Praktikum zur Vorlesung

Prozessorarchitektur

SS 2018

Übungsblatt 8

Implementierung einer superskalaren Prozessorarchitektur

Abgabefrist: Mittwoch 13.06.2018, 10:00 Uhr

1.1. Einführung

Eine superskalare Prozessor-Architektur erlaubt die parallele Ausführung von Befehlen aus demselben Befehlsstrom in verschiedenen Funktionseinheiten. Dadurch kann in der Regel ein höherer Befehlsdurchsatz als mit einer skalaren Architektur erreicht werden. Der bisherige skalare Prozessor mit Pipeline soll nun zu einem 2-fach superskalaren Prozessor mit statischer Ablaufplanung erweitert werden.

1.2. Projektbeschreibung

In dieser Übung werden Sie eine superskalare Architektur basierend auf der Architektur von Übung 7 entwerfen. Sie modifizieren Ihren Entwurf aus Übung 7, indem Sie die Befehlspipeline duplizieren, um die superskalare Ausführung zu ermöglichen. Die Ausführung in der Pipeline eines superskalaren Prozessors wird in Abbildung 1.1. verdeutlicht.

	clk 1	*clk 2*	*clk 3*	*clk 4*	*clk 5*
i1	InF	ID	EX	WB	
i2	InF	ID	EX	WB	
i3		InF	ID	EX	WB
i4		InF	ID	EX	WB

Abbildung 1.1. Superskalare Ausführung.

Es werden immer zwei Operationen gleichzeitig aus dem Speicher geholt, die dann gemeinsam die Pipeline durchlaufen. Wie auch in der Pipeline der skalaren Architektur gibt es in der superskalaren Pipeline-Architektur Datenabhängigkeiten zu beachten. In unserem Beispiel gehen wir wieder davon aus, dass die Datenabhängigkeiten in der Übersetzungsphase des Programmes aufgelöst wurden. Das Programm wird in einem 32-bit breiten Speicher abgelegt, welcher zwei 16-bit Instruktionen gleichzeitig pro Takt bereitstellt. Der Speicher ist auf 16-bit

ausgerichtet. Dafür ist ein Programmzähler erforderlich, der nach dem Holen eines Instruktionspaares um 2 erhöht wird.

1.3. Schritt 1: Erstellen der VHDL Quellen für die Zielmodule

Sie werden ein neues Projekt in Vivado mit dem Namen lab8 erstellen. Alle VHDL Quellen aus der Übung 7 werden in das neue Projekt importiert. Sie werden die bestehenden VHDL-Quellen ändern, so dass sie die erforderliche Funktionalität erfüllen. Das Blockdiagramm in Abbildung 1.2. zeigt die superskalare Zielarchitektur.

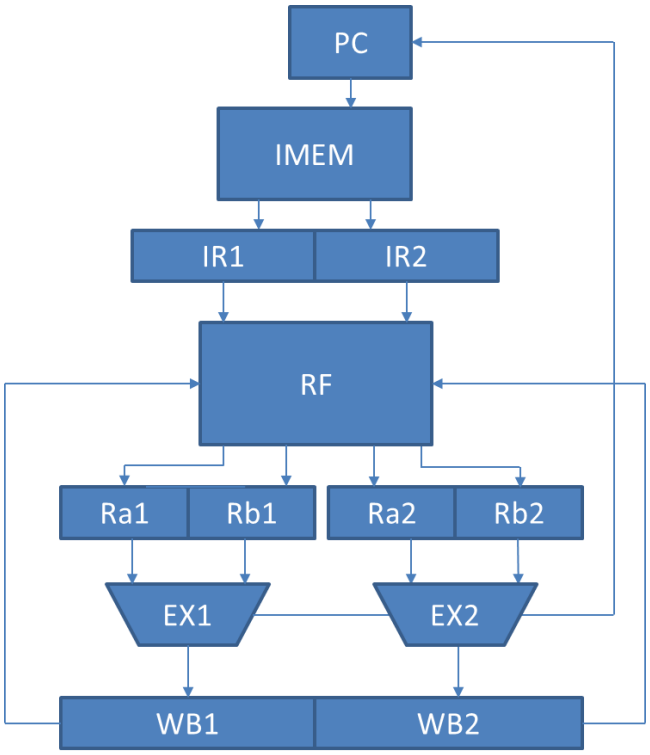

Abbildung 1.2. Superskalare Datenpfadarchitektur.

In der Zielarchitektur hat die Ausführungsstufe zwei funktionale Ausführungseinheiten. Entsprechend liefert der Befehlsspeicher zwei 16-Bit Instruktionen an die Dekodierstufe. Die Instruktionspaare sind an aufeinanderfolgenden Adressen (gerade und ungerade) im Programmspeicher abgelegt. Die Registerbank muss jetzt das gleichzeitige Schreiben zweier Registerwerte und das gleichzeitige Lesen von vier Registerwerten unterstützen. Der Programmzähler wird immer auf eine gerade Adresse aktualisiert. Das Steuerwerk liefert getrennte Steuersignale für jede Funktionseinheit.

1. Modifizieren Sie `rf.vhd` um die Anforderungen der superskalaren Zielarchitektur zu entsprechen. Beachten Sie, dass die Registerbank in einer superskalaren Architektur vier

Lese- und zwei Schreibports anbieten muss. Der erste Schreibport der Registerbank hat Vorrang wenn gleichzeitiges Schreiben in das gleiche Register der Registerbank vorkommt.

2. Passen Sie `pc.vhd` entsprechend der Spezifikation an. Der Programmzähler erhält separate Verzweigungssignale von beiden Ausführungseinheiten. Er wird auf eine gerade Adresse aktualisiert. Setzen die Ausführungseinheiten ein Verzweigungs-Flag gleichzeitig, so wechselt der Programmzähler in einen undefinierten Zustand.
3. Modifizieren Sie `ctrl.vhd` um die notwendigen Steuersignale zu den Datenpfadeinheiten weiter zu geben.
4. Importieren Sie `imem.vhd`**.** Beachten Sie die zwei Ausgabe-Ports zum Befehlsregister.
5. Erstellen Sie zum Schluss das Top-Modul des superskalaren Zielprozessors in VHDL. Bearbeiten Sie dazu `datapath.vhd`.

   Das Top-Modul sollte folgende I/O-Ports haben:

   - `clk` – Takteingang.
   - `rst_n` – asynchrones Reset (low aktiv).
   - `pc_data[7:0]` – Ausgabedaten aus dem Programm Counter.
   - `ir_data1[15:0]` – Ausgabedaten aus dem Instruktionsregister (Instruktion an der geraden Adresse, Instruktionsfolge 1) (IF/ID Pipeline Stufe)
   - `ir_data2[15:0]` – Ausgabedaten aus dem Instruktionsregister (Instruktion an der ungeraden Adresse, Instruktionsfolge 2) (IF/ID Pipeline Stufe)
   - `rf_data1[20:0]` – Ausgabedaten aus der ID/EX Pipeline Stufe (Instruktionsfolge 1). *rf_data(1) = 'Dst Sel(1)' & 'Reg A(1)' & 'Reg B(1)' & 'EX(1)' & 'WB(1)';*
   - `rf_data2[20:0]` – Ausgabedaten aus der ID/EX Pipeline Stufe (Instruktionsfolge 2). *rf_data(2) = 'Dst Sel(2)' & 'Reg A(2)' & 'Reg B(2)' & 'EX(2)' & 'WB(2)';*
   - `alu_data1[11:0]` – Ausgabedaten aus der EX/WB Pipeline Stufe (Instruktionsfolge 1). *alu_data(1) = 'Dst Sel(1)' & 'Alu Reg(1)' & 'WB(1)';*
   - `alu_data2[11:0]` – Ausgabedaten aus der EX/WB Pipeline Stufe (Instruktionsfolge 2). *alu_data(2) = 'Dst Sel(2)' & 'Alu Reg(2)' & 'WB(2)';*
   - `ctrl_data1[9:0]` – Ausgabedaten aus der Steuereinheit (Instruktionsfolge 1). *ctrl_data(1) = pc_wr(1) & ir_wr(1) & rf_ra_sel(1) & alu_reg_wr(1) & alu_b_sel(1) & rf_reg_wr(1) & rf_dst_wr(1) & alu_op(1);*
   - `ctrl_data2[9:0]` – Ausgabedaten aus der Steuereinheit (Instruktionsfolge 2). *ctrl_data(2) = pc_wr(2) & ir_wr(2) & rf_ra_sel(2) & alu_reg_wr(2) & alu_b_sel(2) & rf_reg_wr(2) & rf_dst_wr(2) & alu_op(2);*
   - `br_data1[10:0]` - Ausgabe der Ausführungseinheit in Bezug auf die Verzweigungsausführung (Instruktionsfolge 1). *br_data(1) = branch_condition[1:0](1) & branch_flag_to_pc(1) & branch_address_to_pc[7:0](1)*
   - `br_data2[10:0]` - Ausgabe der Ausführungseinheit in Bezug auf die Verzweigungsausführung (Instruktionsfolge 2). *br_data(2) = branch_condition[1:0](2) & branch_flag_to_pc(2) & branch_address_to_pc[7:0](2)*

1.4. Schritt 2: Simulieren des Top-Designs

Importieren Sie die Testbench-Datei `datapath_tb.vhd` und starten Sie die Simulation. Es wird das gleiche Programm wie in der vorigen Übung ausgeführt. Der Programmablauf ist so organisiert, dass die Ausführung auf dem superskalaren Zielprozessor unterstützt wird. Das Programm ist im IMEM-Block gespeichert. Das vollständige Programm und die Anordnung im Speicher ist in Tabelle 1 beschrieben. Beachten Sie die zwei Instruktionsfolgen, die parallel ausgeführt werden:

Tabelle 1. Testbench-Programm.

address in memory	instruction queue 1	action	address in memory	instruction queue 2	action
00	addi 0, r0	r0 = 0	01	addi 1, r1	r1 = 1
02	addi 2, r2	r2 = 2	03	addi 3, r3	r3 = 3
04	addi 4, r4	r4 = 4	05	addi 5, r5	r5 = 5
06	addi 6, r6	r6 = 6	07	addi 7, r7	r7 = 7
08	add r0, r1, r7	r7 = 1	09	nop	stall
0A	sub r5, r4, r6	r6 = 1	0B	inc r0	r0 = 1
0C	dec r2	r2 = 1	0D	and r1, r5, r5	r5 = 1
0E	xori 5, r4	r4 = 1	0F	andi 1, r3	r3 = 1
10	or r1, r1, r1	r1 = 1	11	addi x”FE”, r0	r0 = x”FF”
12	nop	stall	13	nop	stall
14	nop	stall	15	nop	stall
16	subi 1, r0	r0 = x”FE”	17	nop	stall
18	nop	stall	19	nop	stall
1A	nop	stall	1B	nop	stall
1C	not r0, r0	r0 = 1	1D	nop	stall
1E	jmp x”30”	PC = x”30”	1F	nop	stall
20	nop	stall	21	nop	stall
22	nop	stall	23	nop	stall
24	dec r7	r7 = 1	25	nop	stall
…	…	…	…	…	…
30	sub r2, r3, r4	r4 = 0	31	nop	stall
32	nop	stall	33	nop	stall
34	nop	stall	35	nop	stall
36	beqz x”80”, r4	if r4 = 0, PC=x”80”, else PC+2	37	nop	stall
38	nop	stall	39	nop	stall
3A	nop	stall	3B	nop	stall
3C	add r5, r6, r7	r7 = 2	3D	nop	stall
3E	nop	stall	3F	nop	stall
40	nop	stall	41	nop	stall
42	bneqz x”24”, r7	if r7 /= 0, PC=x”24”, else PC+2	43	nop	stall

3E	nop	stall	3F	nop	stall
40	nop	stall	41	nop	stall
…	…	…	…	…	…
80	inc r4	r4 = 1	81	nop	stall
82	jmp x”3C”	PC = x”3C”	83	nop	stall
84	nop	stall	85	nop	stall
86	nop	stall	87	nop	stall

1.5. Schritt 3: Synthetisieren des Top-Designs

- Führen Sie die Synthese des Top-Moduls in Vivado durch.
- Generieren Sie den ‘Timing Summary Report‘ und überprüfen Sie die Ergebnisse für ‘Setup Delay‘ unter ‘Unconstrained Paths‘. Was ist das ‘Worst Setup Delay‘? Was ist der Unterschied zur skalaren Befehlspipeline?
- Wie ist die Programmausführungszeit und wie verhält sie sich zur Ausführungszeit der skalaren Befehlspipeline?