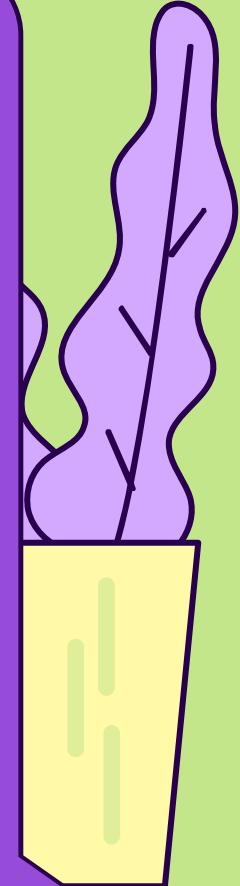
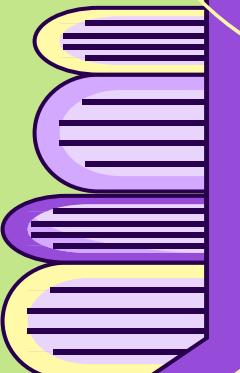




Herzlich willkommen
zum achten ERA
Tutorium des
Semesters!



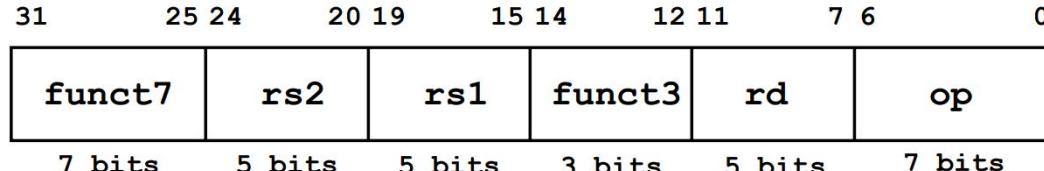
Die Instruktionen wurden bisher in **Assemblersprache** notiert, der Prozessor versteht aber nur **Maschinensprache**

- Instruktionen müssen binär codiert werden → Instruktionscodierung
add t0, s1, s2 → 00000001001001001000001010110011

Bei RISC-V sind **alle Instruktionen 32 Bit lang!**

- Da die verschiedenen Instruktionen unterschiedlich viele Operanden haben, werden 6 Instruktionsformate verwendet:
 - **R-Typ:** Register-zu-Register Operationen
 - **I-Typ:** Short Immediates und Ladebefehle
 - **U-Typ:** Befehle für Long Immediates
 - **S-Typ:** Abspeichern von Datenworten
 - **B-Typ:** Bedingte Sprünge
 - **J-Typ:** Unbedingte Sprünge

Instruktionsformat R-Typ ([Register-Format](#)) wird meist für arithmetische und logische Instruktionen verwendet

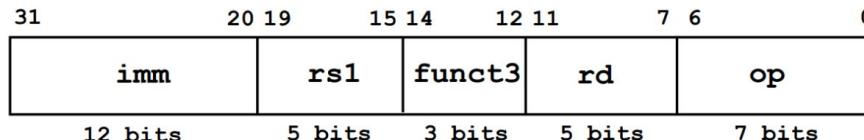


Felder

- funct7 Funktionscode 7 Bit (function) *
- rs2 Register des zweiten Quelloperanden (source 2)
- rs1 Register des ersten Quelloperanden (source 1)
- funct3 Funktionscode 3 Bit (function) *
- rd Register, in dem das Ergebnis gespeichert wird (destination)
- op Operationscode (OP-Code) * : 51 (arithm. und logische)

Instruktsionsformat I-TYP

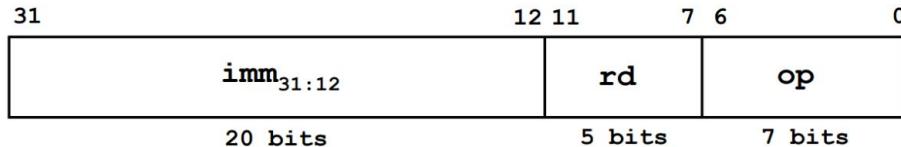
- Instruktsionsformat I-Typ ([Immediate-Format](#)) wird verwendet für
 - Immediate-Versionen der arithmetischen und logischen Instruktionen, Datentransferinstruktionen (ladend) und für unbedingte Sprünge



- Felder
 - immediate 12-Bit Konstante Sign Extended (Werte von -2^{11} bis $2^{11} - 1$)
 - rs1 Quellregister
 - funct3 Funktionscode 3 Bit (function)
 - rd Zielregister
 - op Operationscode (OP-Code): 3 (load), 19 (arithm. und logisch), 103 (Sprünge)
- Shift Operationen vom I-Typ verwenden die obersten 7 Bit als Funktionscode (instr_{31:25})

Instruktionsformat U-TYP

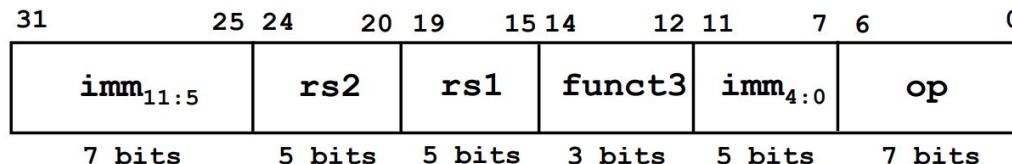
- Instruktionsformat U-Typ ([Upper Immediate-Format](#)) wird verwendet für
 - Instruktionen um die höheren 20-Bit in Registern mit Immediate Werten ansprechen zu können



- Felder
 - immediate 20-Bit Konstante
 - rd Zielregister
 - op Operationscode (OP-Code): 23, 55
- U-Typ Instruktionen haben kein Funktionsfeld

- Instruktionsformat S-Typ (**Store-Format**) wird verwendet für
 - Datentransferinstruktionen (speichernd)

sw a0, imm(a1)

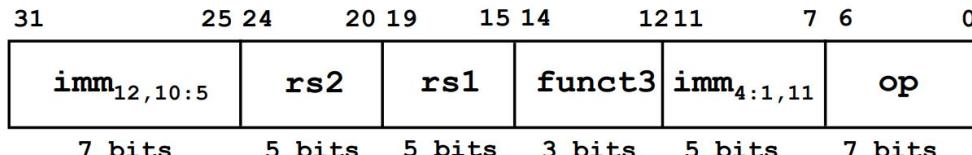


Felder

- immediate 12-Bit Konstante aufgeteilt auf 2 Felder (Werte von -2^{11} bis $2^{11} - 1$)
- rs1 Basisregister
- rs2 Wert, der gespeichert werden soll
- funct3 Funktionscode 3 Bit (function)
- op Operationscode (OP-Code): 35

Instruktionsformat B-Typ (Branch-Format)

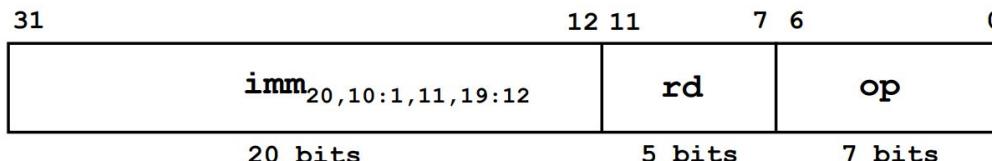
- Wird verwendet für Bedingte Sprünge



Felder

- immediate 13-Bit Konstante aufgeteilt auf 2 Felder (Werte von -2^{11} bis $2^{11} - 1$)
- rs1, rs2 Registeroperanden (Vergleichsregister)
- funct3 Funktionscode 3 Bit (function)
- op Operationscode (OP-Code): 99

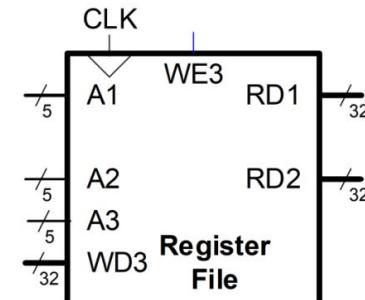
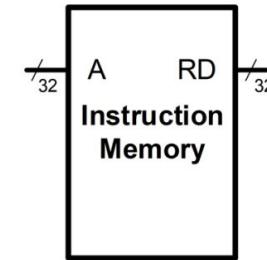
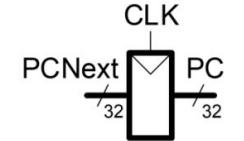
- Instruktionsformat J-Typ ([Jump-Format](#))
 - wird für unbedingte Sprünge verwendet



- Felder
 - immediate 20-Bit Konstante
 - rd Zielregister
 - op Operationscode (OP-Code): 111
- 21 Bits genügen um jede gültige Adresse im Text-Segment des Speichers darzustellen
 - Wie beim B-Type ist das niedrigste Bit immer 0 und daher reichen 20 Bit zum Kodieren
- J-Typ Instruktionen haben kein Funktionsfeld

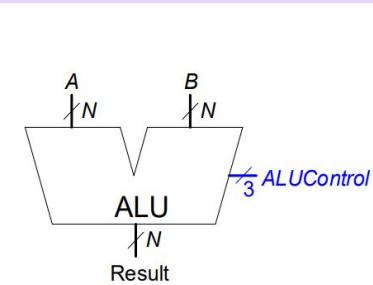
Komponenten (1)

- **Program Counter (PC):** Adresse des nächsten Befehls
- **Instruktionsspeicher**
 - A: Adresse
 - RD (Read Data): Gelesene Instruktion
 - Zunächst starke Vereinfachung: nur ROM, Lesen damit rein kombinatorisch
- **Registerbank**
 - 2 Ports zum Lesen von Operanden (RD1, RD2; zugehörige Adressen A1 und A2)
 - 1 Port zum Schreiben des Resultats (WD3 und Adresse A3)
 - Es wird nur geschrieben wenn WE3=1 ist (Write Enable)



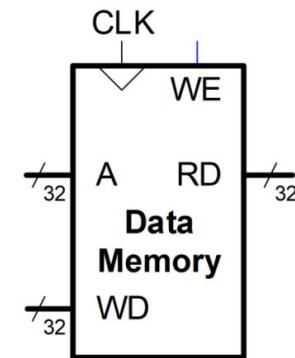
■ Arithmetic Logic Unit (ALU)

- Rein kombinatorisch
- 32-Bit Ausgang mit Ergebnis
- 1-Bit Ausgang, der angibt, ob Ergebnis 0 ist (nicht dargestellt)



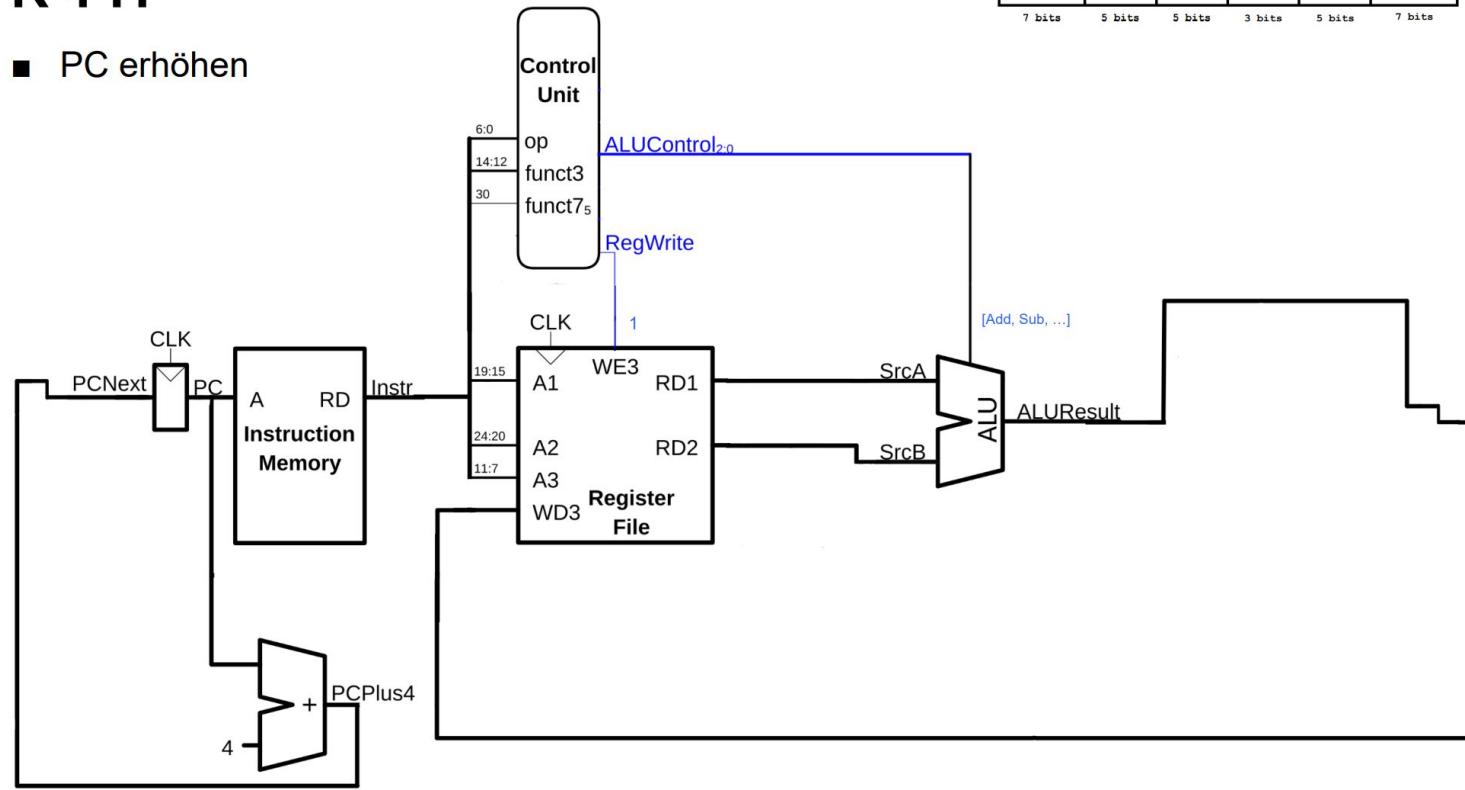
■ Datenspeicher

- A: Adresse zum Lesen/Schreiben (Wortadressierung)
- RD (Read Data): Gelesenes Datum (lesen ist rein kombinatorisch)
- WD (Write Data): Zu schreibendes Datenwort
- Geschrieben wird nur bei positiver Taktflanke und wenn WE=1



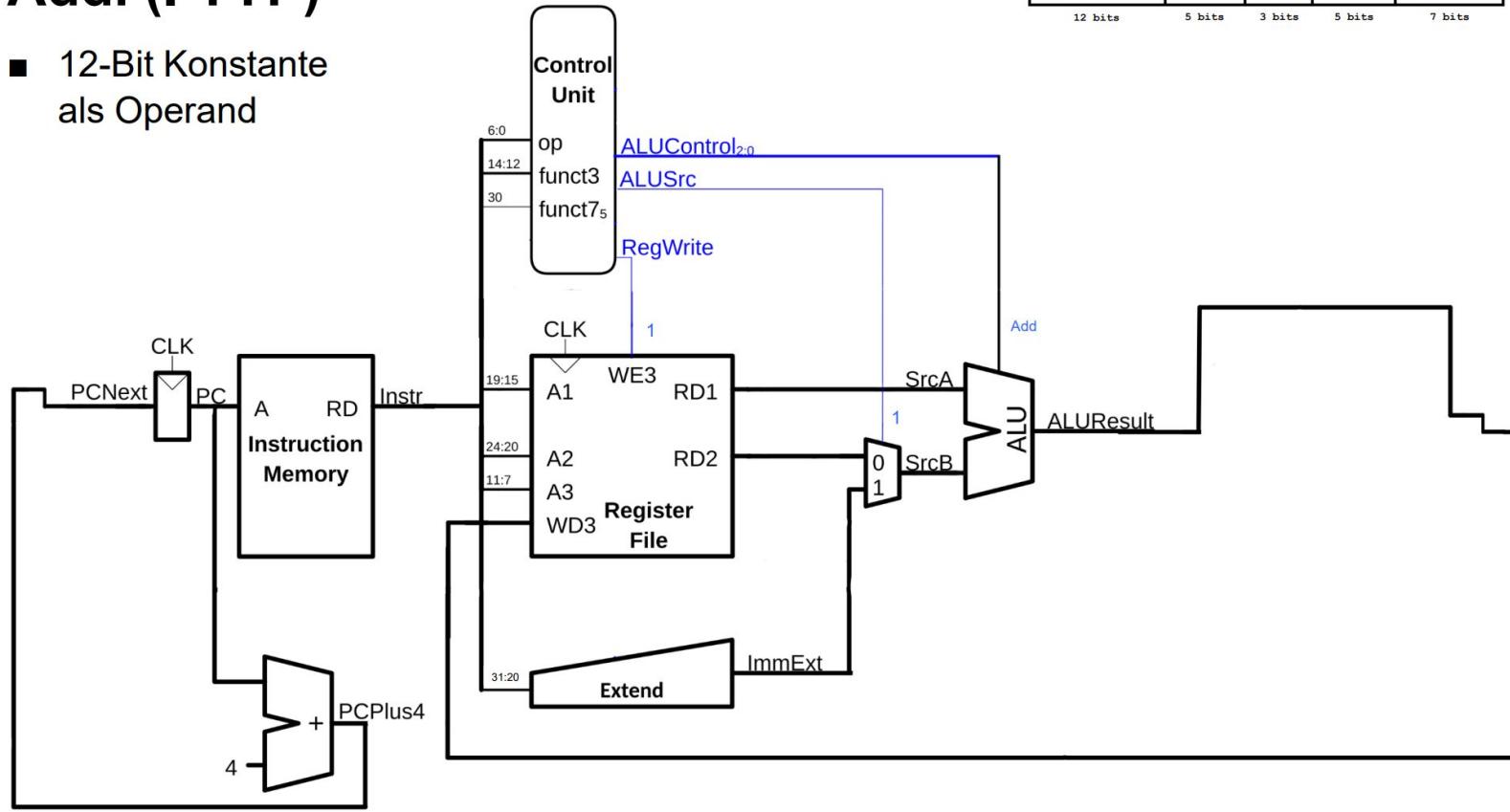
R-TYP

- PC erhöhen



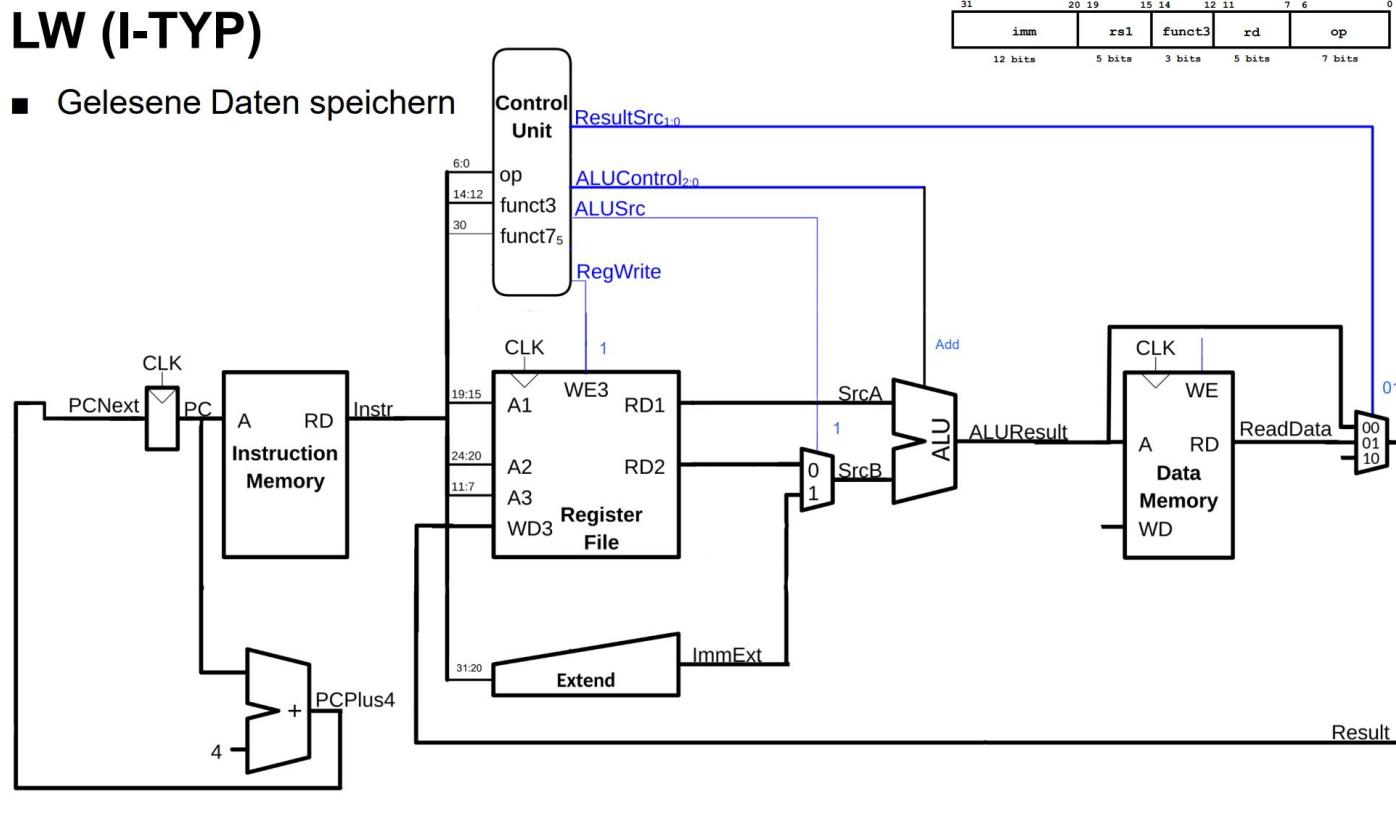
Addi (I-TYP)

- 12-Bit Konstante als Operand



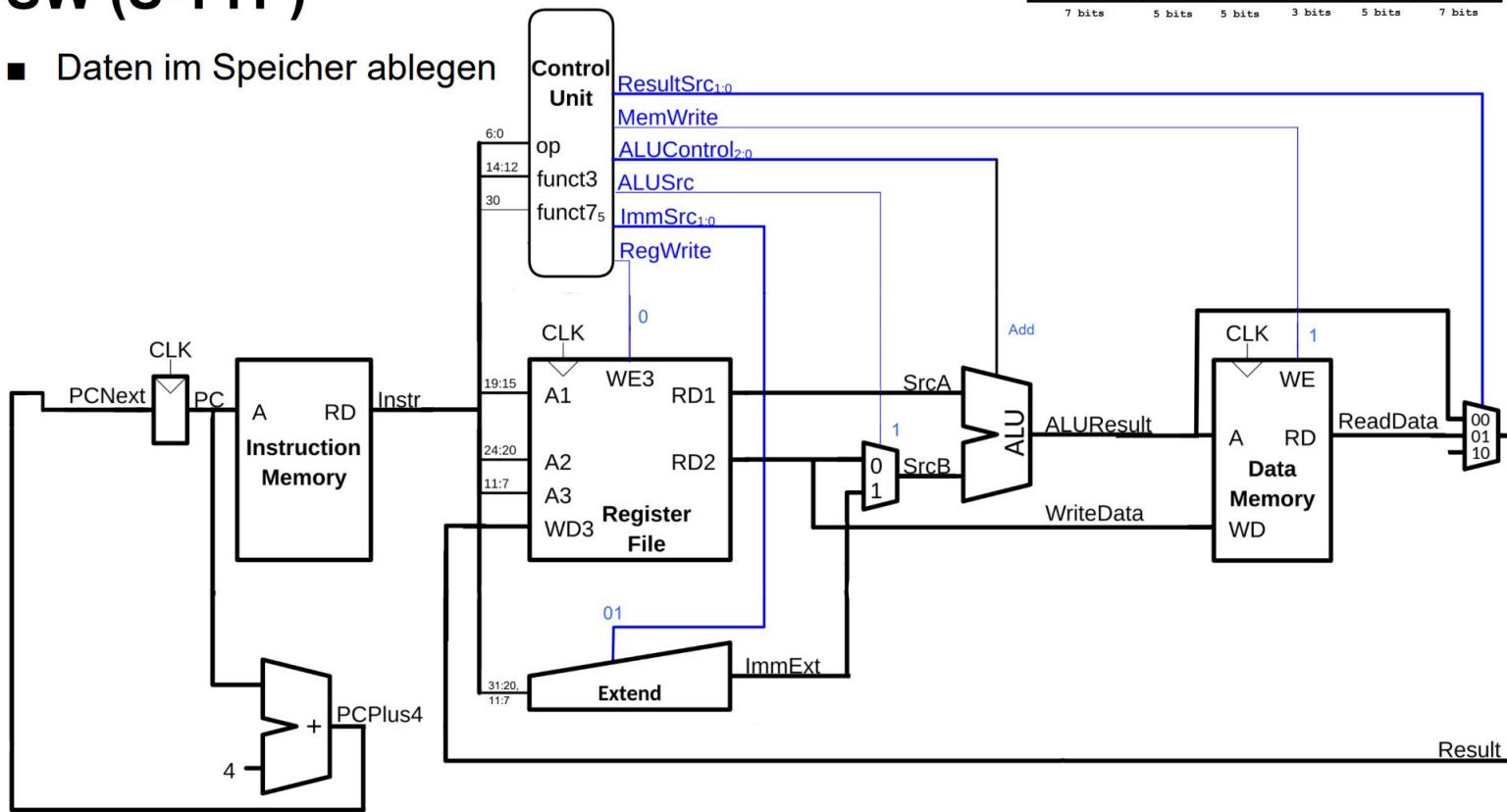
LW (I-TYP)

- Gelesene Daten speichern



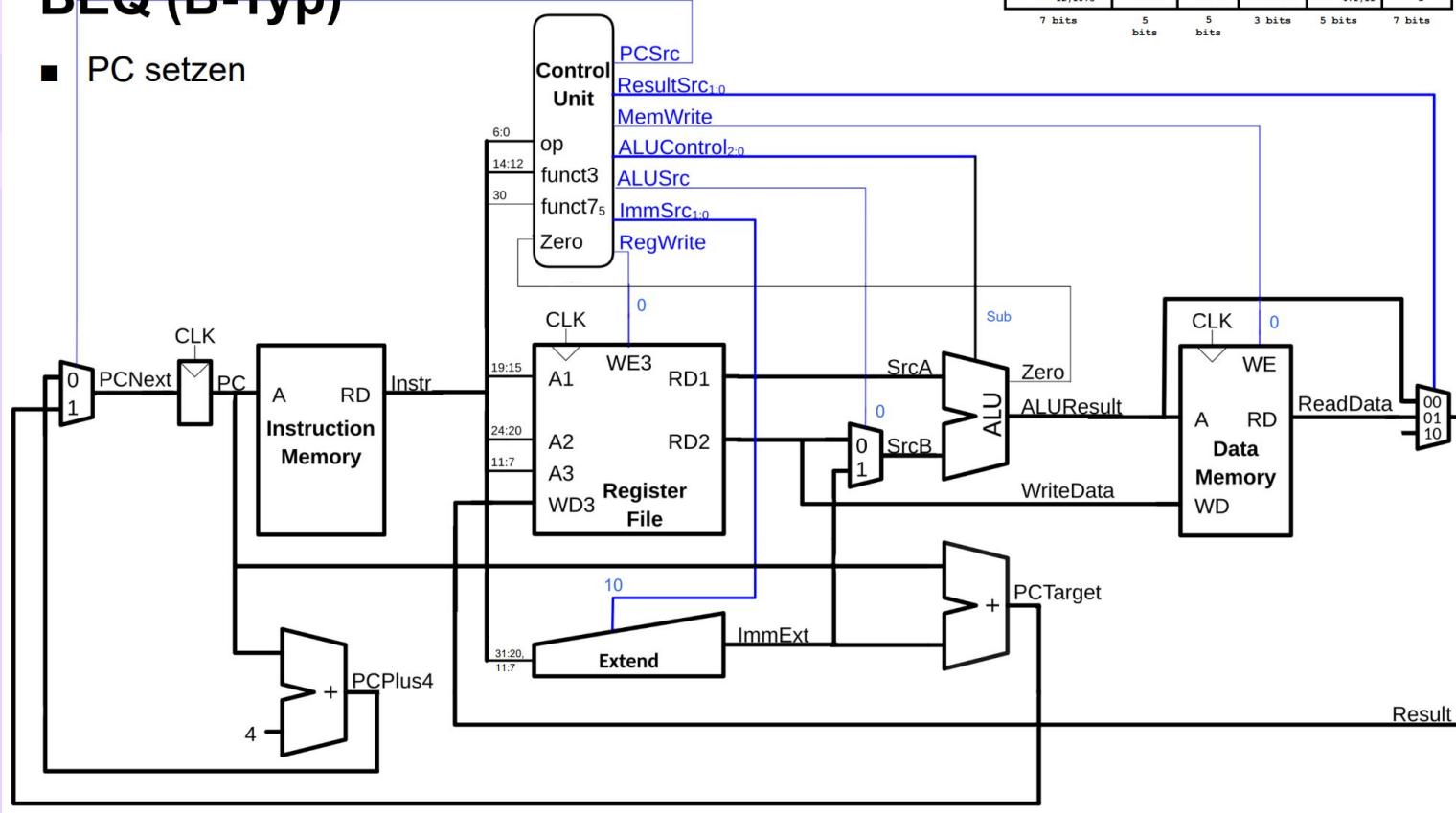
SW (S-TYP)

- Daten im Speicher ablegen



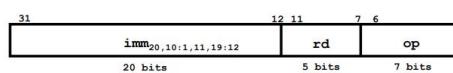
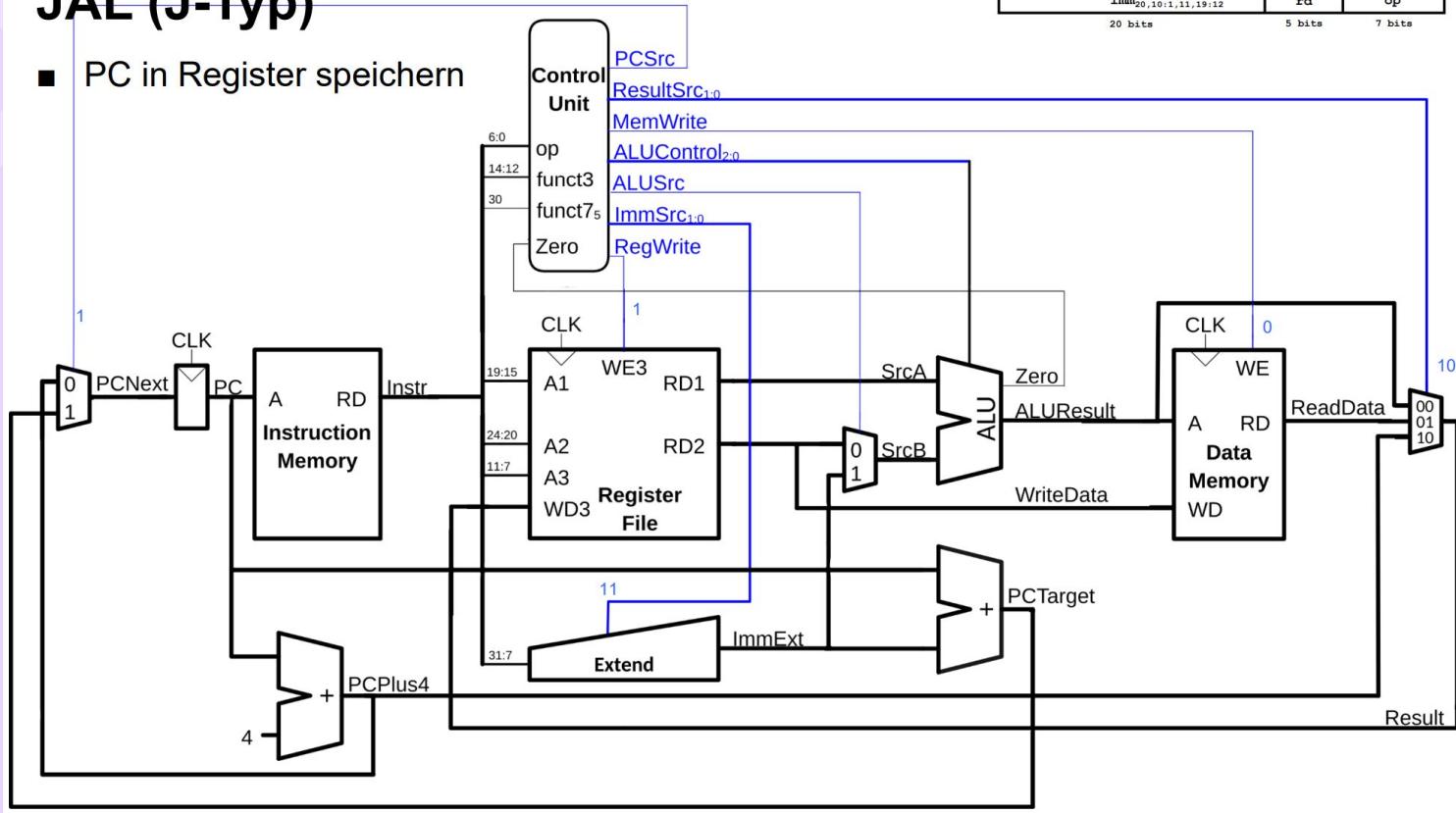
BEQ (B-Typ)

- PC setzen

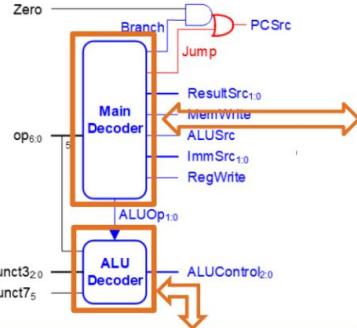


JAL (J-Typ)

- PC in Register speichern



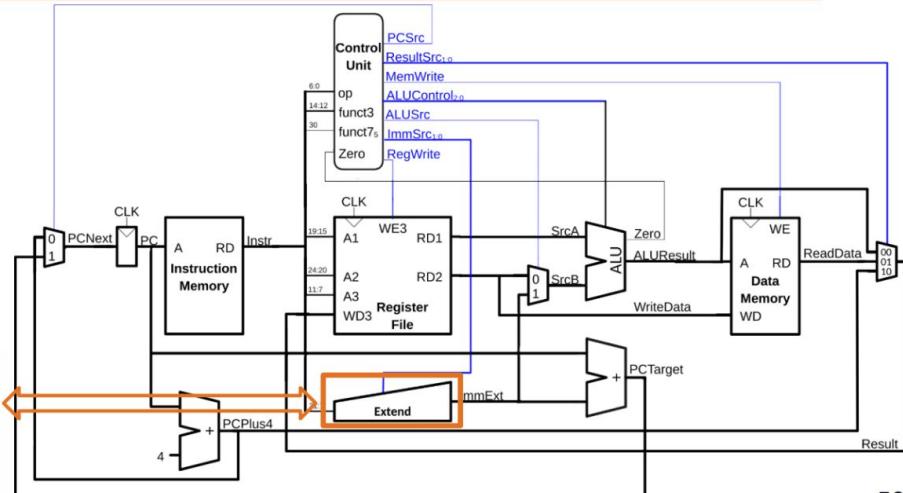
Zusammenfassung: Single-cycle RISC-V



Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
lw	0000011	1	00	1	0	01	0	00	0
sw	0100011	0	01	1	1	xx	0	00	0
R-type	0110011	1	xx	0	0	00	0	10	0
beq	1100011	0	10	0	0	xx	1	01	0
I-type ALU	0010011	1	00	1	0	00	0	10	0
jal	1101111	1	11	x	0	10	0	xx	1

ALUOp	funct3	{op _s , funct7 _s }	ALUControl	Instruction
00	x	x	000 (add)	lw, sw
01	x	x	001 (subtract)	beq
10	000	00, 01, 10	000 (add)	add
		11	001 (subtract)	sub
010	x	101 (set less than)	slt	
110	x	011 (or)	or	
111	x	010 (and)	and	

ImmSrc	ImmExt	Type	Description
00	[(20[Instr[31]]), Instr[31:20]]	I	12-bit signed immediate
01	[(20[Instr[31]]), Instr[31:25], Instr[11:7]]	S	12-bit signed immediate
10	[(20[Instr[31]]), Instr[7], Instr[30:25], Instr[11:8], 1'b0]	B	13-bit signed immediate
11	[(12[Instr[31]]), Instr[19:12], Instr[20], Instr[30:21], 1'b0]	J	21-bit signed immediate



- Übersetze das folgende RISC-V Assembly Programm in RISC-V Maschinensprache

```
add s7, s8, s9
sll t1, t2, t3
srli s3, s1, 14
sw s9, 16(t4)
```

0000000 1100111000000101110110011
0000000 11100000111001001100110011

0000000 1100100101100110 010011

0000000 1100101101010100000100011

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	op	R-Type
imm _{11:0}		rs1	funct3	rd	op	I-Type
imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op	S-Type
imm _{12:10:5}	rs2	rs1	funct3	imm _{4:1,11}	op	B-Type
imm _{31:12}				rd	op	U-Type
imm _{20,10:1,11,19:12}				rd	op	J-Type
fs3	funct2	fs2	fs1	funct3	fd	R4-Type
5 bits	2 bits	5 bits	5 bits	3 bits	5 bits	7 bits

0x013c0bb3

0x01c38333

0x00e4d883

0x018ea823

Konvertiere den folgenden RISCV Maschinencode zurück in RISC-V Assembly:

0x01200513
0x00300593
0x00000393
0x00058e33

0x01c54863
0x00138393
0x00be0e33
0xff5ff06f
0x00038533

31:25	24:20	19:15	14:12	11:7	6:0		
funct7	rs2	rs1	funct3	rd	op	R-Type	
imm _{11:0}		rs1	funct3	rd	op	I-Type	
imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op	S-Type	
imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op	B-Type	
imm _{31:12}				rd	op	U-Type	
imm _{20,10:1,11,19:12}				rd	op	J-Type	
fs3	funct2	fs2	fs1	funct3	fd	op	R4-Type



Erkläre was das Programm aus der vorherigen Aufgabe berechnet.

t3	t2
3	0
6	1
9	2
12	3
15	4
18	5
21	6

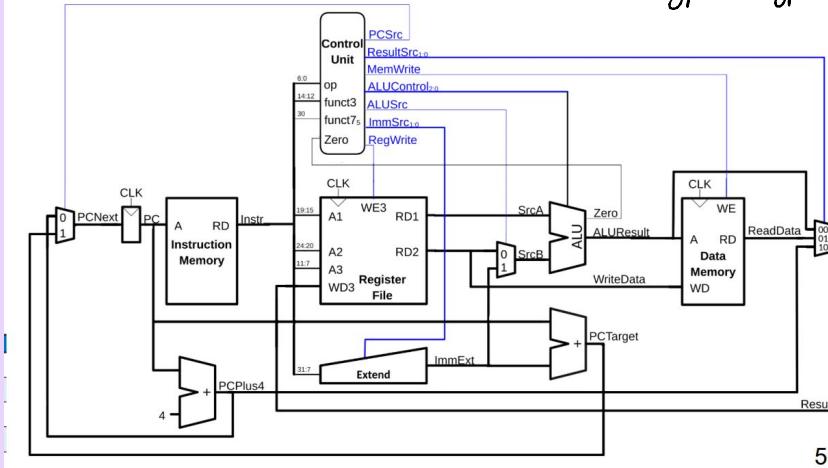
Hieraus ergibt sich folgender RISC-V Assembly:

```
_start:  
    addi a0, zero, 18  
    addi a1, zero, 3  
    addi t2, zero, 0  
    add t3, a1, zero  
  
loop_start:  
    blt a0, t3, loop_end  
    addi t2, t2, 1  
    add t3, t3, a1  
    jal zero, loop_start  
  
loop_end:  
    add a0, t2, zero
```

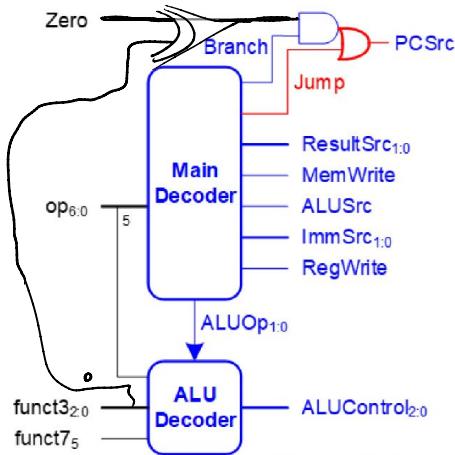
Angenommen der Prozessor in Abbildung 2 hat fehlerhafte Kontrollsignale, deren Wert konstant auf einen bestimmten Wert gelegt ist. Die folgende Tabelle listet Kontrollsignale und deren konstanten Wert auf. Welche der Befehle, die in der Vorlesung vorgestellt wurden, funktionieren nicht mehr unter dieser fehlerhaften Belegung?

Kontrollsignal	Konstante Belegung	Fehlerhafte Befehle
ALUControl	000 (ADD)	BEQ, R-Typ super ADD, ALU
PCSrc	0	BEQ, JAL
ResultSrc	01	JAL, R-Typ, I-Typ ALU

1-Typ super ADD



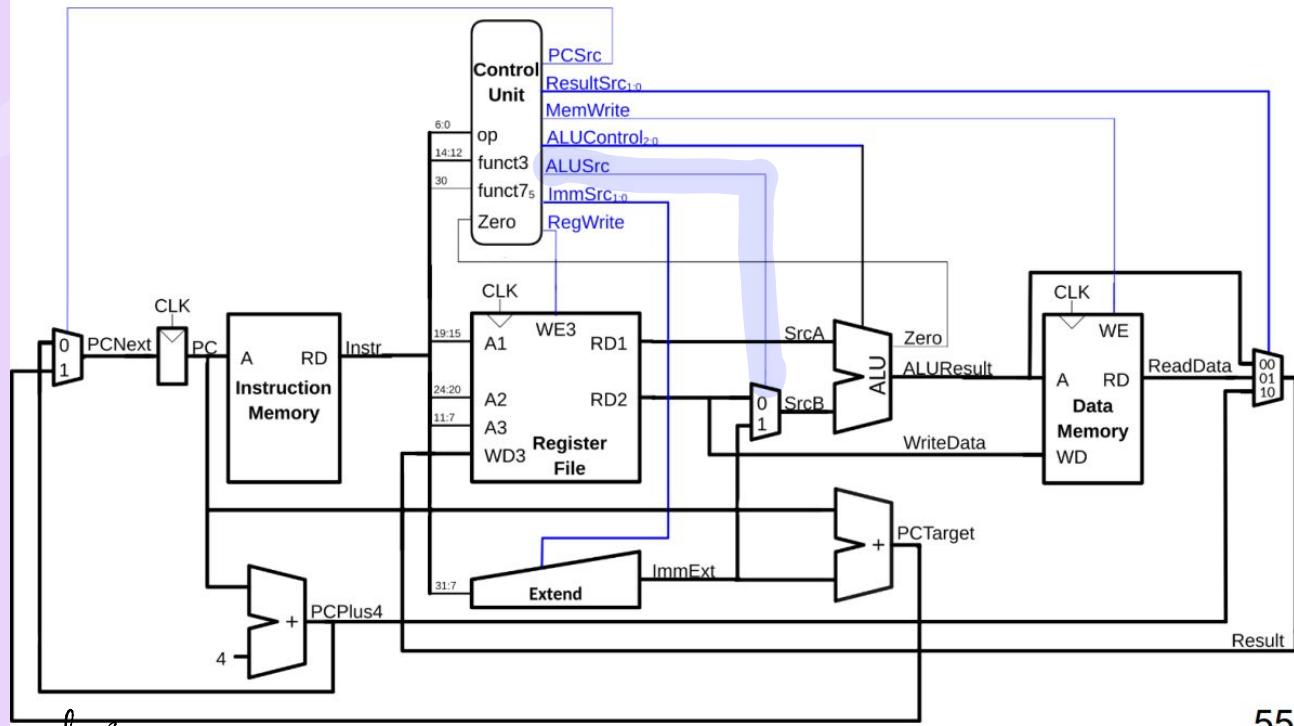
Das Steuerwerk des Prozessors sieht im genaueren wie folgt aus:



func3 ₀	zero	Branch
0	0	0
0	1	1
1	0	1
1	1	0

Insbesondere ist die Logik des **PCSrc** Signals genauer abgebildet: **PCSrc** wird auf 1 gesetzt wenn **Jump** auf 1 (bei JAL) ist oder ein Branch gemacht wird und das **Zero** Signal 1 liefert (bei BEQ).

Erweitere das Steuerwerk und gegebenenfalls das Prozessorschaltbild so, dass auch ein BNE (Branch Not Equal) Befehl durchgeführt werden kann. Der BNE Befehl vergleicht zwei Quellregister und addiert einen relativen Offset auf den PC falls die Register *nicht* gleich sind. Der Maschinencode des BNE ist in Abbildung 3 abgebildet.



<i>op</i>	<i>func3</i>							
1100011 (99)	000	-	B	beq	rs1, rs2, label	branch if =		if (rs1 == rs2) PC = BTA
1100011 (99)	001	-	B	bne	rs1, rs2, label	branch if ≠		if (rs1 ≠ rs2) PC = BTA

Gebe die Belegung der Kontrollssignale und den Befehlstyp für folgende Befehle an. Für die ALUControl reicht der Name der ALU-Instruktion (ADD, SUB, AND, ...). Gebe Signale bei denen die Belegung keinen Einfluss auf die Funktionalität hat explizit mit *Don't Care* ('x') an.

Befehl	Befehlstype	Branch	ResultSrc	MemWrite	ALUControl	ALUSrc	ImmSrc	RegWrite
OR	R	0	00	0	OR	0	X	1
BNE	B	1	X	0	SUB	0	10	0
XORI	I	0	00	0	XOR	1	00	1
SW	S	0	X	1	ADD	1	01	0

