

Zentralübung 8: Single-Cycle

1) Altklausuraufgabe (Retake 23/24)

Aufgabe 8 Single-Cycle RISC-V Prozessor (15 Punkte)

Betrachten Sie das Schaltbild des single-cycle RISC-V Prozessors in Abbildung 8.1.

a)* Vervollständigen Sie die Belegung der Kontrollsignale des Steuerwerks für die angegebenen Befehle in der untenstehenden Tabelle. Geben Sie *don't cares* explizit mit x an. Es ist ausreichend bei ALUControl die Funktion (ADD, SUB, ...) anzugeben. Nehmen Sie weiterhin an, dass das Extend-Bauteil wie folgt gesteuert wird:

Befehltyp	ImmSrc
I	00
S	01
B	10
J	11

Eine Befehlsreferenz für RISC-V Befehle finden Sie in Abbildung 10.1 auf Seite 23.

Befehl	Branch	Jump	ResultSrc	MemWrite	ALUControl	ALUSrc	ImmSrc	RegWrite
BEQ	1	X	X					
JAL	X	1	10	0	X	X	11	1
ANDI								
SW								
OR	0	0	0	0	011	0	X	1

b)* Nehmen Sie an, dass durch einen Fehler in der Fertigung das Signal *ResultSrc* konstant auf 00 gesetzt ist. Welche der folgenden Befehle funktionieren dadurch nicht mehr einwandfrei?

☒ jal

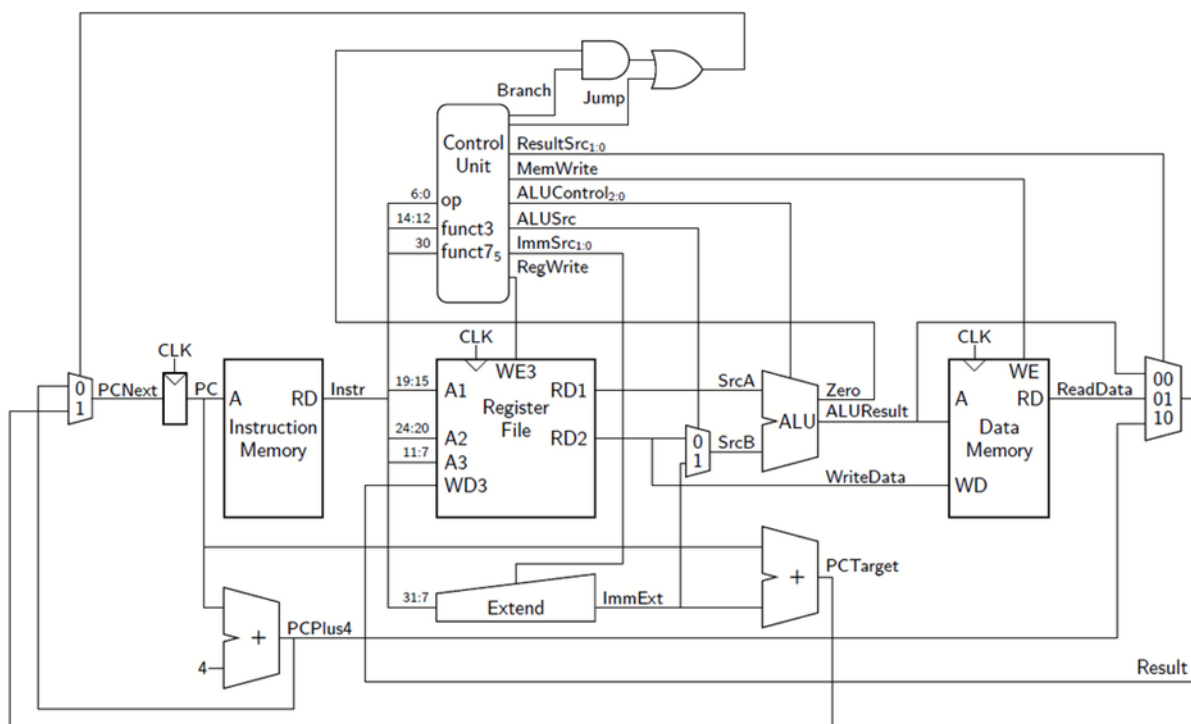
☐ sw

☒ lw

☐ addi

☐ or

☐ beq



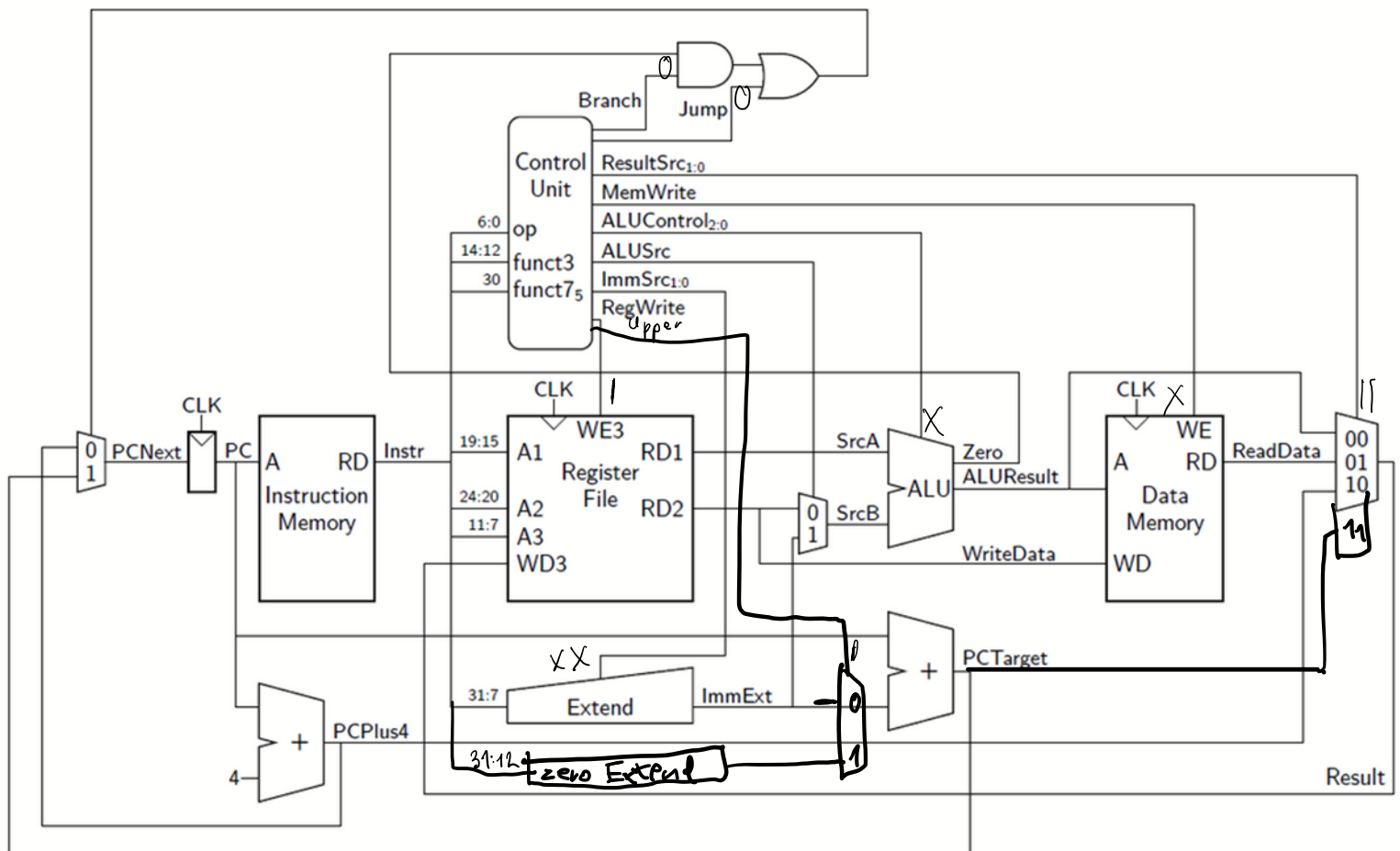
c)* Der Instruktionssatz des Prozessors soll um den Befehl `auipc` (add upper immediate to pc) erweitert werden. Der Befehl `auipc rd, upimm` addiert das mit Nullen zu einem 32-Bit Signal erweiterte `upimm` zu dem aktuellen Program Counter (bevor PC um 4 erweitert wird) und speichert das Ergebnis in `rd` (siehe Befehlsreferenz 10.1).

Erweitern Sie den Datenpfad des Prozessors in Abbildung 8.1 mit möglichst wenig Logik, sodass der Befehl `auipc` unterstützt wird. Tragen Sie auch die Werte aller Kontrollsignale des Steuerwerks an die entsprechenden Signale im Schaltbild ein. Geben Sie *don't cares* explizit mit X an. Sie können wenn nötig neue Kontrollsignale einführen. Benutzen Sie in diesem Fall beschreibende Signalbezeichner.

Die Extend Unit unterstützt keine U-Typ Befehle und darf für diese Aufgabe nicht verändert werden. Sie dürfen für diese Aufgabe ein Bauteil verwenden das eine 20-Bit Zahl als Eingabe erhält und zu einer 32-Bit Zahl erweitert indem die niederwertigsten Bits mit Nullen aufgefüllt werden.

Nehmen Sie an, dass die ALU die folgenden Befehle unterstützt:

ALUControl	Instruction
000 (Add)	$ALUResult = SrcA + SrcB$
001 (Sub)	$ALUResult = SrcA - SrcB$
101 (Set Less Than)	$ALUResult = 1 \text{ if } SrcA < SrcB \text{ else } ALUResult = 0$
011 (Or)	$ALUResult = SrcA \vee SrcB$ (Bitweises Oder)
010 (And)	$ALUResult = SrcA \wedge SrcB$ (Bitweises Und)



Bonus: Keine neuen Kontrollsignale erlaubt

<div> <div>31:25</div> <div>24:20</div> <div>19:15</div> <div>14:12</div> <div>11:7</div> <div>6:0</div> </div>							
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
5 bits	2 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

- imm: signed immediate in imm_{11:0}
- uimm: 5-bit unsigned immediate in imm_{4:0}
- upimm: 20 upper bits of a 32-bit immediate, in imm_{31:12}
- Address: memory address: rs1 + SignExt(imm_{11:0})
- [Address]: data at memory location Address
- BTA: branch target address: PC + SignExt((imm_{12:1}, 1'b0))
- JTA: jump target address: PC + SignExt((imm_{20:1}, 1'b0))
- label: text indicating instruction address
- SignExt: value sign-extended to 32 bits
- ZeroExt: value zero-extended to 32 bits
- csr: control and status register

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	000	-	I	lb rd, imm(rs1)	load byte	rd = SignExt([Address] _{7:0})
0000111 (3)	001	-	I	lh rd, imm(rs1)	load half	rd = SignExt([Address] _{15:0})
0001011 (3)	010	-	I	lw rd, imm(rs1)	load word	rd = [Address] _{31:0}
0001111 (3)	100	-	I	lbu rd, imm(rs1)	load byte unsigned	rd = ZeroExt([Address] _{7:0})
0010011 (3)	101	-	I	lhu rd, imm(rs1)	load half unsigned	rd = ZeroExt([Address] _{15:0})
0010011 (19)	000	-	I	addi rd, rs1, imm	add immediate	rd = rs1 + SignExt(imm)
0010011 (19)	001	0000000*	I	slli rd, rs1, uimm	shift left logical immediate	rd = rs1 << uimm
0010011 (19)	010	-	I	slti rd, rs1, imm	set less than immediate	rd = (rs1 < SignExt(imm))
0010011 (19)	011	-	I	sltiu rd, rs1, imm	set less than imm. unsigned	rd = (rs1 < SignExt(imm))
0010011 (19)	100	-	I	xori rd, rs1, imm	xor immediate	rd = rs1 ^ SignExt(imm)
0010011 (19)	101	0000000*	I	srlr rd, rs1, uimm	shift right logical immediate	rd = rs1 >> uimm
0010011 (19)	101	0100000*	I	srai rd, rs1, uimm	shift right arithmetic imm.	rd = rs1 >>> uimm
0010011 (19)	110	-	I	ori rd, rs1, imm	or immediate	rd = rs1 SignExt(imm)
0010011 (19)	111	-	I	andi rd, rs1, imm	and immediate	rd = rs1 & SignExt(imm)
0010111 (23)	-	-	U	auipc rd, upimm	add upper immediate to PC	rd = {upimm, 12'b0} + PC
0100011 (35)	000	-	S	sb rs2, imm(rs1)	store byte	[Address] _{7:0} = rs2 _{7:0}
0100011 (35)	001	-	S	sh rs2, imm(rs1)	store half	[Address] _{15:0} = rs2 _{15:0}
0100011 (35)	010	-	S	sw rs2, imm(rs1)	store word	[Address] _{31:0} = rs2
0110011 (51)	000	0000000	R	add rd, rs1, rs2	add	rd = rs1 + rs2
0110011 (51)	000	0100000	R	sub rd, rs1, rs2	sub	rd = rs1 - rs2
0110011 (51)	001	0000000	R	sll rd, rs1, rs2	shift left logical	rd = rs1 << rs2 _{4:0}
0110011 (51)	010	0000000	R	slt rd, rs1, rs2	set less than	rd = (rs1 < rs2)
0110011 (51)	011	0000000	R	sltu rd, rs1, rs2	set less than unsigned	rd = (rs1 < rs2)
0110011 (51)	100	0000000	R	xor rd, rs1, rs2	xor	rd = rs1 ^ rs2
0110011 (51)	101	0000000	R	srl rd, rs1, rs2	shift right logical	rd = rs1 >> rs2 _{4:0}
0110011 (51)	101	0100000	R	sra rd, rs1, rs2	shift right arithmetic	rd = rs1 >>> rs2 _{4:0}
0110011 (51)	110	0000000	R	or rd, rs1, rs2	or	rd = rs1 rs2
0110011 (51)	111	0000000	R	and rd, rs1, rs2	and	rd = rs1 & rs2
0110111 (55)	-	-	U	lui rd, upimm	load upper immediate	rd = {upimm, 12'b0}
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
1100011 (99)	100	-	B	blt rs1, rs2, label	branch if <	if (rs1 < rs2) PC = BTA
1100011 (99)	101	-	B	bge rs1, rs2, label	branch if ≥	if (rs1 ≥ rs2) PC = BTA
1100011 (99)	110	-	B	bltu rs1, rs2, label	branch if < unsigned	if (rs1 < rs2) PC = BTA
1100011 (99)	111	-	B	bgeu rs1, rs2, label	branch if ≥ unsigned	if (rs1 ≥ rs2) PC = BTA
1100111 (103)	000	-	I	jalc rd, rs1, imm	jump and link register	PC = rs1 + SignExt(imm), rd = PC + 4
1101111 (111)	-	-	J	jalc rd, label	jump and link	PC = JTA, rd = PC + 4

*Encoded in instr_{31:25}, the upper seven bits of the immediate field

2) Maschinensprache

```

_start:
    addi t0, zero, 3
    addi t1, zero, 5
    auipc t2, 5
    sub t2, t1, t0
loop_start:
    addi t0, t0, 1
    beq t1, t0, loop_end
    jal zero, loop_start
loop_end:
    sw t2, 100(t1)
    lw t3, 105(zero)
    ebreak

```

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0-2	x5-7	Temporary registers
s0/fp	x8	Saved register / Frame pointer
s1	x9	Saved register
a0-1	x10-11	Function arguments / Return values
a2-7	x12-17	Function arguments
s2-11	x18-27	Saved registers
t3-6	x28-31	Temporary registers

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
5 bits	2 bits	5 bits	5 bits	3 bits		5 bits		7 bits				

op	funct3	funct7	Type	Instruction
0000011 (3)	000	-	I	lb rd, imm(rs1)
0000011 (3)	001	-	I	lh rd, imm(rs1)
0000011 (3)	010	-	I	lw rd, imm(rs1)
0000011 (3)	100	-	I	lbu rd, imm(rs1)
0000011 (3)	101	-	I	lhu rd, imm(rs1)
0010011 (19)	000	-	I	addi rd, rs1, imm
0010011 (19)	001	0000000*	I	slli rd, rs1, uimm
0010011 (19)	010	-	I	slti rd, rs1, imm
0010011 (19)	011	-	I	sltiu rd, rs1, imm
0010011 (19)	100	-	I	xori rd, rs1, imm
0010011 (19)	101	0000000*	I	srlr rd, rs1, uimm
0010011 (19)	101	0100000*	I	srai rd, rs1, uimm
0010011 (19)	110	-	I	ori rd, rs1, imm
0010011 (19)	111	-	I	andi rd, rs1, imm
0010111 (23)	-	-	U	auipc rd, upimm
0100011 (35)	000	-	S	sb rs2, imm(rs1)
0100011 (35)	001	-	S	sh rs2, imm(rs1)
0100011 (35)	010	-	S	sw rs2, imm(rs1)
0110011 (51)	000	0000000	R	add rd, rs1, rs2
0110011 (51)	000	0100000	R	sub rd, rs1, rs2
0110011 (51)	001	0000000	R	sll rd, rs1, rs2
0110011 (51)	010	0000000	R	slt rd, rs1, rs2
0110011 (51)	011	0000000	R	sltu rd, rs1, rs2
0110011 (51)	100	0000000	R	xor rd, rs1, rs2
0110011 (51)	101	0000000	R	srl rd, rs1, rs2
0110011 (51)	101	0100000	R	sra rd, rs1, rs2
0110011 (51)	110	0000000	R	or rd, rs1, rs2
0110011 (51)	111	0000000	R	and rd, rs1, rs2
0110111 (55)	-	-	U	lui rd, upimm
1100011 (99)	000	-	B	beq rs1, rs2, label
1100011 (99)	001	-	B	bne rs1, rs2, label
1100011 (99)	100	-	B	blt rs1, rs2, label
1100011 (99)	101	-	B	bge rs1, rs2, label
1100011 (99)	110	-	B	bltu rs1, rs2, label
1100011 (99)	111	-	B	bgeu rs1, rs2, label
1100111 (103)	000	-	I	jalr rd, rs1, imm
1101111 (111)	-	-	J	jal rd, label

*Encoded in instr_{31:25}, the upper seven bits of the immediate field

3) auipc-Erweiterung in Digital

