sein
  - Schreibweise (Mikrobefehle):

```
PC <= PC + 4
A <= Reg[Addr]
B <= Memory[Addr]
```

Befehl holen und dekodieren

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

1. Instruction Fetch

- PC als Speicheradresse um nächsten Befehl zu holen
- wichtig: den Befehl müssen wir jetzt zwischenspeichern, um ihn im nächsten Takt weiter verwenden zu können ⇨ Instruktionsregister **`IR`**
- außerdem können wir jetzt schon PC mit 4 inkrementieren

```
IR <= Memory[PC];
PC <= PC + 4;
```

2. Instruction Decode / Register Fetch

- Lese Register **`rs`** und **`rt`**, falls benötigt

```
A <= Reg[IR[25:21]]; B <= Reg[IR[20:16]];
```

- Berechne Sprungadresse, falls benötigt

```
ALUOut <= PC + (sign-extend(IR[15:0])<<2);
```

Befehl ausführen

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

<table>
<tr><td>R-Type</td><td rowspan="3">Befehl Holen</td><td rowspan="3">Reg. lesen</td><td>Ausführung</td><td>Register schreiben</td><td></td></tr>
<tr><td><code>lw,sw</code></td><td>Adress-berechnung</td><td>Speicherzugriff</td><td>Register schreiben</td></tr>
<tr><td><code>beq</code></td><td>Sprung</td><td></td><td></td></tr>
</table>

3. Execute

- Berechnung der Speicheradresse für Lade- und Speicherbefehle:

```
ALUOut <= A + sign-extend(IR[15:0]);
```

- R-type:

```
ALUOut <= A op B;
```

- Branch:

```
if (A==B) PC <= ALUOut;
```

  - Sprungzieladresse in ALUOut wurde im vorigen Schritt berechnet

- Jump: `PC <= {PC[31:28], (IR[25:0]<<2)}`

Speicherzugriffe und Ergebnisse schreiben

4. Memory Access / Write Back

- Register für **`lw`** um geladene Daten zwischenzuspeichern: **`MDR`** (Memory Data Register)

  `lw`: **`MDR <= Memory[ALUOut];`**
- **`sw`**: **`Memory[ALUOut] <= B;`**
- R-Befehle werden abgeschlossen: **`Reg[IR[15-11]] <= ALUOut;`**

5. Write Back

- Ladebefehle werden abgeschlossen: **`Reg[IR[20-16]] <= MDR;`**

Überblick

1. Einleitung
2. Vorbereitung: ALU-Erweiterung für `slt` und `beq`
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

Entwurf des Datenpfades

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Datenpfad: Umsetzung der Mikrobefehle in Hardware
- Blockschaltbild: benötigte Hardware (Register, Speicher, ALU, …) und deren Verbindungen
- Multiplexer fügen die Teile des Datenpfades zusammen und ermöglichen Steuerung, welcher Teil ausgeführt werden soll

Entwerfen Sie den Datenpfad für den Mehrzyklenprozessor!

Hinweise:

1. Blättern Sie in den Folien gern noch mal zurück, um sich die einzelnen Schritte (und ggf. den Datenpfad des Eintaktprozessors) nochmal anzuschauen.
2. Setzen Sie zunächst den 1. Schritt (Instruction Fetch) um und erweitern Sie das Blockschaltbild schrittweise.
3. Setzen Sie Hardware sparsam ein! Insbesondere soll hier nur ein Speicher für Befehle und Daten sowie nur eine ALU (und keine zusätzlichen Addierer) verwendet werden.

Befehl holen

1. Instruction Fetch

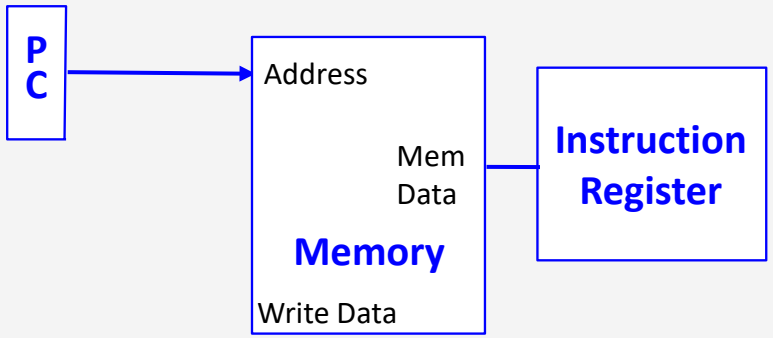
- PC am Speicher anlegen, um Befehl ins Instruktionsregister zu speichern
- Inkrementiere PC mit 4 und speichere Ergebnis zurück in den PC

```
IR <= Memory[PC];
PC <= PC + 4;
```

Befehl holen und PC erhöhen

1. Instruction Fetch: Befehl holen: **`IR <= Memory[PC];`**

PC erhöhen: **`PC <= PC + 4;`**

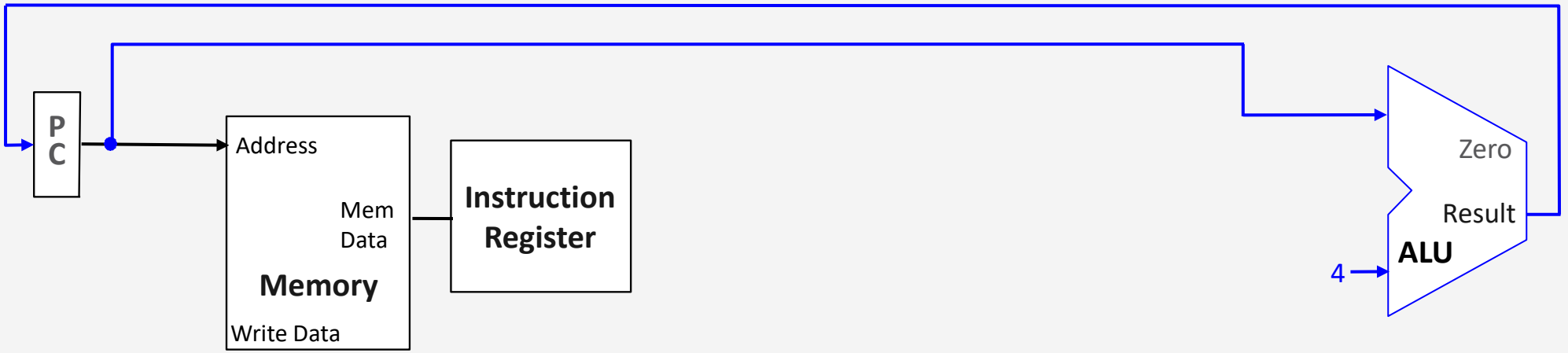


Befehl holen und PC erhöhen

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

1. Instruction Fetch: Befehl holen: `IR <= Memory[PC];`

PC erhöhen: `PC <= PC + 4;`



Befehl dekodieren und Register lesen

2. Instruction Decode

- Lese Register **`rs`** und **`rt`**, falls benötigt

```
A <= Reg[IR[25:21]]; B <= Reg[IR[20:16]];
```

- Berechne Sprungadresse, falls benötigt

```
ALUOut <= PC + (sign-extend(IR[15:0])<<2);
```

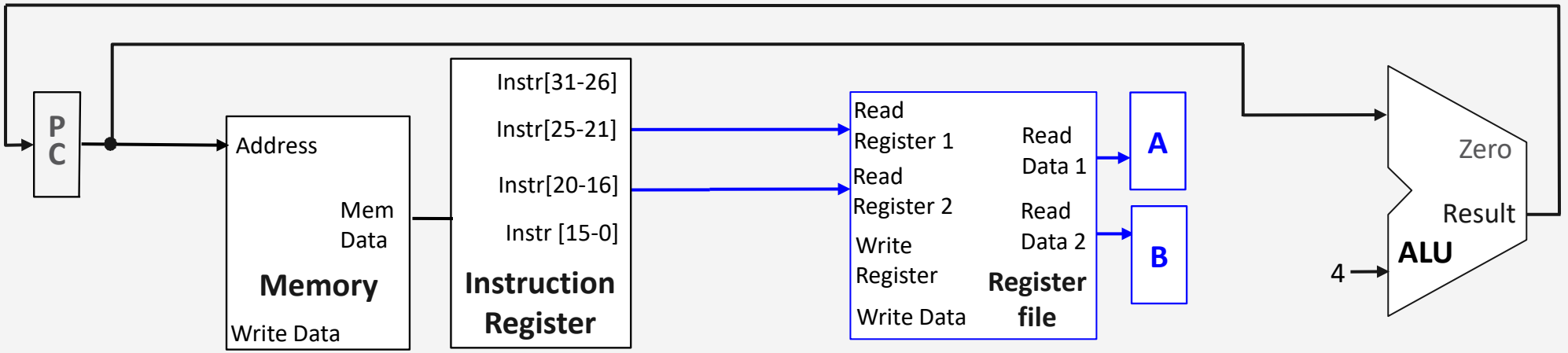
Befehl dekodieren und Register lesen

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

2. Instruction Decode:

Register lesen: `A <= Reg[IR[25:21]]; B <= Reg[IR[20:16]];`

Sprungziel berechnen: `ALUOut <= PC + (sign-extend(IR[15:0])<<2);`



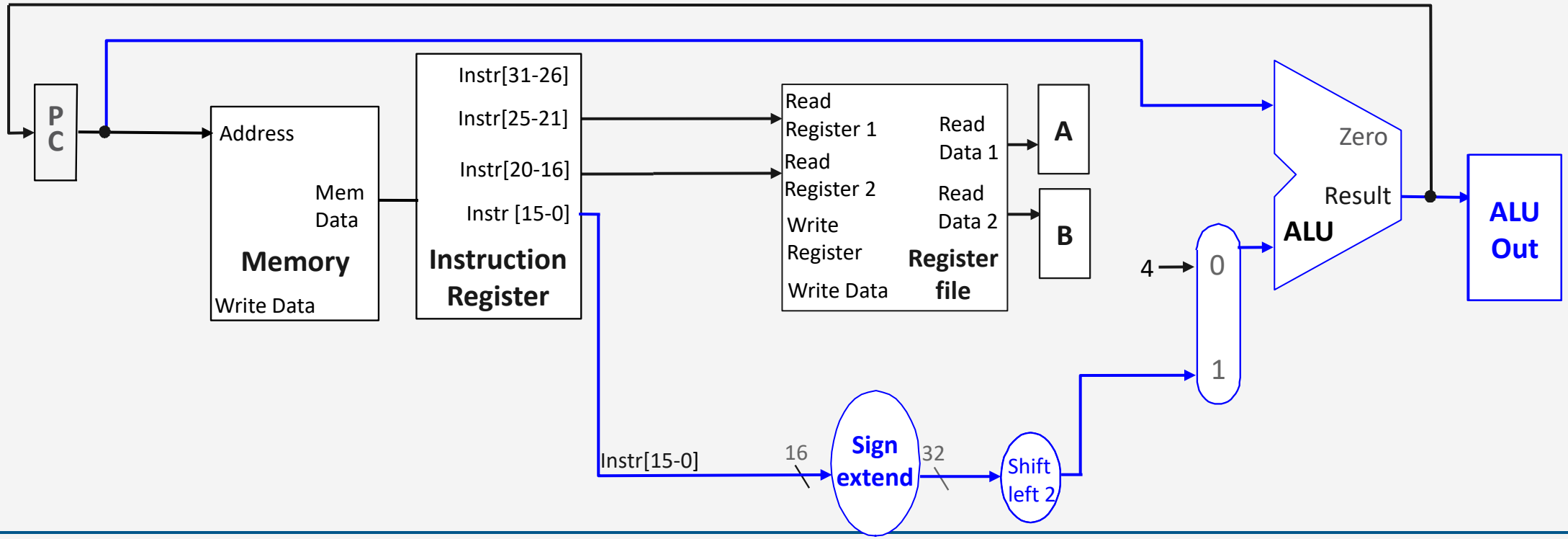
Befehl dekodieren und Register lesen

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

2. Instruction Decode (**beq**):

Register lesen: `A <= Reg[IR[25:21]]; B <= Reg[IR[20:16]];`

Sprungziel berechnen: **`ALUOut <= PC + (sign-extend(IR[15:0])<<2);`**



Befehl ausführen

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

R-Typ	Befehl Holen	Reg. lesen	Ausführung	Register schreiben	
`lw,sw`			Adress-berechnung	Speicherzugriff	Register schreiben
`beq`			Sprung		

3. Execute

- R-Typ Befehle:

```
ALUOut <= A op B;
```

- Berechnung der Speicheradresse für Lade- und Speicherbefehle:

```
ALUOut <= A + sign-extend(IR[15:0]);
```

- Branch:

```
if (A==B) PC <= ALUOut;
```

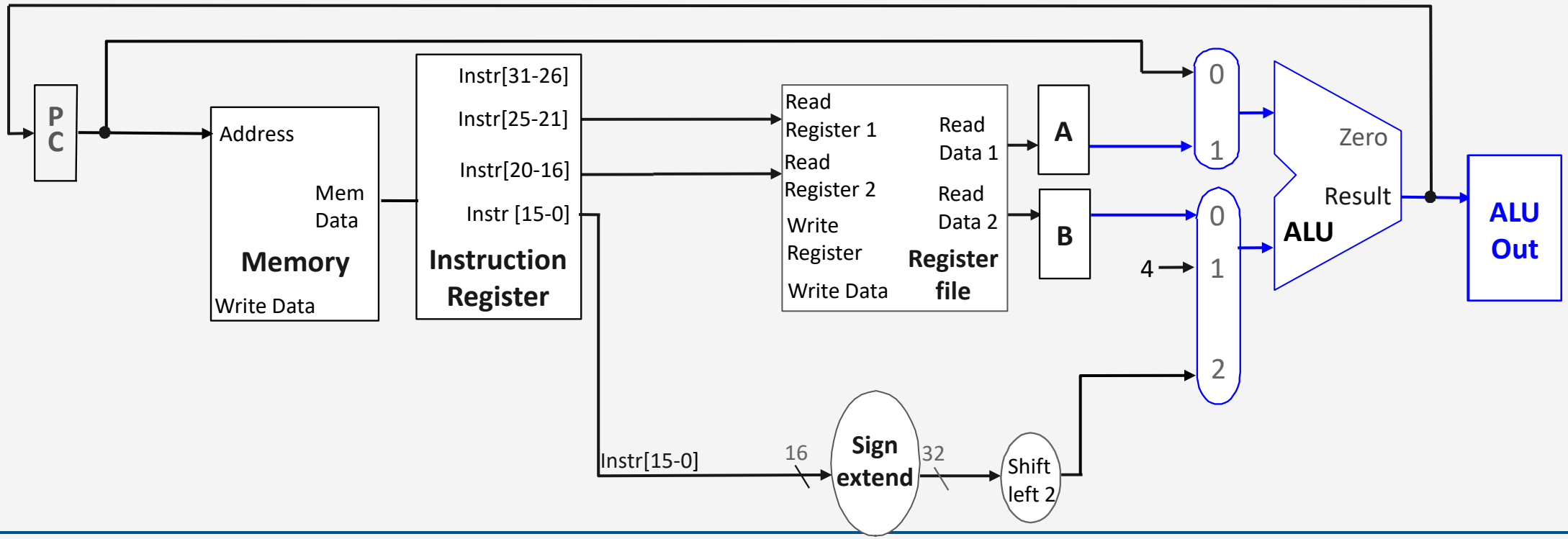
  - Sprungzieladresse in ALUOut wurde im vorigen Schritt berechnet

- Jump: `PC <= PC[31:28]#(IR[25:0]<<2);`

Ausführung (R-Befehle)

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

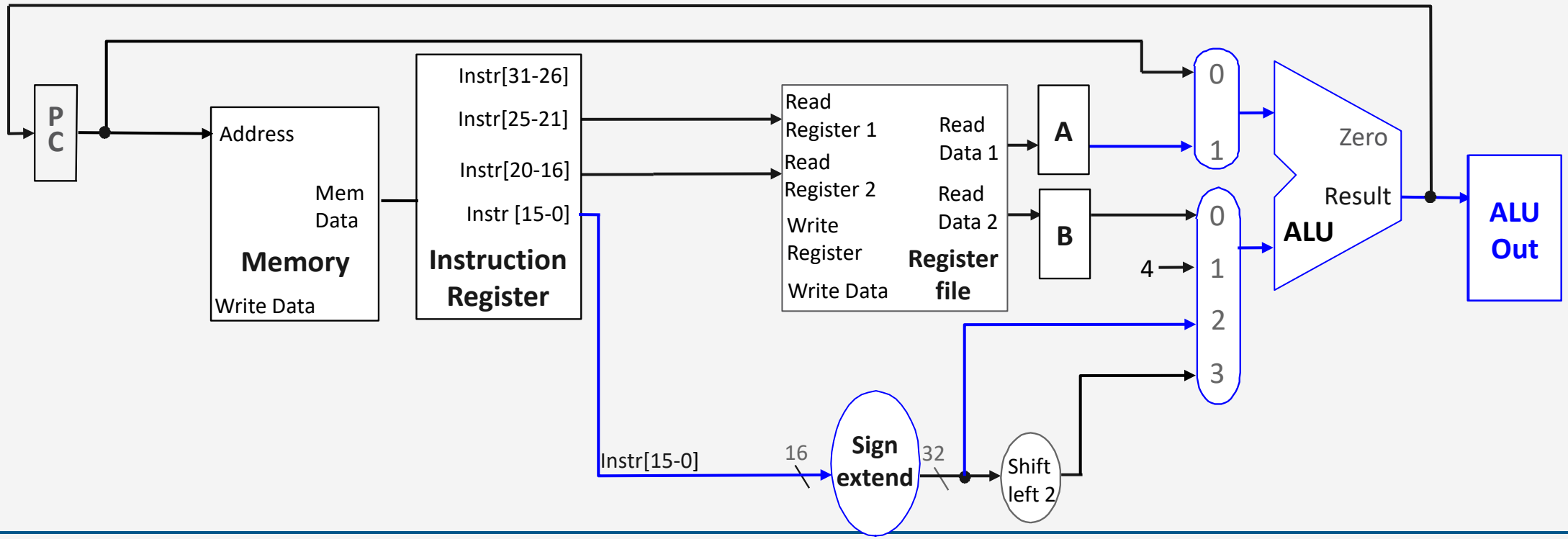
- 3. Execute (R-Typ): Ergebnis berechnen: **`ALUOut <= A op B;`**



Ausführung (Lade- und Speicherbefehle)

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

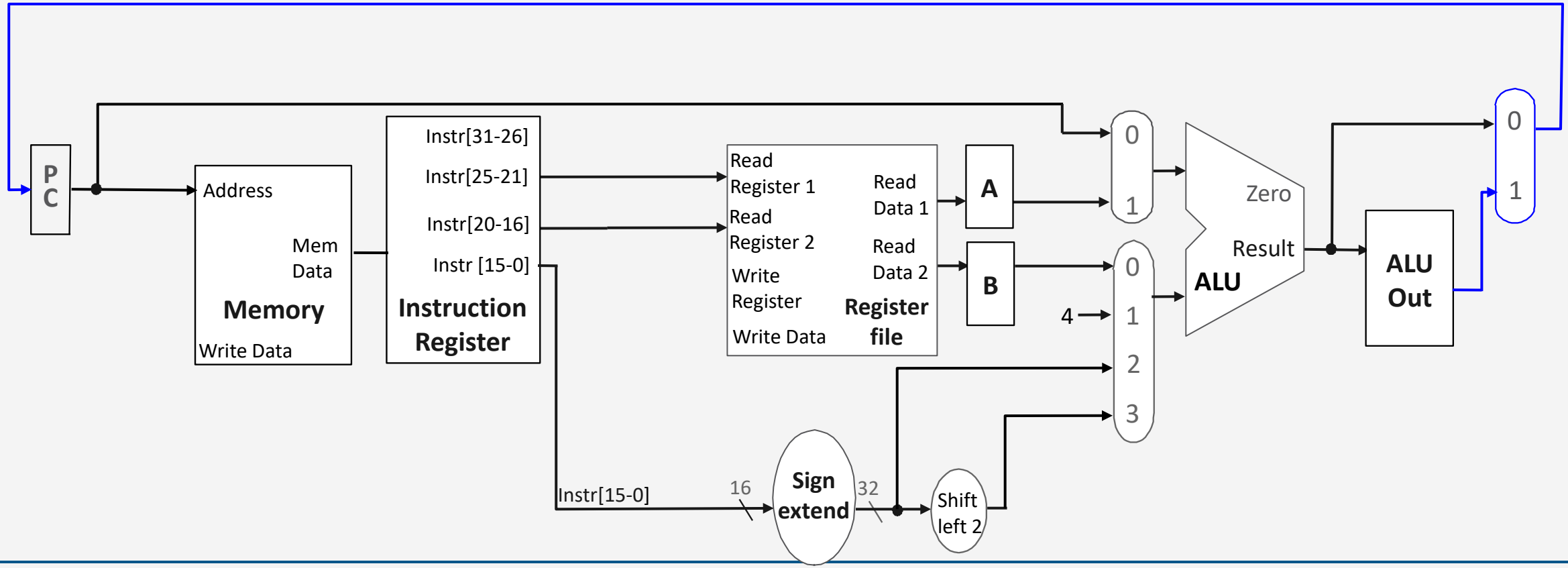
- 3. Execute (`lw`, `sw`):
  Speicheradresse berechnen: `ALUOut <= A + sign-extend(IR[15:0]);`



Ausführung (Sprungbefehle)

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

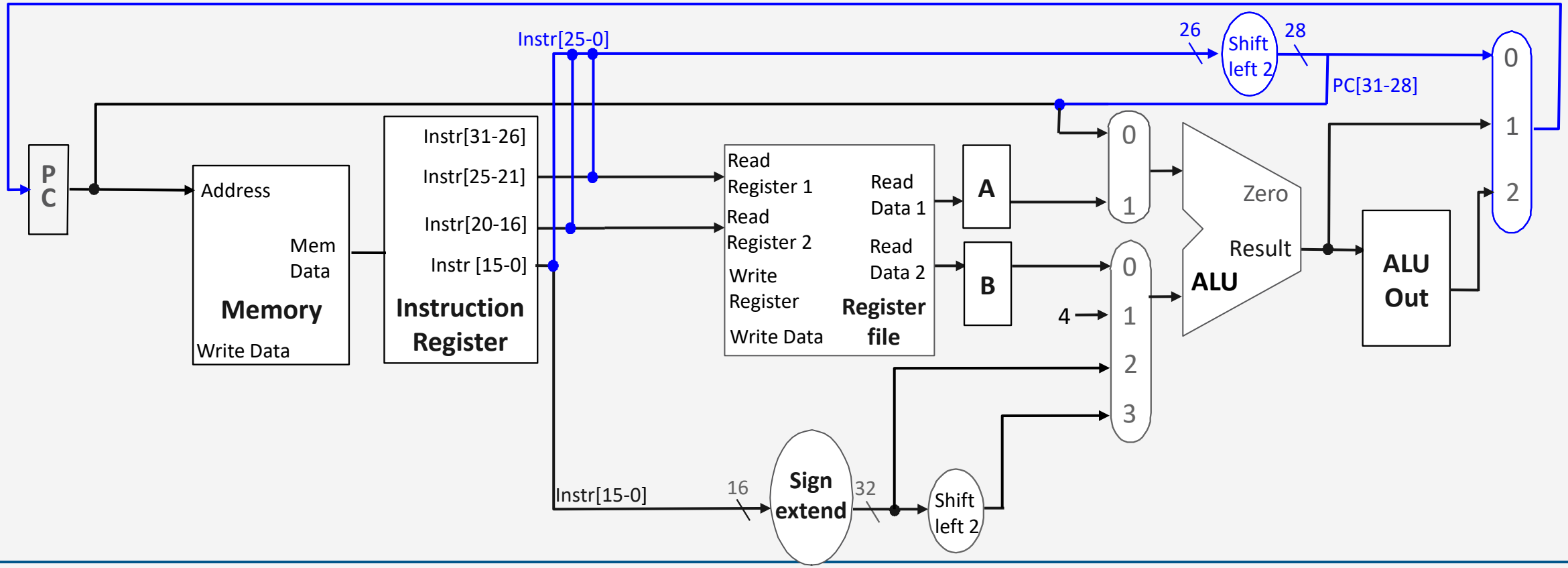
- 3. Execute (**`beq`**): PC auf Sprungziel setzen: **`if (A==B) PC <= ALUOut;`**



Ausführung (Sprungbefehle)

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- 3. Execute (`j`): PC auf Sprungziel setzen: **`PC <= PC[31:28]#(IR[25:0]<<2);`**



Speicherzugriffe und Ergebnisse schreiben

R-Type	Befehl Holen	Reg. lesen	Ausführung	Register schreiben	
`lw,sw`			Adress-berechnung	Speicherzugriff	Register schreiben
`beq`			Sprung		

4. Memory Access / Write Back

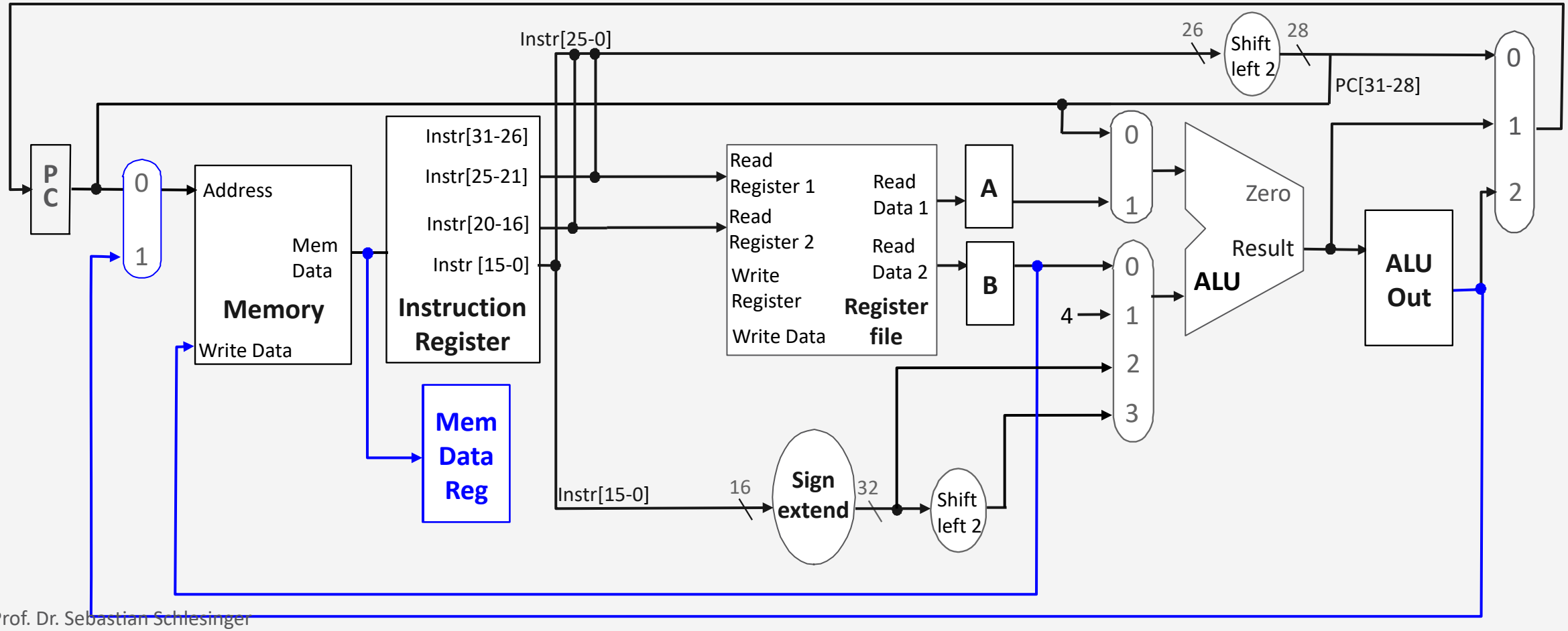
- Register für **`lw`** um geladene Daten zwischenzuspeichern: **`MDR`** (Memory Data Register)
- **`lw`**: **`MDR <= Memory[ALUOut];`**
- **`sw`**: **`Memory[ALUOut] <= B;`**
- R-Befehle werden abgeschlossen: **`Reg[IR[15-11]] <= ALUOut;`**

Speicherzugriff (Ladebefehle)

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

4. Memory Access / Write Back (`lw`):

Wert aus den Speicher laden: **`MDR <= Memory[ALUOut];`**

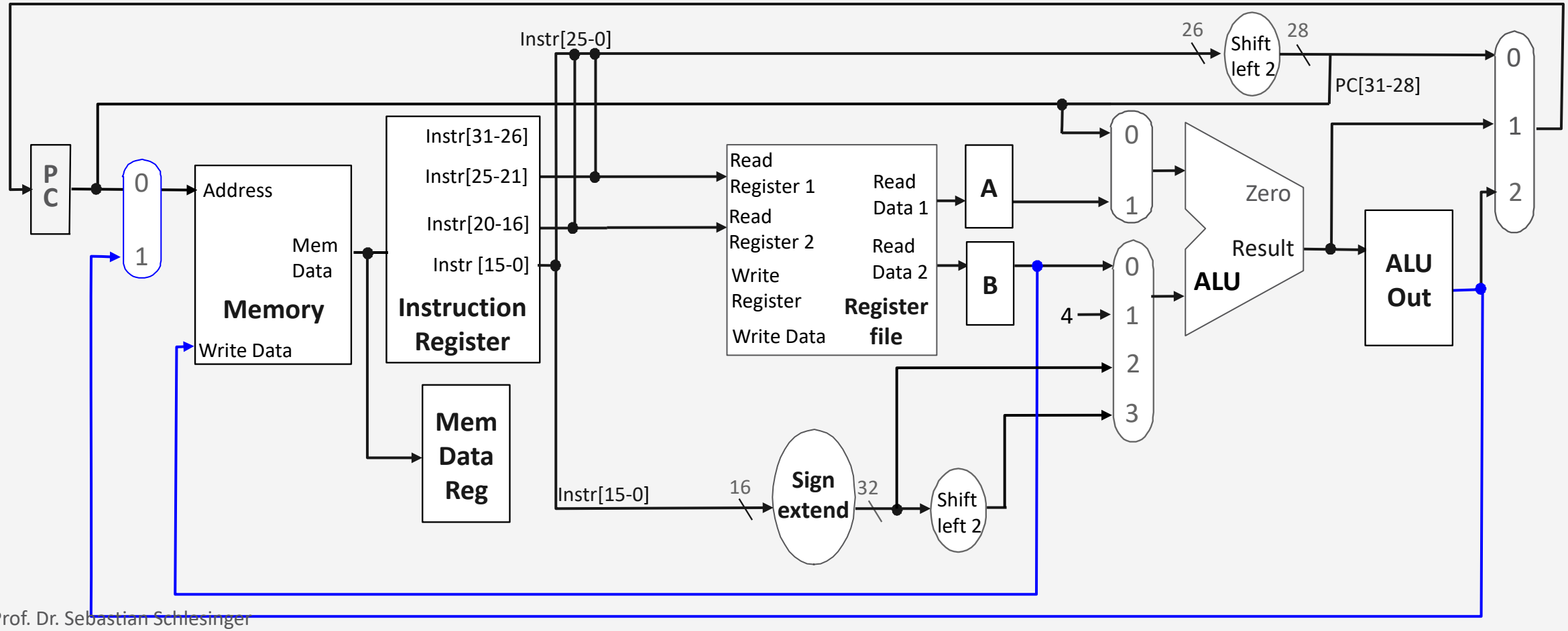


Speicherzugriff (Speicherbefehle)

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

4. Memory Access / Write Back (`sw`):

Ergebnis in den Speicher schreiben: **`Memory[ALUOut] <= B;`**

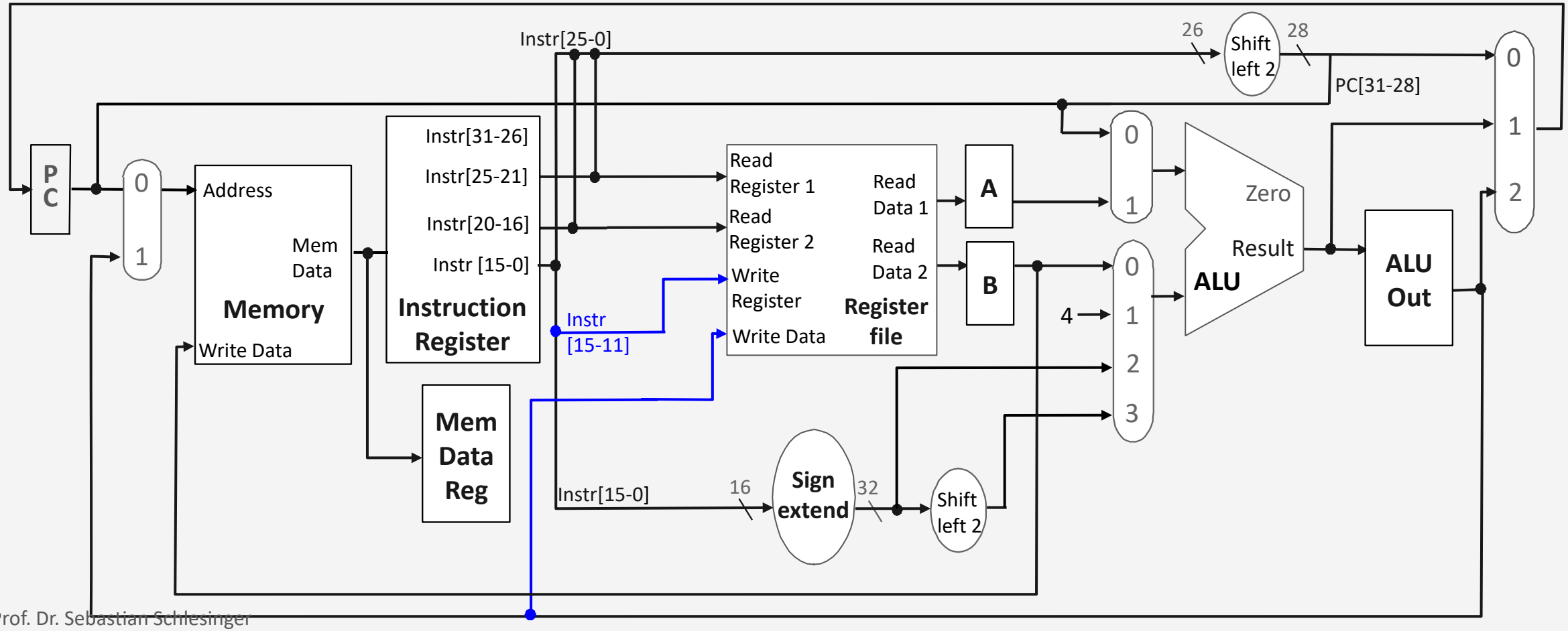


Befehlsabschluss für R-Befehle

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

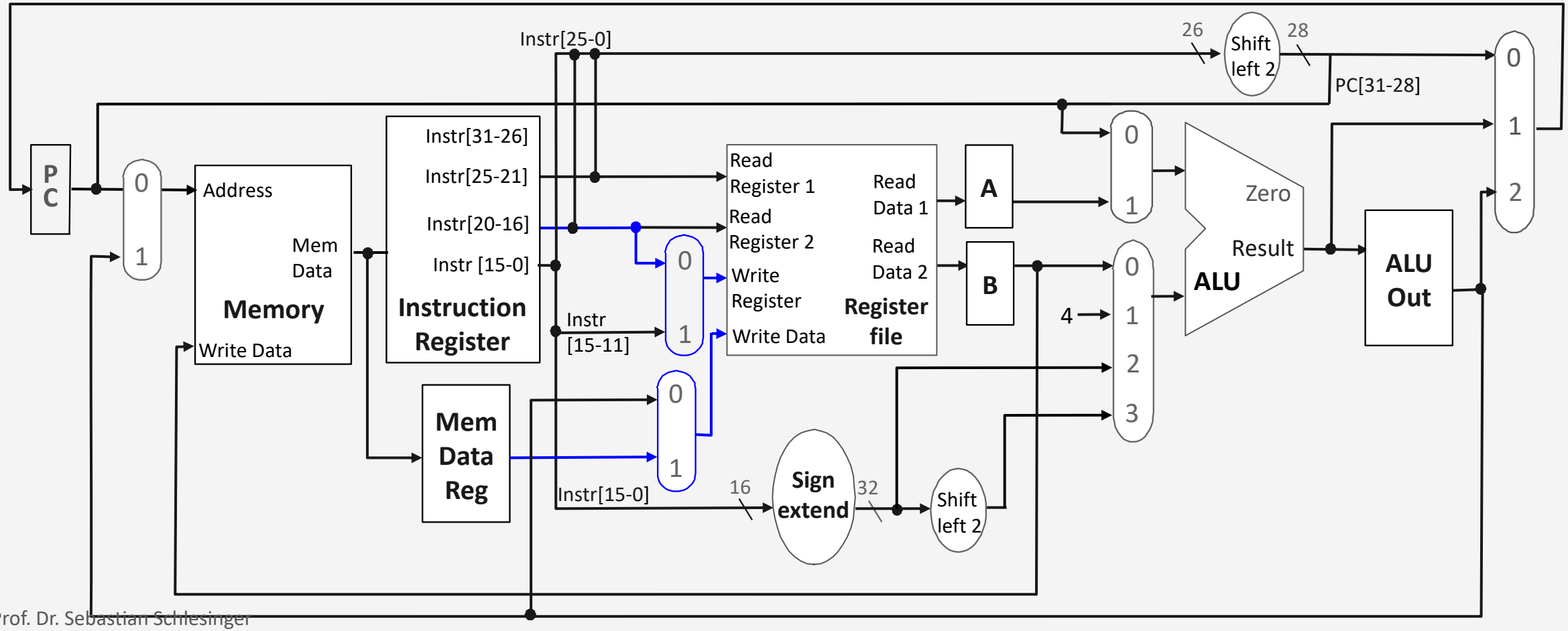
4. Memory Access / Write Back (R-Type):

Ergebnisse schreiben: **`Reg[IR[15-11]] <= ALUOut;`**



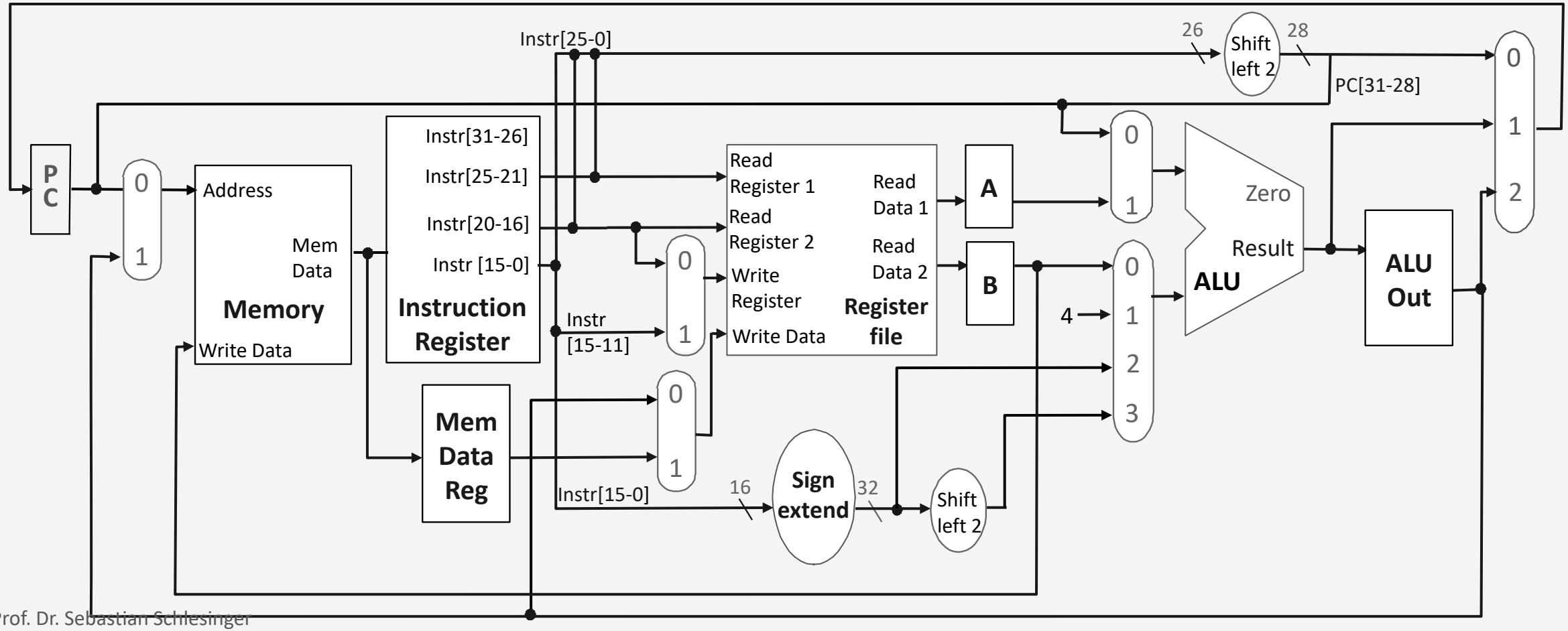
Befehlsabschluss für Ladebefehle

- 5. Write Back: Ladebefehle werden abgeschlossen: **`Reg[IR[20-16]] <= MDR;`**



Mehrtaktprozessor: Datenpfad

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law



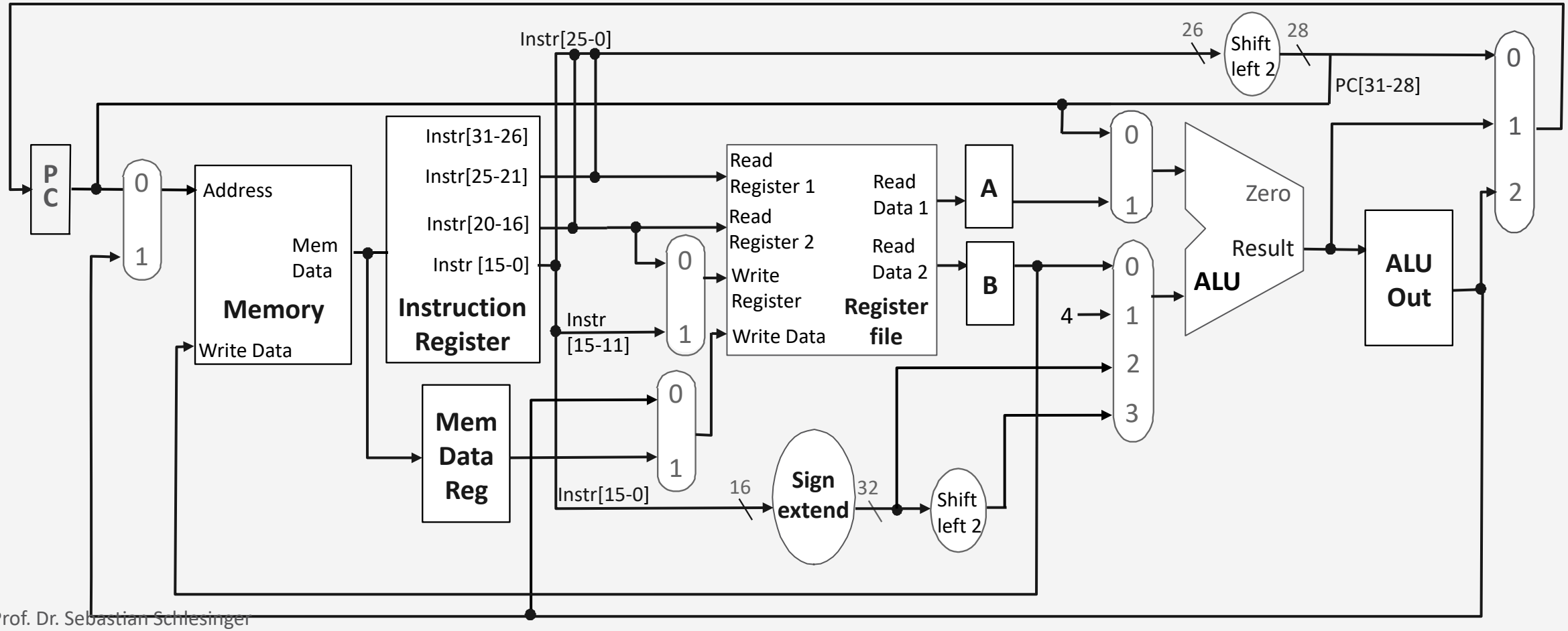
Überblick

1. Einleitung
2. Vorbereitung: ALU-Erweiterung für `slt` und `beq`
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

Steuerung des Mehrtaktprozessors

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Welche Steuersignale brauchen wir?

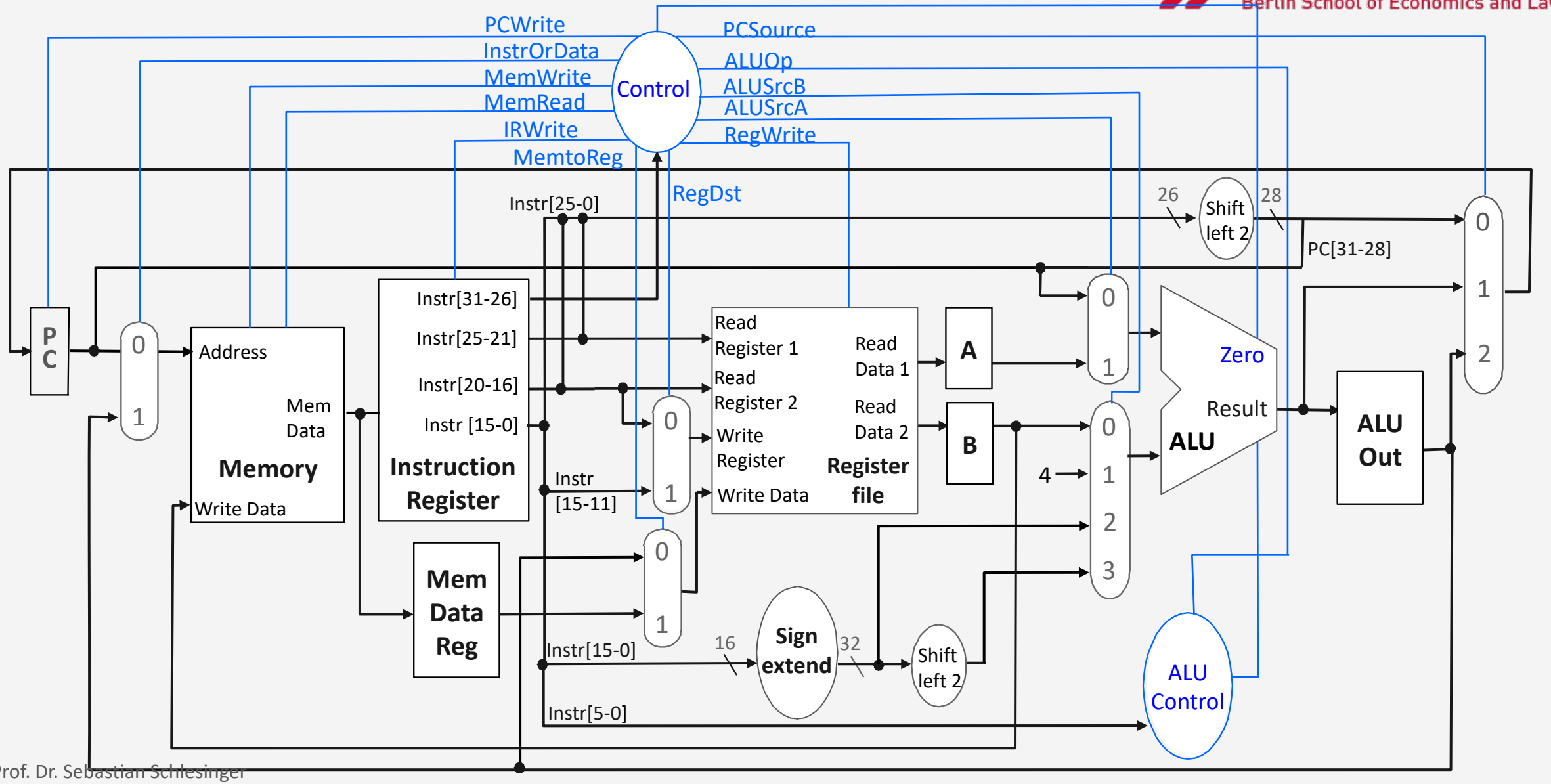


Steuersignale

- alle Multiplexer brauchen Steuersignale
- Register Steuersignale
  - **`PC`** braucht Steuersignal PCWrite
    - Nur schreiben am Ende des 1. Takt (PC+4) und bei Branch oder Jump Ausführung
  - **`IR`** braucht Steuersignal IRWrite
    - Soll nur am Ende des 1.Takt geschrieben werden
  - Registersatz braucht Steuersignal RegWrite
    - wie im Eintaktprozessor
  - **`MDR`, `A`, `B`, `ALUOut`** brauchen keine Steuersignale
    - Sollen Daten nur zwischen aufeinanderfolgenden Taktzyklen halten
- Übrige Steuersignale
  - MemRead, MemWrite, ALUOp kennen wir vom Eintaktprozessor

Mehrtaktprozessor

Hochschule für Wirtschaft und Recht Berlin
Berlin School of Economics and Law



Mikrobefehle

- Mikrobefehle für die Ausführung aller Befehlsklassen:

Schritt	R-Typ Befehle	Speicher-befehle	Verzwei-gungen	Sprünge
Befehlsholschritt	IR = Memory[PC] PC = PC+4			
Befehlsentschlüsselungs- und Registerholschritt	A = Reg[IR[25-21]] B = Reg[IR[20-16] ALUOut = PC + (sign-extend(IR[15-0]) << 2)			
Ausführung oder Adressberechnung oder Sprungausführung	ALUOut = A op B	ALUOut = A + sign-extend(IR[15-0])	if (A==B) then PC = ALUOut	PC = PC[31-28] || (IR[25-0]<<2)
Speicherzugriff oder R-Befehlabschlussschritt	Reg[IR[15-11]] = ALUOut	Load: MDR = Mem[ALUOut] Store: Mem[ALUOut] = B		
Speicherleseabschluss		Load: Reg[IR[20-16]] = MDR		

Eintakt vs. Mehrzyklen

- Eintaktprozessor
  - Alle Instruktionen dauern 1 Takt
  - Längerer Takt
  - Separater Befehls- und Datenspeicher
  - Steuersignale nur abhängig vom Befehl
  - Separate ALU, Addierer um PC+4 zu berechnen, Addierer um Sprungzieladresse zu berechnen
- Mehrzyklenprozessor
  - Instruktionen dauern 3-5 Takte
  - Kürzerer Takt
  - Ein Speicher für Befehle sowie für Daten
  - Steuersignale hängen auch ab vom aktuellen Ausführungstakt
  - Die ALU
    - führt Operationen aus
    - berechnet Speicheradresse
    - inkrementiert Programmzähler
    - berechnet Sprungzieladresse
    - jedoch in verschiedenen Schritten/Takten

Was findet wann statt?

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Wie viele Taktzyklen dauert es, diesen Code auszuführen?

```
lw   $t2,0($t3)
lw   $t3,4($t3)
beq $t2,$t3,L1# nehme an, nicht genommen
add $t5,$t2,$t3
sw   $t5,8($t3)
L1:  ...
```

- Was passiert im 8. Ausführungszyklus?

- In welchem Zyklus findet die eigentliche Addition von **`$t2`** und **`$t3`** statt?

Was findet wann statt?

Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Wie viele Taktzyklen dauert es, diesen Code auszuführen?

```
lw   $t2,0($t3)
lw   $t3,4($t3)
beq $t2,$t3,L1# nehme an, nicht genommen
add $t5,$t2,$t3
sw   $t5,8($t3)
L1:  ...
```

  - **lw** 5 Takte, **beq** 3 Takte, **add** und **sw** 4 Takte: 5 + 5 + 3 + 4 + 4 = 21 Taktzyklen
- Was passiert im 8. Ausführungszyklus?
  - 3. Ausführungszyklus des 2. **lw** → ALU berechnet die Adresse
- In welchem Zyklus findet die eigentliche Addition von **$t2** und **$t3** statt?
  - 3. Zyklus von add = 16. Ausführungzyklus insgesamt

Steuerung des Mehrzyklenprozessors

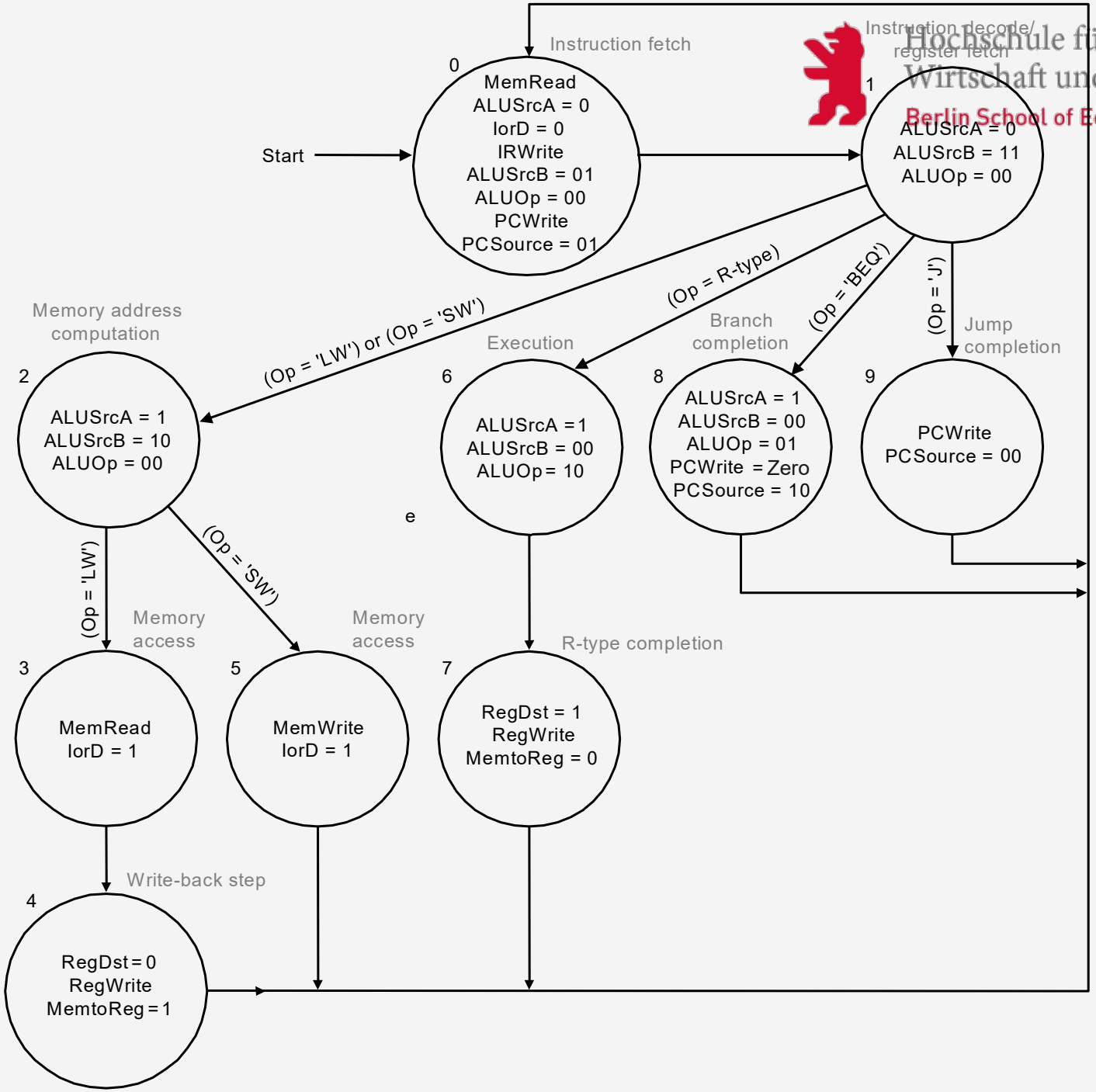
- Steuersignale hängen ab von:
  - Welcher Befehl wird ausgeführt
  - Aktueller Ausführungsschritt der Instruktion

- Benutze die vorhandene Information um einen endlichen Zustandsautomaten (*finite state machine*, FSM) zu spezifizieren:
  - Spezifiziere FSM graphisch, oder
  - Benutze Mikroprogrammierung

- Implementierung kann von der Spezifikation abgeleitet werden.

Steuerung als FSM

1. Instruction fetch
2. Instruction decode / Register fetch
3. Execution
4. Memory access / R-type completion
5. Write-Back



Überblick

1. Einleitung
2. Vorbereitung: ALU-Erweiterung für `slt` und `beq`
3. Eintaktprozessor (führt alle Befehle in 1 Takt aus)
   - Datenpfad
   - Steuerung
   - Leistung
4. Mehrtaktprozessor
   - Ausführungsschritte von Befehlen auf Register-Transfer-Ebene
   - Datenpfad
   - Steuerung
5. Zusammenfassung

Zusammenfassung

- Wir können einen Eintaktprozessor bauen!
  - Datenpfad: ALU, Register, Speicher, Multiplexer
  - Steuersignale wählen die Operation aus und kontrollieren den Datenfluss (Multiplexer, ALU-Operation, Lese-/Schreibzugriffe)
  - Steuersignale werden aus dem Opcode und ggf. dem Funktionsfeld berechnet

⇨ Eintaktprozessor ist voll funktionstüchtig aber ineffizient

- Mehrzyklenprozessor
  - kürzerer Takt und Wiederverwendung von Hardware zur Effizienzsteigerung
  - komplexere Steuerung (FSM oder Mikroprogrammierung)

CISC vs. RISC

- MIPS Designphilosophie: *Reduced Instruction Set Computer* (RISC)
  - nur einfach zu dekodierende und schnell auszuführende Befehle

- Alternative Designphilosophie: *Complex Instruction Set Computer* (CISC)
  - komplexere und umfangreichere Befehlssätze
  - um auch komplexere Rechenschritte mit einem Maschinenbefehl ausführen zu können
  - um dadurch schneller und leistungsfähiger zu werden
- Steuerung implementiert mittels Mikroprogrammierung (Mikrobefehle auf Register-Transfer-Ebene)
- Intels Befehlssatz (x86, IA32) kann man als CISC bezeichnen
  - wird jedoch während Ausführung zu RISC-ähnlichen Mikrooperationen übersetzt, die dann ausgeführt werden

Intel Nehalem Mikroarchitektur

- 1. Einsatz: Core i7 (2008)
- 45 nm
- Kombiniert festverdrahtete (FSM) Steuerung für einfache Befehle mit mikrokodierter Steuerung für komplexe Befehle (seit 80486)
- Bis zu vier Befehle werden pro Takt übersetzt in Mikro-Operationen
- Komplexe x86 Instruktionen werden durch ein Mikroprogramm abgewickelt (Register-Transfer-Befehle)