Rechnerarchitektur
Sommersemester 25

ertis embedded real-time systems ulm university institute

Laborprojekt – Versuch 6

In Ihrem letzten Versuch haben Sie eine auf Immediate- und Register-Befehle beschränkte

RISC-V-Architektur mit Pipelining umgesetzt. In diesem Versuch erweitern Sie Ihre bestehende Architektur um die sogenannten U-Befehle.

In der RISC-V-Architektur bezieht sich U auf Befehle, die für die Verarbeitung von »unbeschränkten« (englisch: *unbounded) Datentypen optimiert sind. Konkret handelt es sich dabei um die U-Befehle LUI (Load Upper Immediate), AUIPC (Add Upper Immediate to PC) und JAL (Jump and Link). Zusätzlich betrachten wir in diesem Versuch den Befehl JALR (Jump and Link Register), obwohl dieser offiziell zu den I-Befehlen gehört.

Ziele des 6. Versuchs

- Konzeptionierung, Umsetzung und Validierung der U-Befehle LUI, AUIPC, JAL sowie des I-Befehls JALR.

Vorbereitung

Spezifikation RISC-V (R-, I- und U-Befehle)

Eignen Sie sich den Ablauf, die Phasen und die Ansteuerung für die Ausführung von U-Befehlen basierend auf der Vorlesung und der Referenz (RV32I Base Integer Instruction Set, Version 2.0) an. Verstehen Sie die Befehlscodierung und Steuerung so gut, dass Sie diese mithilfe der Referenzkarten für jede Phase erklären können. Erarbeiten Sie ein Konzept zur Umsetzung der U-Befehle in der RISC-V-Architektur.

GHDL-Standards

Machen Sie sich mit den GHDL-Optionen vertraut, insbesondere zur Einstellung des Sprachstandards. Ab sofort sind alle Komponenten und Testbenches ausschließlich mit VHDL 2008 zu verwenden.

Aufgaben

Aufgabe 1: LUI (Load Upper Immediate)

Der Befehl LUI lädt das 20-Bit-Immediate des Sign-Extenders in die oberen 20 Bits des Zielregisters. Die unteren 12 Bits werden durch den Sign-Extender bereits auf null gesetzt. Um diesen Befehl in die bestehende Architektur einzubinden, müssen Sie den Decoder anpassen und das Testsystem um zusätzliche Register und einen Multiplexer nach der ALU erweitern. Im *Types_Package* ist das Steuersignal *WB_SEL* zu ergänzen.

- Erweitern Sie den Decoder so, dass er den Befehl *LUI_OP_INS* erkennt und das Steuerwort gemäß dem uFormat setzt. Der Befehl *LUI* wird mit einer Additionsoperation umgesetzt, wobei *I_IMM_SEL* auf $'1'$ und *WB_SEL* auf $"01"$ gesetzt wird. *WB_SEL* ist 2 Bit breit und steuert, welche Daten in der WB-Phase ins Zielregister geschrieben werden: ALU-Ergebnis (00) oder Immediate (01).
- Kopieren Sie den Inhalt Ihrer Datei *ri_only_RISC_V.vhdl* in *riu_only_RISC_V.vhdl*.
- Sparen Sie Register, indem Sie nach dem Sign-Extender eine `when ... select`-Abfrage einfügen, die je nach OP-Code das passende Immediate weiterleitet. Erweitern Sie diese Abfrage sukzessive.
- Erweitern Sie das Testsystem um eine Pipeline-Register-Folge, die das Immediate bis zur Write-Back-Phase vorhält.
- Ergänzen Sie die Testbench um einen 4:1-Multiplexer. Der erste Eingang erhält das ALU-Ergebnis, der zweite das Immediate. Der Selektor wird über *WB_SEL* angesteuert. Der dritte und vierte Eingang führen zunächst Nullen.
- Testen Sie Ihr System erfolgreich mit dem ersten Test der Testbench (*ri_only_RISC_V_tb.vhdl*).
- Erstellen Sie ein Bash-Skript im Ordner *<Projektordner>/Skripte/RISCV*, das die Simulation inklusive Analyse und Elaboration automatisiert und erfolgreich durchführt.

Aufgabe 2: AUIPC (Add Upper Immediate to PC)

AUIPC fügt ein 20-Bit-Immediate zum aktuellen Befehlszähler (PC) hinzu und speichert das Ergebnis im Zielregister. Die unteren 12 Bits des Immediates werden auf null gesetzt. Zur Umsetzung müssen Sie den Decoder anpassen und das Testsystem um Register und einen Multiplexer erweitern. Das Steuersignal *A_SEL* wählt dabei den zweiten ALU-Operand aus (rs1 oder PC+4).

- Der Decoder muss den Befehl *AUIPC_OP_INS* erkennen und das Steuerwort im uFormat anpassen. *AUIPC* nutzt eine Additionsoperation, wobei *I_IMM_SEL* und *A_SEL* auf 1 gesetzt werden.
- Fügen Sie eine Pipeline-Register-Folge ein, die PC+4 bis zur EX-Phase speichert.
  Hinweis: Das Abgreifen von PC aus späteren Phasen ist fehleranfällig. Eine mögliche Lösung ist ein separater Addierer, der den PC konstant um 4_{10} erhöht.
- Ergänzen Sie einen Multiplexer mit *A_SEL* zur Steuerung von ALU-Operand 1 (OP1). Der erste Eingang bleibt der ursprüngliche ALU-Operand, der zweite erhält PC+4.
- Erweitern Sie die Immediate-Auswahl hinter dem Sign-Extender um den OP-Code von AUIPC.
- Testen Sie Ihr System erfolgreich mit dem ersten und zweiten Test der Testbench (*ri_only_RISC_V_tb.vhdl*).
- Passen Sie ggf. das Bash-Skript im Ordner *<Projektordner>/Skripts/RISCV* an und führen Sie es erfolgreich aus.

Aufgabe 3: JAL (Jump and Link)

JAL springt relativ zur aktuellen Adresse (PC) um einen J-Immediate-Offset und speichert PC+4 im Zielregister. Er wird oft zur Implementierung von Funktionsaufrufen genutzt.

- Der Decoder muss *JAL_OP_INS* erkennen und das Steuerwort entsprechend im uFormat anpassen. Die Signale *I_IMM_SEL*, *A_SEL* und *PC_SEL* werden auf 1, *WB_SEL* auf 10 gesetzt.
- Ergänzen Sie eine Pipeline-Register-Folge für PC+4 bis zur WB-Phase.
- Erweitern Sie den 4:1-Multiplexer aus Aufgabe 1 um den dritten Eingang, der nun PC+4 erhält.
- Ergänzen Sie vor dem PC einen Multiplexer, der über *PC_SEL* gesteuert wird. Eingang 1: Ergebnis des Addierers, Eingang 2: ALU-Ausgabe (MEM-Phase).
- Ergänzen Sie die Immediate-Auswahl nach dem Sign-Extender um den JAL-OP-Code. Beachten Sie, dass JAL zur U-Gruppe zählt, aber ein J-Immediate nutzt.
- Testen Sie Ihr System erfolgreich mit den ersten drei Test der Testbench (*ri_only_RISC_V_tb.vhdl*).
- Passen Sie ggf. das Bash-Skript im Ordner *<Projektordner>/Skripts/RISCV* an und führen Sie es erfolgreich aus.

Aufgabe 4: JALR (Jump and Link Register)

JALR springt zu einer Zieladresse, die sich aus Registerwert und einem I-Immediate ergibt. Zusätzlich wird PC+4 im Zielregister gespeichert.

- Der Decoder muss *JALR_OP_INS* erkennen und das Steuerwort im iFormat setzen. JALR nutzt eine Additionsoperation, wobei *I_IMM_SEL*, *PC_SEL* auf 1 und *WB_SEL* auf 10 gesetzt werden.
- Ergänzen Sie ggf. die Immediate-Auswahl um den OP-Code von JALR.
- Testen Sie Ihr System erfolgreich mit allen vier Tests der Testbench (*ri_only_RISC_V_tb.vhdl*).
- Passen Sie ggf. das Bash-Skript im Ordner *<Projektordner>/Skripts/RISCV* an und führen Sie es erfolgreich aus.

Abnahme *Funktion der RISC-V-Architektur für R-, I- und U-Befehle*, Programmierkonventionen, Spezifikation gemäß RISC-V-Standard.

Aufgabe 5: Abgabe

Laden Sie die erstellte Ordnerstruktur mit allen zugehörigen Dateien als ZIP-Datei unter dem Namen *Name_Vorname_Versuch6.zip* in Moodle hoch.