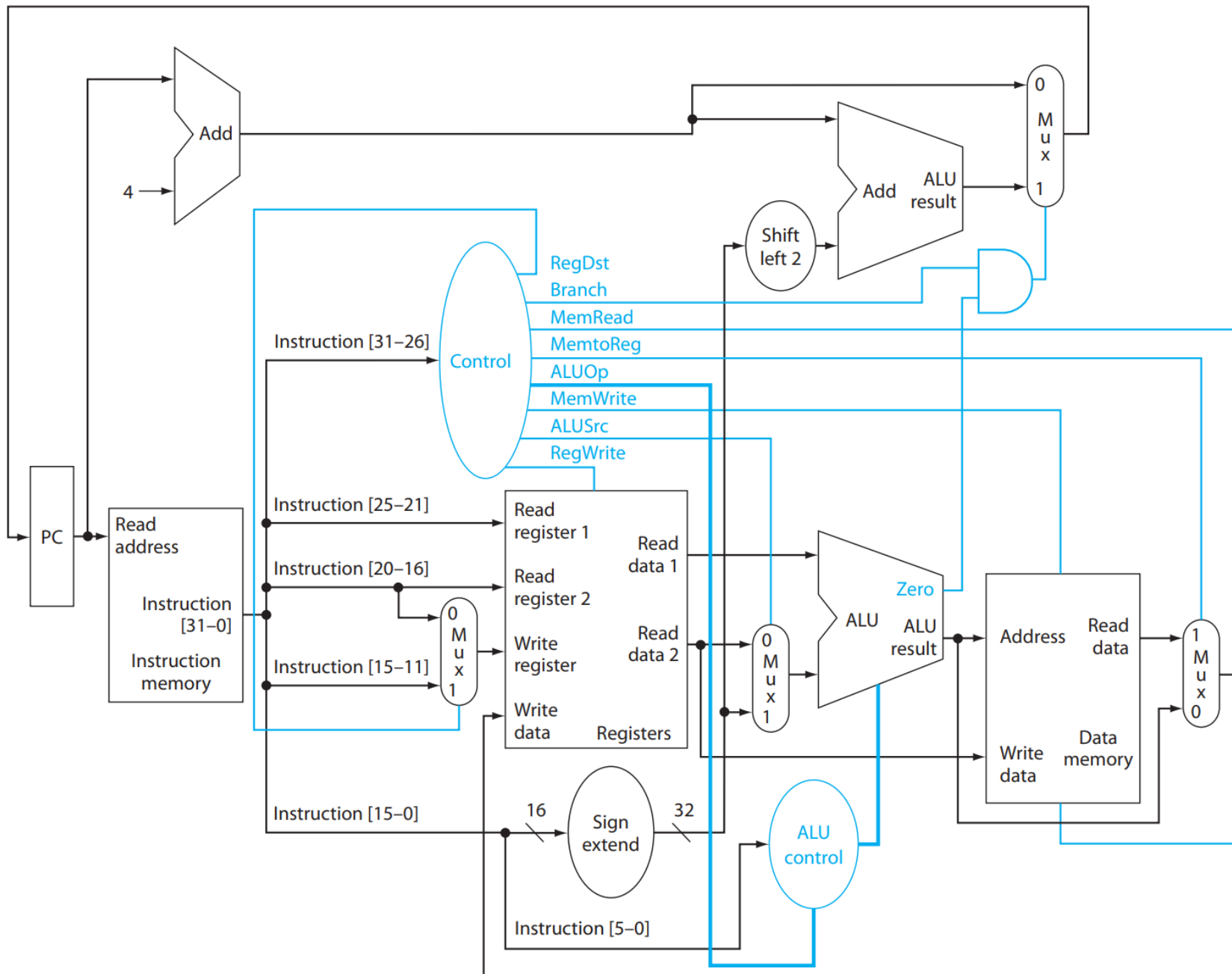


Eintaktprozessor



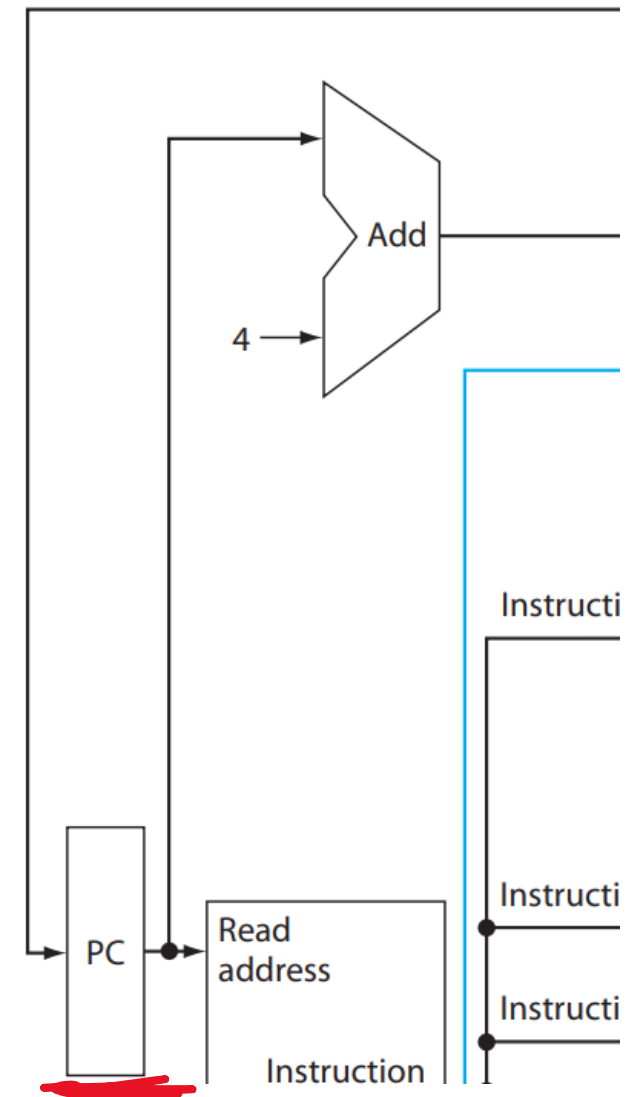
- **Blau: Steuerung**
  - Steuert den Prozessor
  - Kontrolliert den Datenpfad
  - Führt den Befehl aus
- **Schwarz: Datenpfad**
  - Führt tatsächliche Operationen aus
- **Warum Eintaktprozessor?**
  - Ein Takt für einen Befehl



Datenpfad

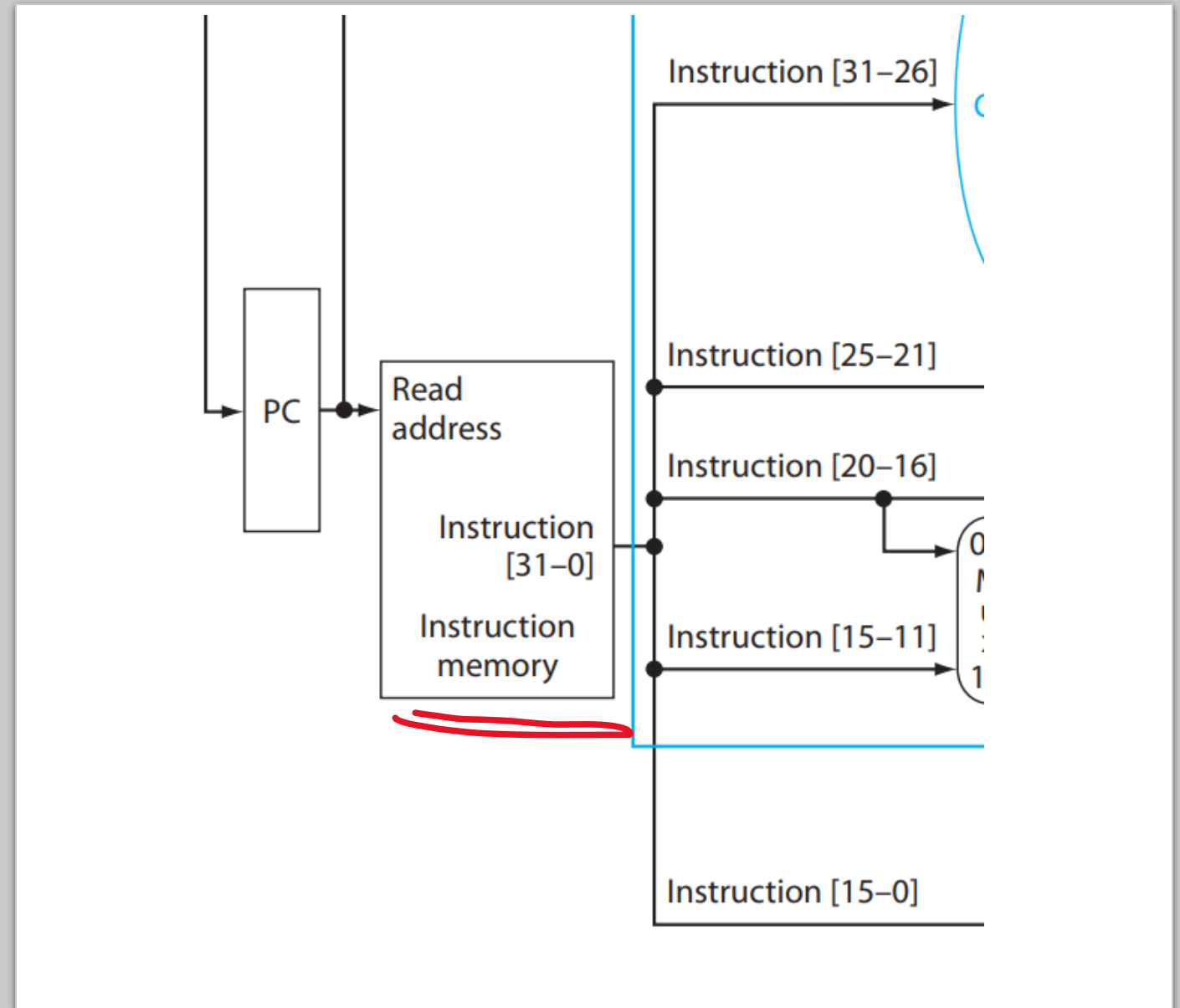
# PC - Program Counter

- Zählt beim wievielten Byte wir im Programmtext sind
- Wird in jedem Takt bedingungslos um 4 erhöht
  - Nächster Befehl
  - Word-Abstand -> 1 Word = 4 Byte



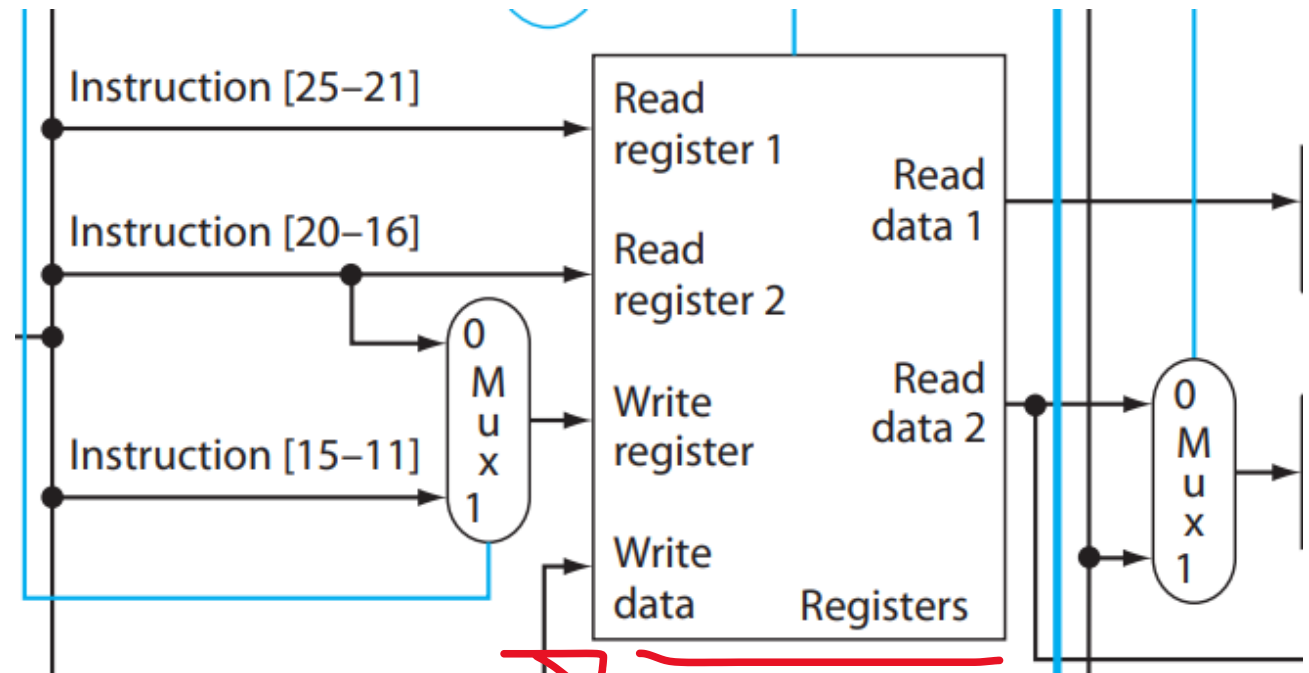
# Instruction Memory

- Befehle vom Programm gespeichert
- Befehl wird vom PC geladen
- NächstBefehl = InsMem[PC]
- In jeden Takt: Befehl bei Adresse PC geladen
- Befehl wird in einzelne Fleder aufgeteilt
  - Op, rt, rs, rt, Imm



# Registers

- Gleicher Registersatz, wie bei MARS
- Input: (vom Geladenen Befehl)
- Read reg 1 =  $\text{Inst}[25-21] = \text{rs}$
- Read reg 2 =  $\text{Inst}[20-16] = \text{rt}$
- Write reg =  $\text{Inst}[15-11] = \text{rd}$   
(oder =  $\text{Inst}[20-16] = \text{rt}$  bei I-Format)
- Outputs:
- Read data 1 =  $\text{Reg}[\text{Inst}[25-21]]$
- Bsp: `add $t0, $t1, $a0`
- Read data 1 =  $\$t0$



(2) 29<sub>hex</sub> Load Immediate  
 (2) 2b<sub>hex</sub> Move  
 (1) 0 / 22<sub>hex</sub>  
 0 / 23<sub>hex</sub>

iate }  
 ate, 2'b0 }  
 }  
 2's comp.)  
 0 not atomic

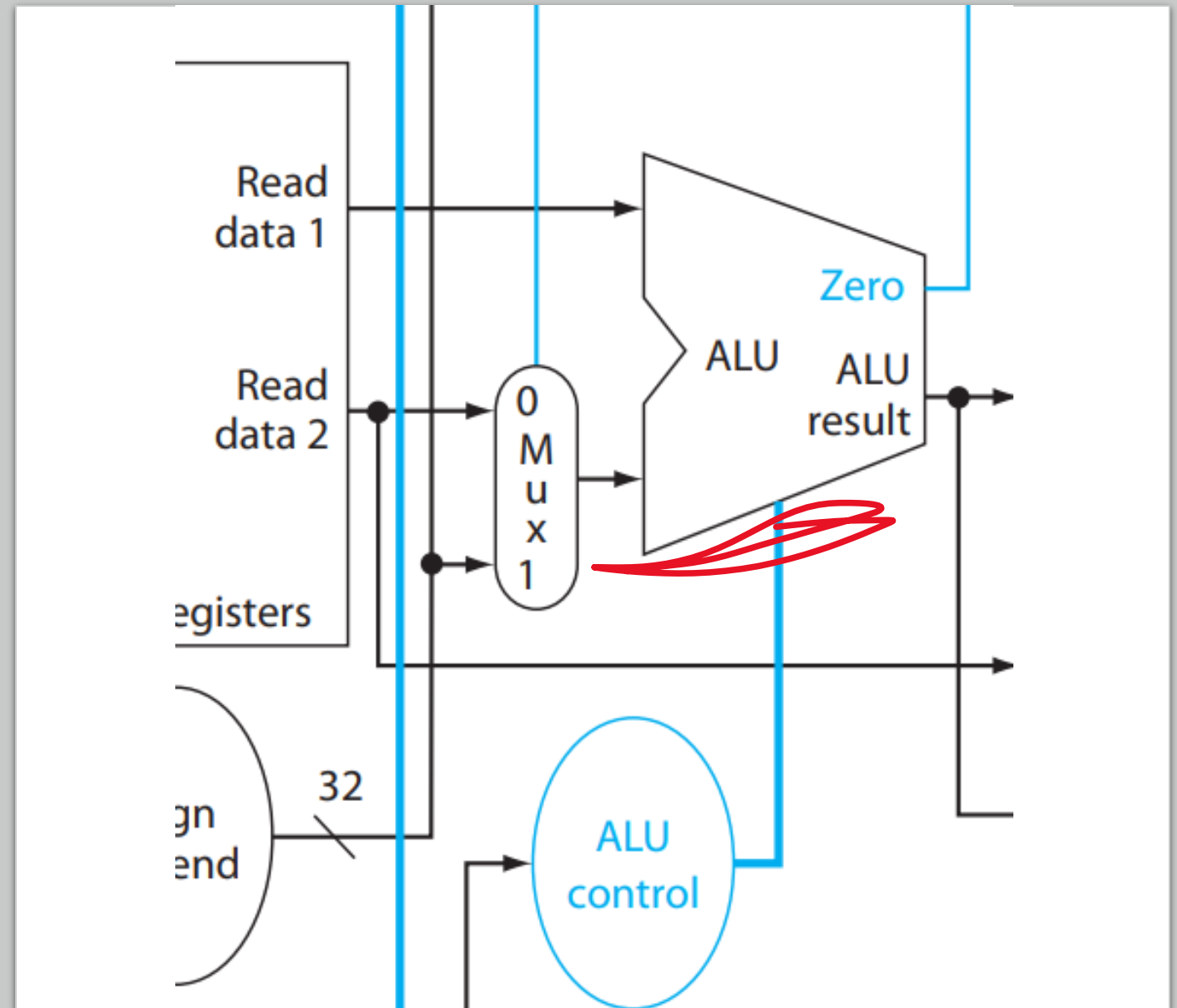
funct  
 6 5 0  
 te  
 0  
 0

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

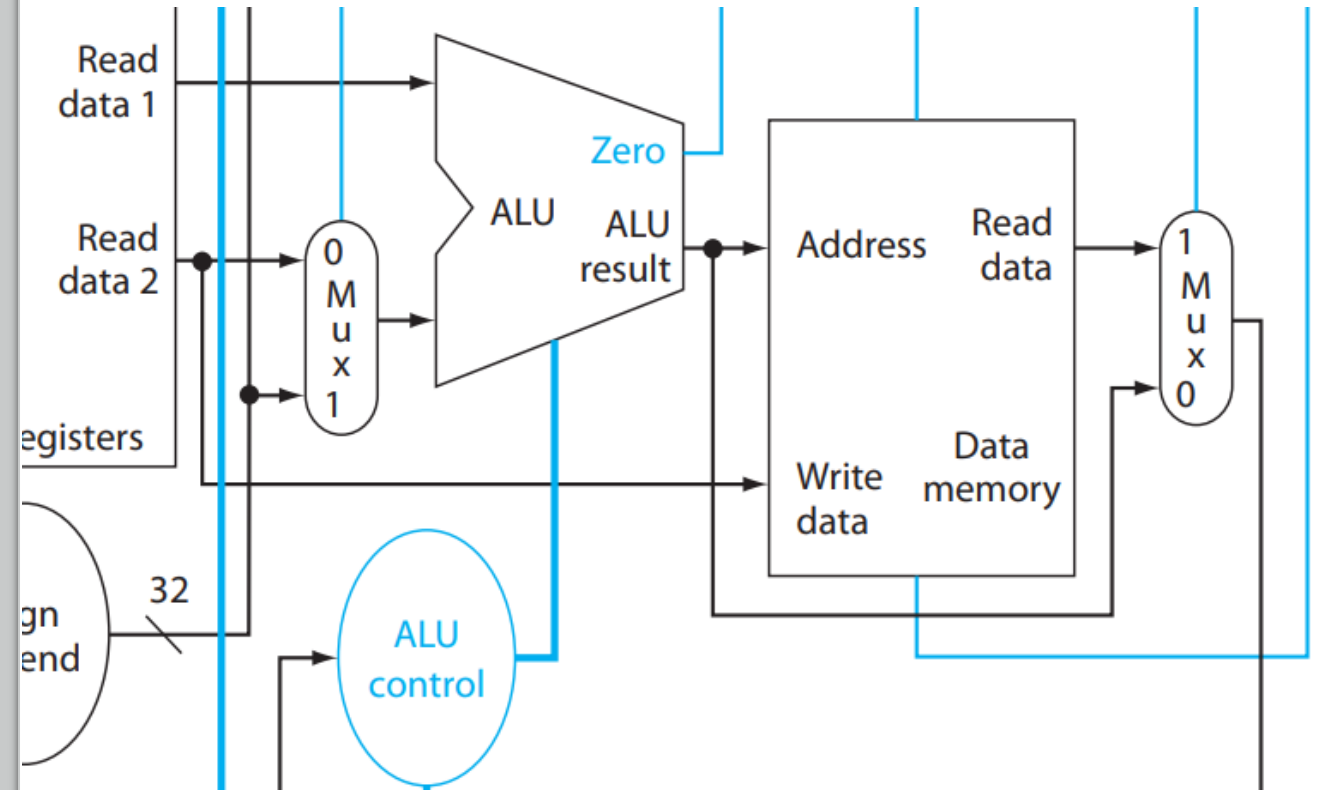
# ALU

- Input:
- 1. Read data 1
- 2. Read data 2 / Imm. (Sign Ext.)
- Steuersignale (A\_inv, B\_inv, OP) <- ALU Control <- Befehl Op u. Funct
- Output:
- In Register geladen (add \$t1, \$t2, \$t3 in \$t1 geladen)
- Oder als Adresse für Speicher
- Zero für branch (bnq, bne)



# Data Memory

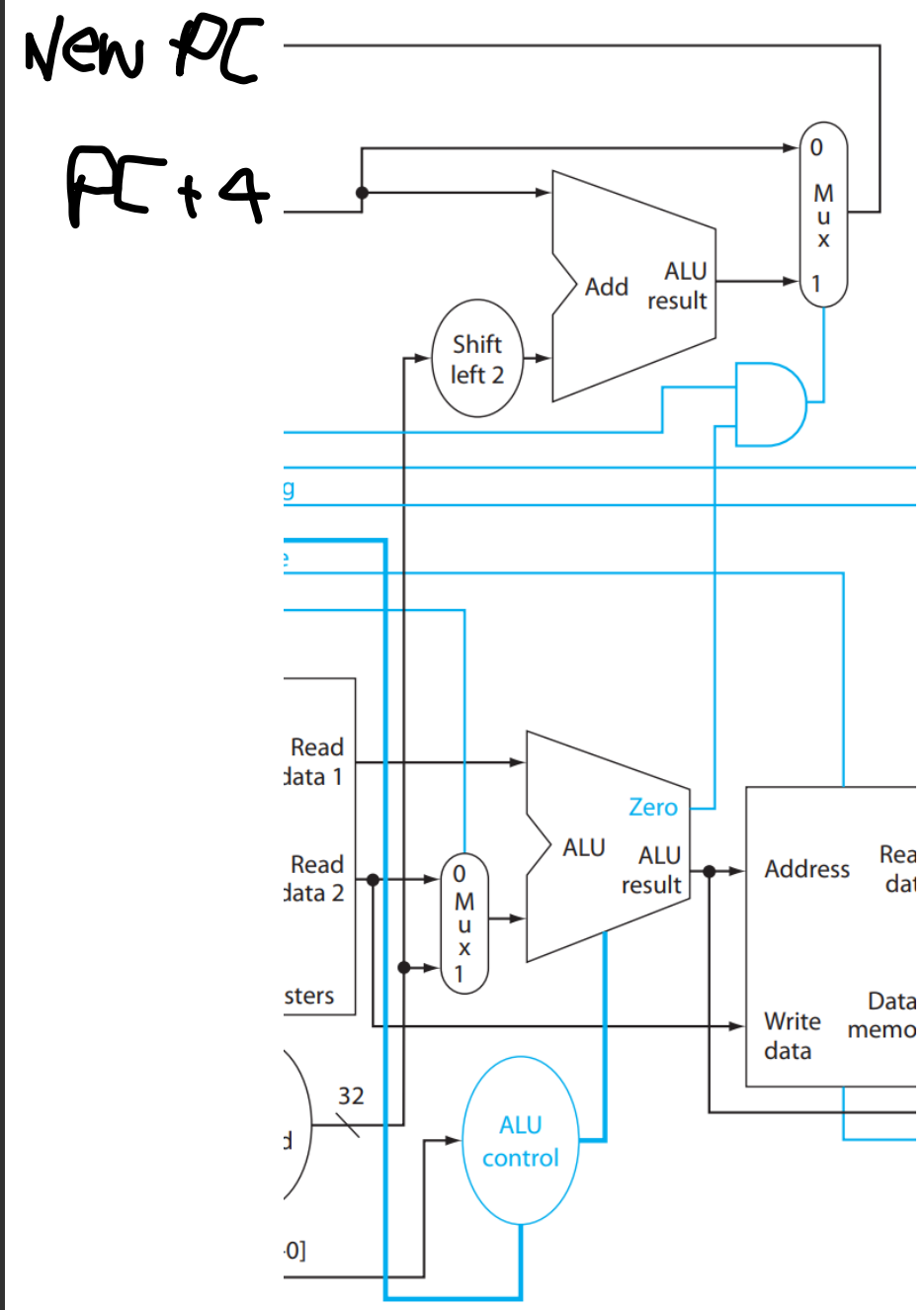
- Speicher
- Benutzt bei: lw, sw, lb, sb ...
- Adresse ist Ergebnis von ALU
  - Reg. Mit Imm addiert: lw \$sp, 4(\$sp) -> Addr = \$sp + 4
- Mux entscheidet zwischen Ergebnis aus Speicher (bei lw) oder Ergebnis von ALU (bei add)





# Branch Logik

- Für beq, bne
- Zielsprungadresse = Alter PC + 4 \* Imm.
- Zero (ALU Output) gibt an ob gleich oder ungleich
- Zero entscheidet ob Zielsprungadresse oder PC + 4 (branch oder kein branch)





Steuerung

# Steuerung

- Durch Op-Code bestimmt
- RegDst: Ergebnis in rt oder rd?
  - 0: rt, 1: rd
- Branch: Branch-Befehl
- MemRead: Aus speicher lesen
- MemtoReg: Aus speicher laden
  - 0: ALU, 1: MEM
- ALUOp: Was die ALU machen soll
- MemWrite: In speicher schreiben
- ALUSrc: 2. Operand Reg oder Imm?
  - 0: Register, 1: Imm
- RegWrite: Ergebnis in Register laden

