Rechnerarchitektur
Sommersemester 24

ertis embedded real-time systems ulm university institute

Laborprojekt – Versuch 5

In Ihrem letzten Versuch haben Sie eine auf Immediate- und Register-Befehle beschränkte RISC-V -Architektur mit Pipelining umgesetzt. In diesem Versuch erweitern Sie die 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: ünbounded") Datentypen optimiert sind. Speziell handelt es sich hierbei 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), wobei der Befehl JALR aber eigentlich zu den I-Befehlen gehört.

Ziele des 5. Versuchs

- Konzeptionierung, Umsetzung und Validierung des U-Befehls LUI
- Konzeptionierung, Umsetzung und Validierung des U-Befehls AUIPC
- Konzeptionierung, Umsetzung und Validierung des U-Befehls JAL
- Konzeptionierung, Umsetzung und Validierung des I-Befehls JALR

Vorbereitung

Spezifikation RISC-V (R/I/U-Befehle)

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

GHDL-Standards

Lesen Sie sich in die GHDL-Optionen, insbesondere wie Sie Standards der Sprache einstellen, ein. Ab sofort werden alle Komponenten und Testbenches nur noch mit VHDL 2008 genutzt.

Aufgaben

Aufgabe 1: LUI (Load Upper Immediate)

Der Befehl LUI (Load Upper Immediate) lädt das 20-Bit-Immediate des Sign-Extenders in die oberen 20 Bits des Zielregisters. Die unteren fehlenden 12 Bits des Registers werden bereits über den Sign-Extender auf null gesetzt. Um diesen Befehl in die bereits bestehende Architektur einzubinden, muss der Decoder entsprechend angepasst und das Testsystem über zusätzliche Register und einem Multiplexer nach der ALU, erweitert werden. Für die Ansteuerung des Selektor muss zusätzlich das *Types_Package* um das Steuersignal *WB_SEL* erweitert werden.

- Ersetzen Sie die bestehenden Packages durch die aktualisierten (udn).
- Erweitern Sie Ihren Decoder so das er erkennt, dass es sich um eine *LUI_OP_INS* handelt und das Steuerwort entsprechend über das uFormat ändert. Der Befehl *LUI* wird mit einer Addition-Operation ausgeführt und nur das Signal *I_IMM_SEL* wird dazu auf 1 und *WB_SEL* auf 01 gesetzt. Der Selektor *WB_SEL* ist 2-Bit breit und steuert die Daten, welche in der WB-Phase in das Zielregister geschrieben werden sollen. Für die Auswahl des Ergebnisses der ALU muss dieser auf 00 und für das Zurückschreiben des Immediates auf 01 gesetzt werden.
- Kopieren Sie sich den Inhalt ihrer Datei *r_i_only_RISC_V_tb.vhdl* in die Datei *riu_only_RISC_V_tb.vhdl*.
- Um Register zu sparen, fügen Sie nach dem Sign-Extender einen Abfrage (*when ...select*) ein, die abhängig vom aktuellen OP-Code das benötigte Immediate weiterleitet (erweitern Sie sukzessive die benötigten OP-Codes). Erweitern Sie nun das Testsystem um eine Pipeline-Register-Folge, die das benötigte Immediate des aktuellen Befehls bis zur Write-Back-Phase vorhalten kann.
- Erweitern Sie Ihre Testbench jetzt um einen Multiplexer (4zu1) der als ersten Eingang den Ausgang des Registers der ALU in der WB-Phase erhält und als zweiten Daten-Eingang das Immediate aus dem Immediate-Register in der WB-Phase. Der Selektor wird über das Steuersignal *WB_SEL* in der WB-Phase angesteuert. Der dritte und vierte Eingang des Multiplexer können an dieser Stelle mit nullen angesteuert werden.

- Erweitern Sie ihre Testbench mit den in Code-Abschnitten für den IC in *Instruction1.txt* und Teile des Testskriptes in *test1.txt* (für LUI) abgebildeten Code-Abschnitten. Testen Sie abschließend ihr Testsystem erfolgreich.

- Erstellen Sie ein Bash-Skript im Ordner *<Projektordner>/Testbenches/RISCV*, mit dem Sie die Simulation der RISC-V für Register-Befehle inklusive der Analyse und Elaboration automatisch und erfolgreich ausführen können.

Aufgabe 2: AUIPC (Add Upper Immediate to PC)

Der Befehl AUIPC (Add Upper Immediate to PC) fügt ein 20-Bit-Immediate zum aktuellen Befehlszähler (PC) hinzu und speichert das Ergebnis in dem Zielregister. Wie LUI setzt dieser Befehl die unteren 12 Bits des Immediate auf null. Um diesen Befehl in die bereits bestehende Architektur einzubinden, muss der Decoder entsprechend angepasst und das Testsystem über zusätzliche Register und einen Multiplexer erweitert werden. Für die Ansteuerung dieses Selektors wird das Steuersignal *A_SEL* genutzt. Der Selektor *A_SEL* wählt aus, ob der Operand 2 der ALU über den Ausgang des Registerfiles (rs1) oder dem Programmzähler des nächsten Befehls (PC+4) angesteuert wird.

- Erweitern Sie Ihren Decoder so, dass dieser erkennt, ob es sich um eine *AUIPC_OP_INS* handelt und das Steuerwort entsprechend über das uFormat ändert. Der Befehl *AUIPC* wird mit einer Addition-Operation ausgeführt, die Signale *I_IMM_SEL* und *A_SEL* werden dazu jeweils auf 1 gesetzt.

- Erweitern Sie das Testsystem um eine weitere Pipeline-Register-Folge, die den aktuell erzeugten PC+4 bis zur Ausführungsphase (EX) vorhält.

- Erweitern Sie Ihr System um einen Multiplexer, der das Steuersignal *A_SEL* verwendet und den ersten Eingang der ALU (OP1) ansteuert. Dieser Multiplexer erhält als ersten Daten-Eingang den ursprünglichen Operanden der ALU und der zweite Daten-Eingang wird mit dem vorgehaltenen PC+4 der Pipeline-Stufe (Ausführungsphase) verbunden.

- Erweitern sie ggf. noch den OP-Code des Befehls AUIPC in Ihrem Entscheider nach dem Sign-Extender.

- Erweitern Sie ihre Testbench mit den in Code-Abschnitten in *Instruction2.txt* und Teile des Testskriptes in *test2.txt* (für LUI und AUIPC). Alternativ können Sie die Testbench auch auslagern. Testen Sie abschließend ihr Testsystem erfolgreich.

- Passen Sie ggf. das Bash-Skript im Ordner *<Projektordner>/Testbenches/RISCV* an und führen Sie dieses erfolgreich aus.

Aufgabe 3: JAL (Jump and Link)

Der Befehl JAL (Jump and Link) führt einen verzweigten Sprung, relativ zur aktuellen Adresse um einen Offset (U-Immediate) verschoben, durch und speichert die Adresse des nächsten Befehls (PC+4) im Zielregister und wird häufig verwendet, um Funktionen zu implementieren. Um diesen Befehl in die bereits bestehende Architektur einzubinden, muss der Decoder entsprechend angepasst und das Testsystem über zusätzliche Register und einen Multiplexer erweitert werden. Für die Ansteuerung dieses Selektors wird das Steuersignal *PC_SEL* genutzt.

- Erweitern Sie Ihren Decoder so das er erkennt, das es sich um eine *JAL_OP_INS* handelt und das Steuerwort entsprechend über das uFormat ändert. Der Befehl *JAL* wird in unserer Umsetzung mit einer Addition-Operation ausgeführt (welche keine Auswirkung hat), die Signale *I_IMM_SEL*, *A_SEL* und *PC_SEL* werden dazu jeweils auf 1 und *WB_SEL* auf 10 gesetzt.

- Erweitern Sie in Ihrem Testsystem die Pipeline-Register-Folge des PC+4 bis zur WB-Phase (WB).

- Erweitern Sie nun den 4zu1-Multiplexer aus der ersten Aufgabe so, dass der dritte Eingang den vorgehaltenen Wert des PC+4 der WB-Phase erhält. Der vierte Eingang des Multiplexer wird an dieser Stelle weiterhin mit nullen angesteuert.

- Erweitern Sie Ihr Testsystem um einen Multiplexer vor dem PC, welcher über das Signal *PC_SEL* gesteuert wird. Der erste Eingang wird dazu mit dem Ausgang des dazugehörigen Volladdierer zu Bestimmung der nächsten Adresse und der zweite Eingang mit dem Ergebnis der ALU aus der MEM-Phase angesteuert.

- Erweitern sie ggf. noch den OP-Code des Befehls JAL in Ihrem Entscheider nach dem Sign-Extender. Denken Sie bei Ihrer Umsetzung daran das der JAL-Befehl zwar offiziell zu den U-Typen gehört aber das Immediate ein J-Immediate ist.

- Erweitern Sie ihre Testbench mit den in *Instruction3.txt* und Teile des Testskriptes in *test3.txt* (für LUI, AUIPC und JAL) abgebildeten Code-Abschnitten. Alternativ können Sie die Testbench auch auslagern. Testen Sie abschließend ihr Testsystem erfolgreich.

- Passen Sie ggf. das Bash-Skript im Ordner *<Projektordner>/Testbenches/RISCV* an und führen Sie dieses erfolgreich aus.

Aufgabe 4: JALR (Jump and Link Register)

Der Befehl JALR (Jump and Link Register) führt einen verzweigten Sprung zu einer Sprungzieladresse durch, die sich aus der Summe des Registers und einem Offset (I-Immediate) ergibt. Der Befehl speichert zudem wie JAL die Adresse des nächsten Befehls im Zielregister.

- Erweitern Sie Ihren Decoder so, dass er erkennt, dass es sich um eine *JALR_OP_INS* handelt und das Steuerwort entsprechend über das iFormat ändert. Der Befehl *JARL* wird mit einer Addition-Operation ausgeführt; die Signale *I_IMM_SEL* und *PC_SEL* werden dazu jeweils auf 1 und *WB_SEL* auf 10 gesetzt.

- Erweitern sie ggf. noch den OP-Code des Befehls JALR in Ihrem Entscheider nach dem Sign-Extender.

- Erweitern Sie ihre Testbench mit den in *Instruction4.txt* und Teile des Testskriptes in *test4.txt* (für LUI, AUIPC, JAL und JALR) abgebildeten Code-Abschnitten. Testen Sie abschließend ihr Testsystem erfolgreich.

- Passen Sie ggf. das Bash-Skript im Ordner *<Projektordner>/Testbenches/RISCV* an und führen Sie dieses erfolgreich aus.

Abnahme *Funktion des RISC-V für R/I/U-Befehle*, Programmierkonventionen, RISC-V Spezifikation der R/I/U-Befehle.

Aufgabe 5: Abgabe

Laden Sie die erstellte Ordnerstruktur mit den dazugehörigen Dateien als Zip-Datei unter dem Namen

Name_Vorname_Versuch5.zip in Moodle hoch.