Institute for Complex Systems

JOHANNES KEPLER UNIVERSITÄT LINZ

Univ.-Prof. Dr. Daniel Große
Christoph Hazott, M.Sc. & Katharina Ruep, M.Sc.

27. April 2023

Projekt
zur Vorlesung Rechnerarchitektur

Abgabe bis **Mittwoch 7 Juni, 2023 10:15** via EPICS: `https://ep.ics.jku.at`.

Im Rahmen des Projekts soll der in der Vorlesung vorgestellte RISC-V Prozessor erweitert und in VHDL realisiert werden. Ziel ist, dass der Prozessor eine Funktion zur Berechnung der auf 1 gesetzten Bits in einem Byte (Popcount) ausführen kann. Um die Leistung des Prozessors zu verbessern soll im zweiten Teil des Projekts der Prozessor um eine *Custom Instruction* erweitert werden. Diese soll die Popcount Funktion auf allen 4 Bytes eines Wortes auf einmal berechnen.

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.

Aufgabe 2 – Implementierung in VHDL (24 Punkte)

Implementiere den erweiterten RISC-V Prozessor in VHDL. Verwende dazu die im Template vorgegebenen Strukturen und füge (wenn nötig) weitere Komponenten hinzu. Zu vervollständigende Abschnitte im VHDL-Code sind mit TODO gekennzeichnet (je nach Implementierung können aber auch andere Stellen betroffen sein). Durch das Erweitern des Prozessors kann es sein, dass sich die Schnittstelle einiger Komponenten ändern. Teste den erweiterten Prozessor, indem du das Programm in der Datei "popcount.txt"

[1] Der in schwarz gezeichnete Teil in Abbildung 2 visualisiert den Datenpfad, während der Kontrollpfad in blau gezeichnet ist.

ausführst. Nachdem die Simulation gestoppt ist, sollten in den Speicherzellen folgende Werte stehen: *Mem[128] = 0*, *Mem[129] = 8*, *Mem[130] = 3*, *Mem[131] = 4*. Schreibende Zugriffe auf den Speicher werden durch die Testbench in der Ausgabe geloggt. Eine entsprechende Testbench[2] ist bereits im Template verfügbar.
Hinweis: Bei der Simulation in EPICS müssen auch die .txt Programmdateien ausgewählt werden. Zusätzlich kann es Sinn machen eigene kleine Testprogramme zu schreiben. Diese können mit der "Dump" Funktion im Venus Simulator in eine hexadezimal Darstellung übersetzt werden die durch den Single Cycle Prozessor ausgeführt werden kann.

Aufgabe 3 – Beschleunigung mit einer Custom Instruction (10 Punkte)

Um die Berechnung zu beschleunigen soll nun eine Custom Instruction im Prozessor implementiert werden. Diese soll die Popcount Funktion für all Bytes eines 32-Bit breiten Registers berechnen und als Ergebnis ein Wort zurückgeben. Die 4 Bytes des Ergebnisses sollen jeweils dem Ergebnis der Popcount Funktion auf dem entsprechendem Byte der Eingabe entsprechen. Die Spezifikation der Custom Instruction ist in 1 angegeben und ein Beispiel dafür ist in 2 zu sehen.

$$\begin{aligned} regs[rd] = & popcount(regs[rs1](31 \text{ downto } 24)) \\ & \& \; popcount(regs[rs1](23 \text{ downto } 16)) \\ & \& \; popcount(regs[rs1](15 \text{ downto } 8)) \\ & \& \; popcount(regs[rs1](7 \text{ downto } 0)) \end{aligned}$$

Abbildung 1: Spezifikation der Custom Instruction

Die Custom Instruktion soll unter dem opcode *"0001011"* in den Prozessor integriert werden und dem Instruktionsschema in Abbildung 3 folgen. Zum testen der Instruktion liegt das Programm "custom.txt" bei, dass bei korrekter implementierung der Custom Instruction den Wert $0x03070005$ in Register *x10* schreiben sollte.

[2]Die testbench testet sowohl Aufgabe 2 als auch Aufgabe 3.

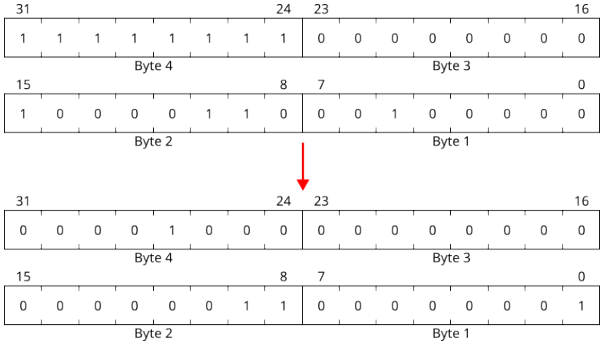

Abbildung 2: Beispiel für eine Anwendung der Custom Instruction

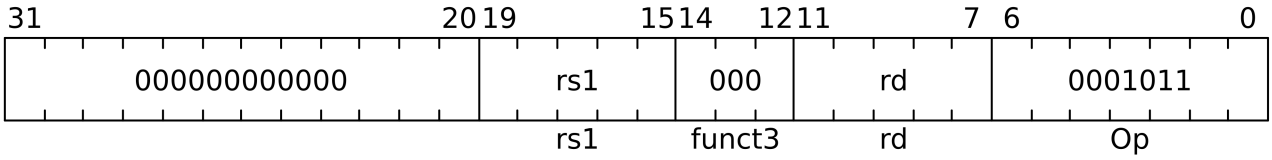

Abbildung 3: Instruktionsschema der Custom Instruction

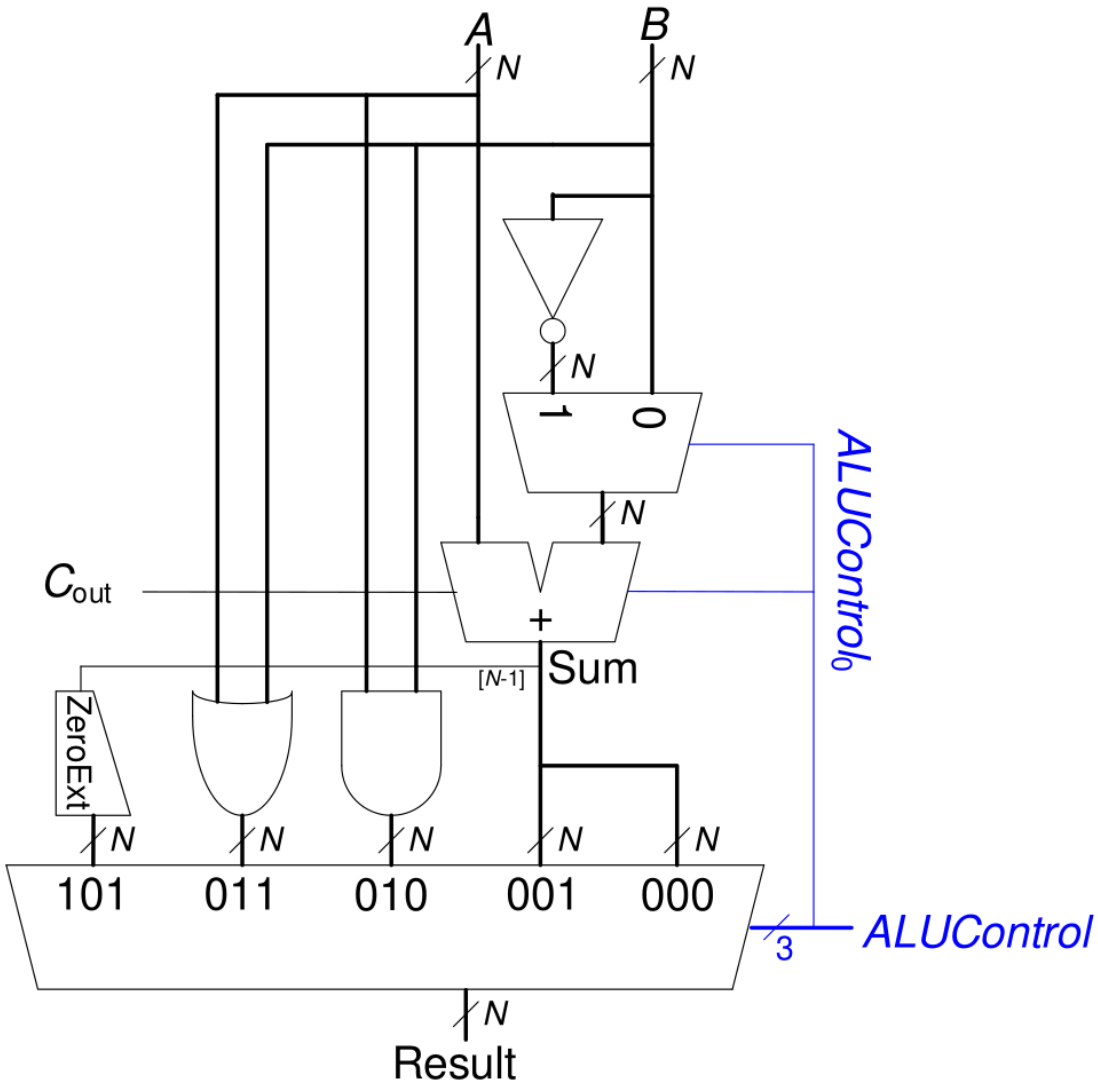

Abbildung 4: ALU des Single Cycle RISC-V Prozessors

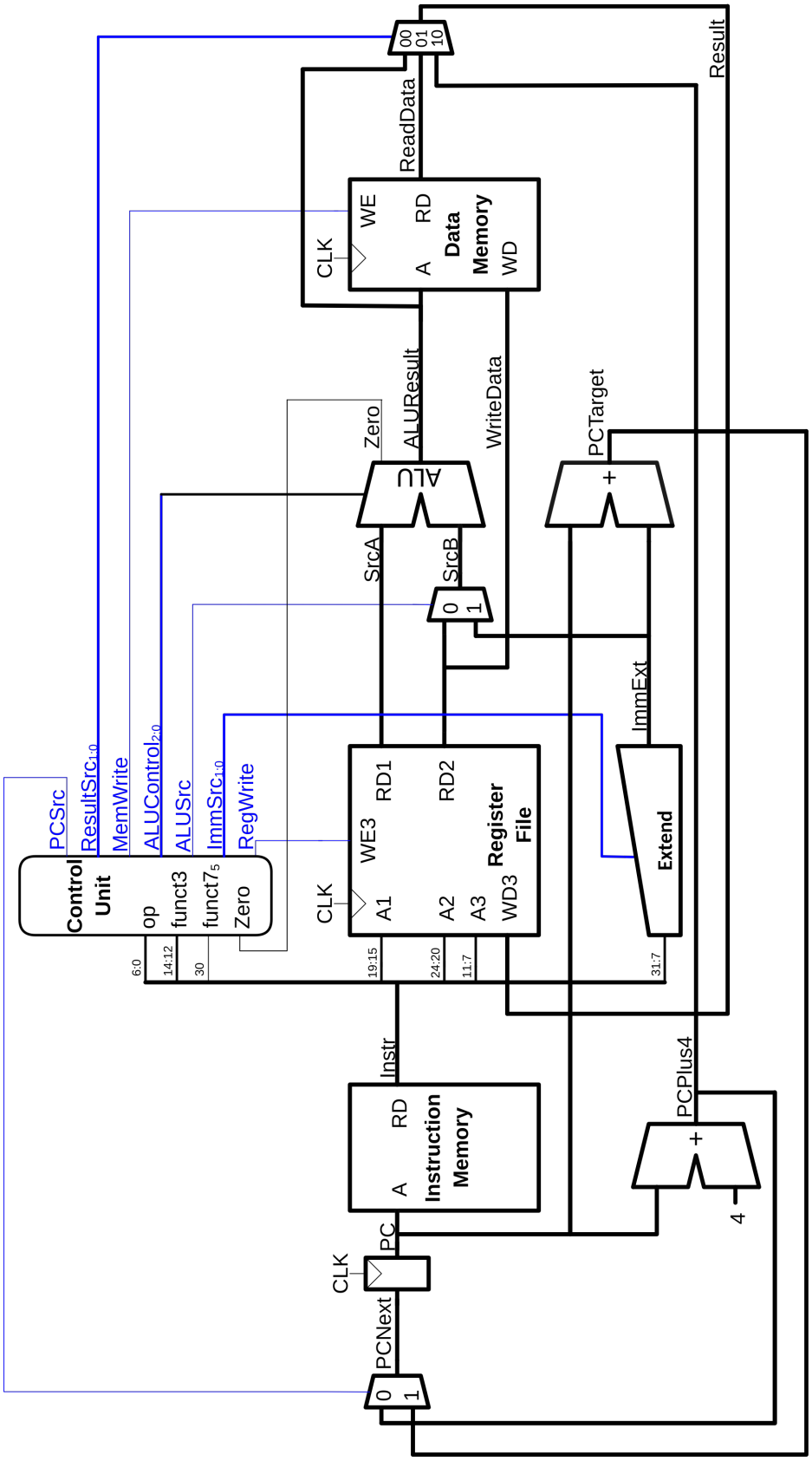

Abbildung 5: Schaltbild des Single Cycle RISC-V Prozessors