Projekt_aufgabe1

Sonntag, 4. Juni 2023 14:45

Aufgabe 1 – Konzeptionelle Erweiterung des Prozessors (14 Punkte)

Der in der Vorlesung vorgestellte RISC-V Prozessor muss erweitert werden damit er die Popcount Berechnungen unterstützt. Hierzu müssen die Instruktionen **slli**, **auipc**, **blt** und **jalr** implementiert werden. Erweitere dazu in Abbildung 5 für jeden dieser Befehle (falls nötig) den Datenpfad und füge weitere Kontrollsignale hinzu.[1] Gib für jeden dieser Befehle auch den Wert aller Kontrollsignale an. Nimm dabei an, das die ALU aus Abbildung 4 verwendet wird und gib don't cares explizit mit x an.
Die Abgabe beinhaltet ein Bild mit dem erweitertem Daten- und Kontrollpfad des Prozessors pro zusätzlichem Befehl (es können auch mehrere Erweiterungen in ein Bild gezeichnet werden) sowie eine Tabelle mit den Werten aller Kontrollsignale für die neuen Befehle. Hinweis: Entnimm die Semantik der einzelnen RISC-V Befehle der in Moodle zur Verfügung gestellten Befehls-Referenz.

slli	rd, rs1, imm	shift left logical imm.	I	0010011	001	0000000^{c}	rd = rs1 << $imm_{[1:0]}$

ALUOP	funct3	{op(5), funct7(5)}	Instr.	ALUControl2:0
00	x	x	lw, sw	000 (add)
01	x	x	beq	001 (subtract)
10	000	00,01,10	add	000 (add)
10	000	11	sub	001 (subtract)
10	010	x	slt	101 (set less than)
10	110	x	or	011 (or)
10	111	x	and	010 (and)
10	001	x	slli	100 (shift)

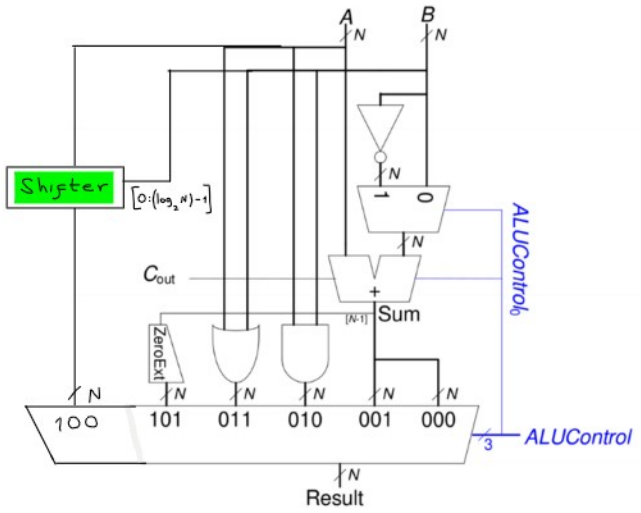


Abbildung 4: ALU des Single Cycle RISC-V Prozessors

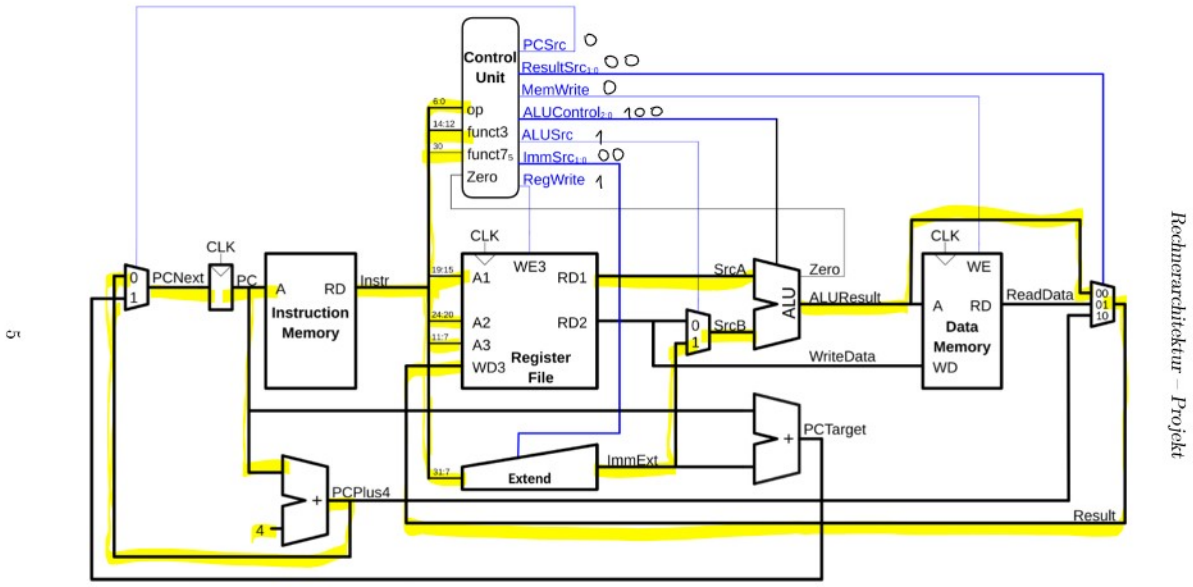


Abbildung 5: Schaltbild des Single Cycle RISC-V Prozessors

op	Intr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	ALUControl	Jump
19	slli	1	000	1	0	00	0	10	100	0

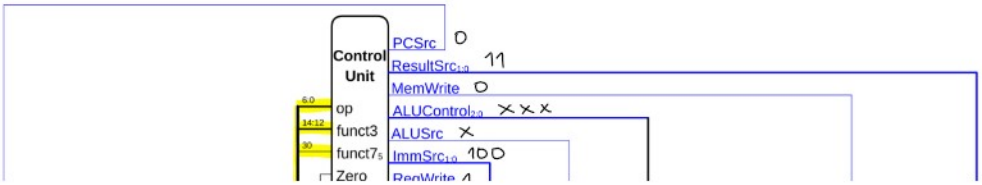
auipc	rd, upimm	add upper imm. to PC	U	0010111	–	–	rd = PC + (imm << 12)

Core Instruction Formats

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
$imm_{[11:0]}$		rs1	funct3	rd	opcode	I-type
$imm_{[11:5]}$	rs2	rs1	funct3	$imm_{[4:0]}$	opcode	S-type
$imm_{[12,10:5]}$	rs2	rs1	funct3	$imm_{[4:1,11]}$	opcode	B-type
$imm_{[31:12]}$				rd	opcode	U-type
$imm_{[20,10:1,11,19:12]}$				rd	opcode	J-type
7bit	5bit	5bit	3bit	5bit	7bit	

ImmSrc	ImmExt	Intrs.Type
000	{{20{instr[31]}}, instr[31:20]}	I
001	{{20(instr[31]}}, instr[31:25], instr[11:7]}	S
010	{{19(instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B
011	{{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0}	J
100	{{20{instr[31]}}, instr[31:12], 12'b0}	U

last 12 bits are 0



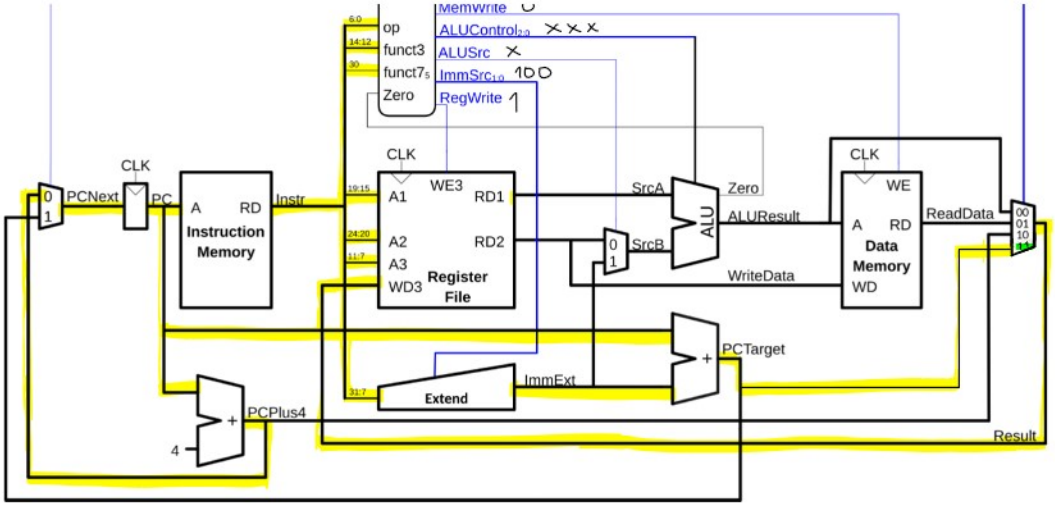


Abbildung 5: Schaltbild des Single Cycle RISC-V Prozessors

op	Intr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	ALUControl	Jump
19	slli	1	000	1	0	00	0	10	100	0
23	auipc	1	100	x	0	11	0	xx	xxx	0

blt	rs1, rs2, label	branch <	B	1100011	100	–	if(rs1 < rs2) PC += imm

Negative == 1

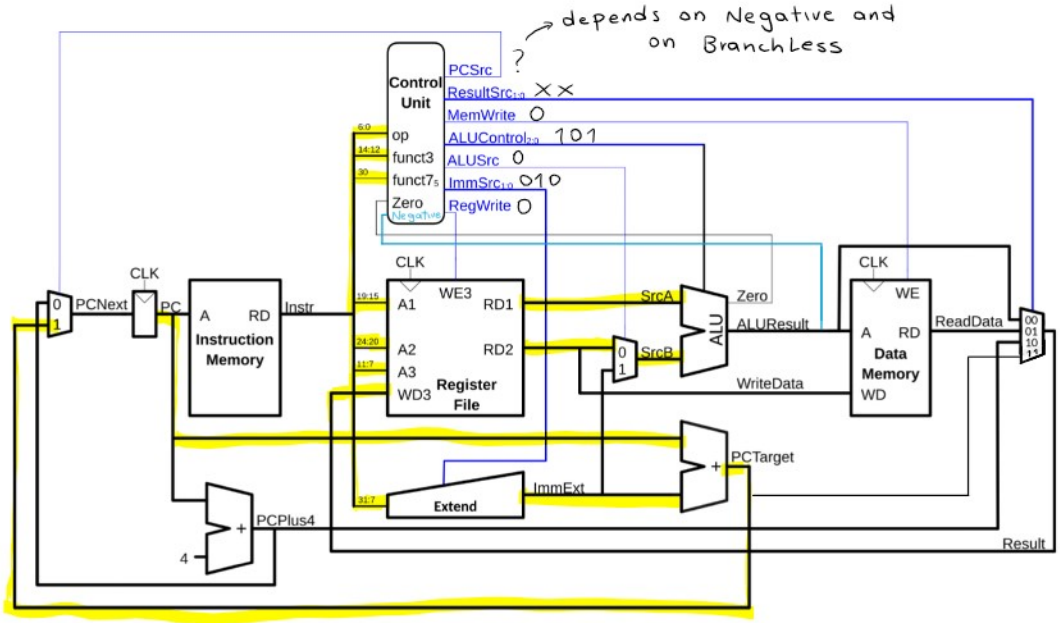
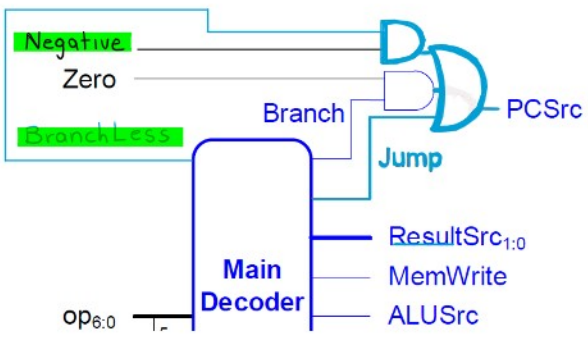
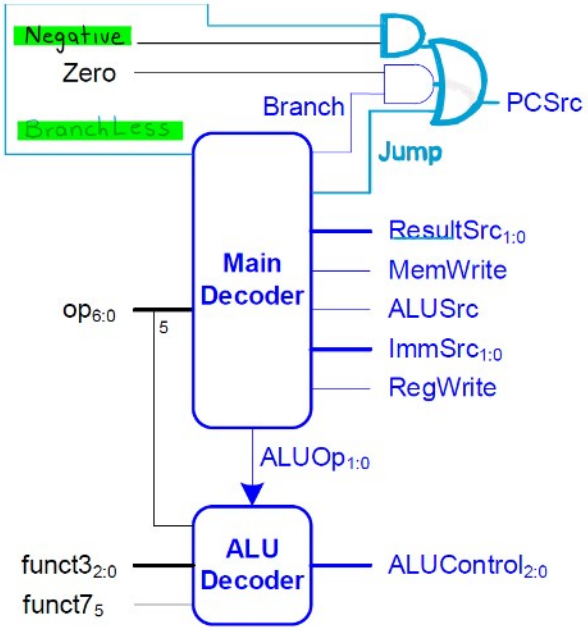


Abbildung 5: Schaltbild des Single Cycle RISC-V Prozessors





op	Intr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	BranchLess	ALUOp	ALUControl	Jump
19	slli	1	000	1	0	00	0	0	10	100	0
23	auipc	1	100	x	0	11	0	0	xx	xxx	0
99	blt	0	010	0	0	xx	0	1	10	101	0

jalr	rd, rs1, imm	jump and link reg	I	1100111	000	–	rd = PC+4; PC = rs1 + imm

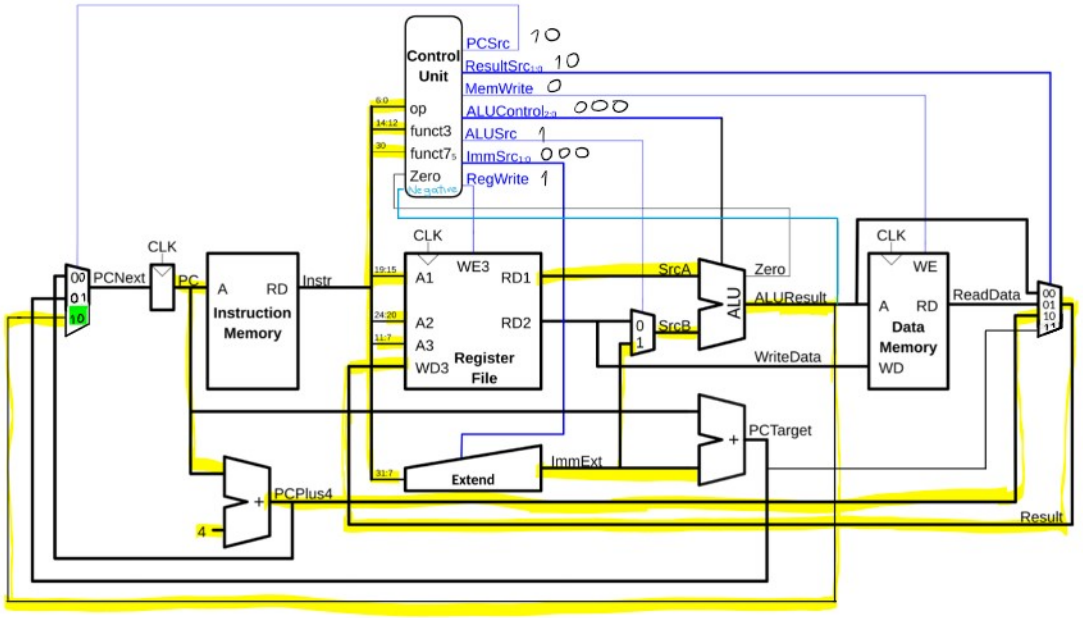
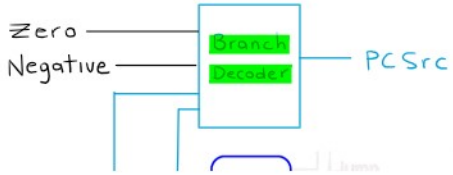
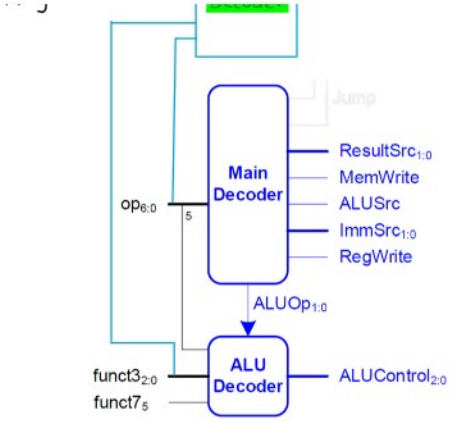


Abbildung 5: Schaltbild des Single Cycle RISC-V Prozessors





op	Intr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	ALUControl
19	slli	1	000	1	0	00	0	10	100
23	auipc	1	100	x	0	11	0	xx	xxx
99	blt	0	010	0	0	xx	0	10	101
103	jalr	1	000	1	0	10	0	00	000