Bonus-Projekt: Implementierung eines einfachen RISC-V Emulators

Lehrveranstaltung „Rechnerarchitektur“

Sommersemester 2025

Zielsetzung

Im Rahmen dieses freiwilligen Einzelprojekts entwickeln Sie einen einfachen Emulator einer RISC-V CPU (RV32I) **ohne Pipeline**. Der Emulator soll grundlegende RISC-V Assemblerprogramme interpretieren und ausführen können.

Lernziele

- Vertieftes Verständnis der RISC-V Instruktionsarchitektur (RV32I)
- Praktische Erfahrung mit dem Fetch–Decode–Execute-Zyklus
- Einblick in Speicherorganisation und Registerverwaltung
- Umgang mit Assembler-Programmierung und maschinennaher Simulation

Projektumfang - muss

- Implementierung eines RV32I-Emulators in einer Sprache Ihrer Wahl (z.B. Python, C, Java)
- Unterstützung folgender Instruktionsklassen:
  - Arithmetisch/logische Instruktionen (`add`, `sub`, `and`, `or`, `xor`)
  - Immediate-Instruktionen (`addi`, `andi`, `ori`)
  - Speicherzugriffe (`lw`, `sw`)
  - Verzweigungen (`beq`, `bne`)
  - Sprünge (`jal`, `jalr`)
- Umsetzung eines einfachen Registersatzes (32 Register, inkl. `x0`)
- Hauptspeicherverwaltung (z.B. als Array implementiert)
- Debug-Ausgabe (z.B. Registerzustand pro Schritt)
- Memory-Dump (selektiver dump): bestimmte Regionen oder belegte Regionen

- Interaktive Kommandozeile (Step-by-step) ✓
- Einlesen und Parsen einfacher Assemblerprogramme als Eingabe: Implementierung eines einfachen Assemblers ✓ Eingabe als txt oder oneliner

Eingabeformat

Der Emulator soll Assemblerprogramme im Klartext einlesen und verarbeiten. Beispiel:

```
addi x1, x0, 5
addi x2, x0, 10
add x3, x1, x2
beq x3, x2, Ziel
```

RISC-V ISA: Übersicht

Architektur	32 Bit (RV32I)
Register	32 allgemeine Register (x0–x31), x0 ist immer 0
Speicher	Load/Store-Architektur, byte-adressiert
Instruktionslänge	32 Bit (fixed width)
Adressierung	Register-indirekt, PC-relativ
Datentypen	Integer (RV32I: nur Ganzzahlen)
Quelltext	RISC-V Assembly (kein Binärcode)

Offizielle Dokumentation: `https://riscv.org/technical/specifications/`

Optionale Aspekte - kann

- Unterstützung weiterer Instruktionen (`lui`, `auipc`, `ecall`, `ebreak`)
- Graphische Visualisierung
- Schätzung der Leistung durch Zuordnung eines CPI-Wertes zu jedem Befehl und Aufsummierung der benötigten Takte für ein Programm.

Rahmenbedingungen

- Bearbeitung als Einzelarbeit
- Abgabe bis spätestens: **10. Juli 2025**
- Abgabe umfasst:
  - Quellcode mit kurzer Dokumentation = README ✓
  - Ausführung der zur Verfügung gestellten Beispielprogramme mit Kommentaren → txt einlesen + mit Debug ausführen
  - Mindestens ein eigenes Beispielprogramm ✓
  - Kurzer Erfahrungsbericht ✓

WICHTIGE Anmerkungen!

- Sie dürfen eine beliebige Programmiersprache verwenden, um Ihren Simulator zu implementieren.
- Sie können ebenfalls einen Jupyter-Notebook (`https://jupyter.org/`) benutzen, um Code und Erklärungen zu kombinieren.
- Falls Sie `C/C++` oder eine andere kompilierte Programmiersprache verwenden, stellen Sie sicher, dass das ausführbare Programm auf dem Terminal erstellt werden kann. Reichen Sie **keine** Projekt-Dateien wie z.B. von Microsoft Visual Studio o.ä. Das Projekt ist einfach genug, dass es mit `gcc` direkt erstellt werden kann.
- `gcc` ist auch für Windows erhältlich: `https://sourceforge.net/projects/mingw/`, `https://de.wikipedia.org/wiki/GNU_Compiler_Collection`

Bewertung und Bonuspunkte

Bei erfolgreicher Bearbeitung des Projekts können bis zu **10 Punkte** für die Klausurpunktzahl erreicht werden. Die Bewertung erfolgt anhand von Funktionalität, Vollständigkeit, Codequalität, Testbarkeit und Präsentation der Ergebnisse.

Kontakt

Bei Fragen: melden Sie sich per Email