Praktikum zur Vorlesung

Prozessorarchitektur

SS 2018

Übungsblatt 9

Implementierung eines Bypasses in einer superskalaren Prozessorarchitektur

Abgabefrist: Mittwoch 27.06.2018, 10:00 Uhr

1.1. Einführung

Pipeline-Architekturen leiden unter Datenabhängigkeiten. Um diese Datenkonflikte zu umgehen muss die Pipeline oft angehalten werden bis das benötigte Resultat einer in Abarbeitung befindlichen Instruktion im Register gespeichert und somit verfügbar ist. Das führt zur Verschlechterung der Prozessorleistung. Die Datenkonflikte können aber zum Teil durch Datenweiterleitung (bypassing, forwarding) aufgelöst werden. Die Bypass-Techniken nutzen Bypass-Pfade in der Pipeline, durch die berechnete Daten zwischen den Pipelinestufen weitergeleitet werden, bevor diese in das Ergebnisregister geschrieben sind.

1.2. Projektbeschreibung

In dieser Übung werden Sie einen Bypass für eine superskalare Architektur implementieren. Ihre Aufgabe ist es, eine Forward-Control-Unit zu entwerfen und diese in die superskalare Architektur der vergangenen Übung zu integrieren. Für jeden parallelen Pfad (Slot) der superskalaren Architektur muss eine Forward-Control-Unit (Fwd-Unit) in der ID-Phase implementiert werden (siehe Abbildung 1.). Beachten Sie, dass in der Abbildung die Verbindung der Kontrollsignale nicht dargestellt ist. Jede Fwd-Unit erhält als Eingabe die in der EX-Phase beider Slots berechneten Ergebnisse, die Werte, die in der WB-Phase gerade in die Registerbank zurückgeschrieben werden und die Operanden aus der Registerbank, die für die Operation in der DE-Phase gerade gelesen werden. Die Fwd-Unit muss nun aus diesen Werten die korrekten Operanden für die Operation in der DE-Phase auswählen. Dazu muss für die Werte, die sich in der EX- und WB-Phase befinden die Zielregister für diese Werte bekannt sein. Abhängig von den Quellregistern aus denen die Quelloperandenwerte aus der Registerbank gelesen werden und den Zielregistern aus der EX- und WB-Phase kann diese Entscheidung in der Fwd-Unit getroffen werden. Erweitern Sie die vorhandenen Pipelinestufen, so dass die Zielregister in der EX- und WB-Phase gespeichert werden können und implementieren Sie dann Fwd-Unit.

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

Sie werden ein neues Projekt in Vivado mit dem Namen lab9 erstellen. Alle VHDL Quellen aus der Übung 8 werden in das neue Projekt importiert. Sie werden neue VHDL-Quellen erstellen und die bestehenden ändern, so dass sie die erforderliche Funktionalität erfüllen.

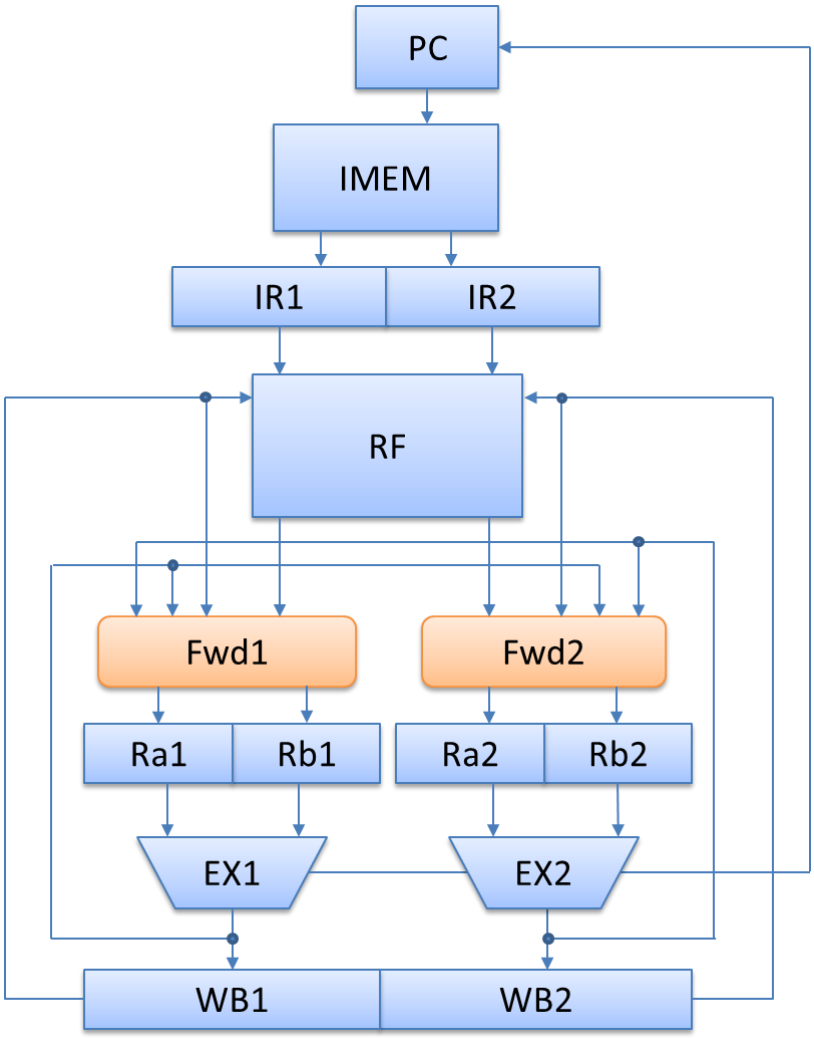

Abbildung 1. Superskalare Datenpfadarchitektur mit Forwarding.

1. Erstellen Sie die VHDL-Beschreibung der Forwarding-Einheit `fwd_unit.vhd`. Dieses Modul organisiert das kontrollierte Weitergeben der Daten aus den EX und WB Stufen der Pipeline zur ID-Stufe. Achten Sie darauf, nur die gültigen Daten weitergegeben werden.
2. Modifizieren Sie bestehende oder erstellen Sie neue VHDL-Beschreibungen um den Zugang zu den Daten zu erhalten, die von der Forwarding-Einheit benötigt werden.
3. Importieren Sie `imem.vhd`.

4. Erstellen Sie das Top-Modul in VHDL. Bearbeiten Sie dazu `datapath.vhd`. Behalten Sie die Konfiguration der externen Ports für das Top-Modul aus der vorangegangenen Übung bei.

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. Die Nop-Instruktionen, die durch Forwarding aufgelöst werden können, wurden aus dem Programm entfernt (Tabelle 1).

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**	sub r5, r4, r6	r6 = 1
0A	inc r0	r0 = 1	**0B**	dec r2	r2 = 1
0C	and r1, r5, r5	r5 = 1	**0D**	xori 5, r4	r4 = 1
0E	andi 1, r3	r3 = 1	**0F**	or r1, r1, r1	r1 = 1
10	addi x”FE”, r0	r0 = x”FF”	**11**	nop	stall
12	subi 1, r0	r0 = x”FE”	**13**	nop	stall
14	not r0, r0	r0 = 1	**15**	jmp x”30”	PC = x”30”
16	nop	stall	**17**	nop	stall
18	…	…	**19**	…	…
24	dec r7	r7 = 1	**25**	nop	stall
…	…	…	**…**	…	…
30	sub r2, r3, r4	r4 = 0	**31**	nop	stall
32	beqz x”80”, r4	if r4 = 0, PC=x”80”, else PC+2	**33**	nop	stall
34	nop	stall	**35**	nop	stall
36	…	…	**37**	…	…
3A	nop	stall	**3B**	nop	stall
3C	add r5, r6, r7	r7 = 2	**3D**	nop	stall
3E	bneqz x”24”, r7	if r7 /= 0, PC=x”24”, else PC+2	**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	…	…	**87**	…	…

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‘? Wie ist der Unterschied zur Architektur mit skalarer Befehlspipeline?
- Wie ist die Programmausführungszeit und wie verhält sie sich zur Ausführungszeit der superskalaren Architektur ohne Bypass?