Praktikum zur Vorlesung

Prozessorarchitektur

SS 2018

Übungsblatt 6

Implementierung einer Befehlspipeline

Abgabefrist: Mittwoch 23.05.2018, 10:00 Uhr

1.1. Einführung

Durch die Einteilung der Befehlsverarbeitung in mehrere Zyklen im vorangegangenen Praktikum, können diese Befehle nun überlappend in einer Befehlspipeline ausgeführt werden. Damit wird die Prozessorleistung verbessert. Zu beachten sind jedoch Datenabhängigkeiten zwischen den parallel abgearbeiteten Befehlen. Außerdem müssen relevante Steuerinformationen durch die Pipeline weitergegeben werden, um die korrekte Ausführung der Befehle zu ermöglichen.

1.2. Projektbeschreibung

In dieser Übung werden Sie eine Befehlspipeline implementieren, die die überlappende Ausführung mehrerer Befehle unterstützt. Ihre Aufgabe ist es, die Mehrzyklen-Architektur aus der Übung 5 in eine Pipeline-Ausführung zu ändern. Die zu implementierende Pipeline soll die aus der Vorlesung bekannten vier Phasen unterstützen und diese in der in Abbildung 1.1 dargestellten Weise überlappen.

clk 1	*clk 2*	*clk 3*	*clk 4*	*clk 5*	*clk 6*
IF	ID	EX	WB		
	IF	ID	EX	WB	
		IF	ID	EX	WB

Abbildung 1.1. Pipeline-Ausführung.

In einer Pipeline-Architektur können Hazards auftreten. Werden zum Beispiel zwei Befehle nacheinander ausgeführt und das Ergebnis des ersten Befehls wird in der Pipeline vom nachfolgenden Befehl verwendet kann es zum Datenhazard kommen. Im Beispiel aus Abbildung 1.2. schreibt der Additionsbefehl das Ergebnis in das Register R2 welches vom folgenden

Subtraktionsbefehl verwendet wird. Die Dekodierung des Subtraktionsbefehls wird aber ausgeführt, bevor das Ergebnis der Addition in das Register geschrieben wurde.

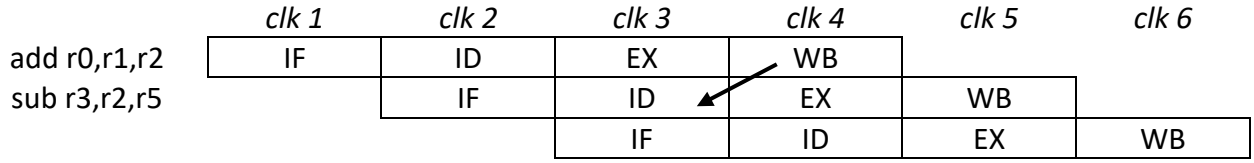

Abbildung 1.2. Hazard in der Pipeline.

Eine Möglichkeit Hazards aufzulösen besteht darin, einige Stufen der Pipeline anzuhalten, bis der Konflikt aufgelöst wurde. In Software kann dies durch das Einfügen von Nop-Instruktionen (*nop*) behoben werden (Abbildung 1.3). In dieser Übung werden wir annehmen, dass die Nop-Instruktionen in das Programm bereits so eingefügt wurden, und somit keine Datenhazards auftreten können.

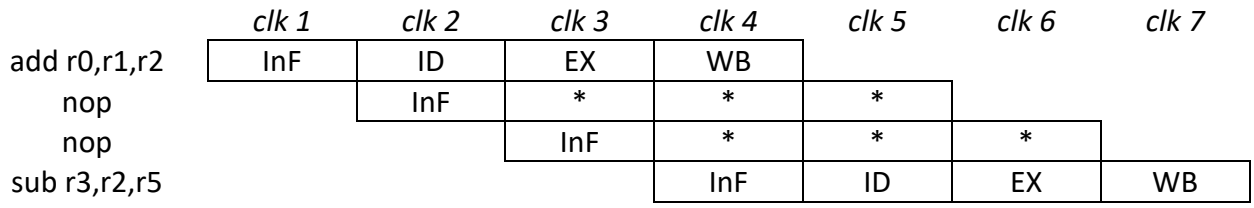

Abbildung 1.3. Beheben von Hazards durch Einfügen von Nop-Instruktionen.

.

Die Nop-Instruktion wird in unserem Prozessor kodiert als:

type = ‘00’	opcode = ‘00000’	sr1 = ‘xxx’	sr2 = ‘xxx’	dst = ‘xxx’

Ein weiteres Problem in einer Pipeline-Architektur ist es, die Konsistenz der Steuerdaten zu gewährleisten. Da das Instruktionsregister mit jedem neuen Taktzyklus aktualisiert wird, müssen alle Steuerdaten für eine Instruktion, die in der Execution- (EX) und Writeback- (WB) Phase verwendet werden, bereits in der Decode-Phase (DE) erzeugt und durch die Pipeline weitergeben werden. Die Zielarchitektur des Pipeline-Datenpfads ist in Abbildung 1.4 dargestellt.

1.3. Schritt 1: Erstellen der VHDL Quellen der Datenpfadblöcke

Erstellen Sie ein neues Projekt in Vivado mit dem Namen *lab6*. Importieren Sie alle VHDL Quellen aus der Übung 5 in das neue Projekt. Ändern Sie die bestehenden VHDL-Quellen, sodass diese die erforderliche Funktionalität erfüllen. Der umzusetzende Pipeline-Datenpfad ist in Abbildung 1.4 dargestellt.

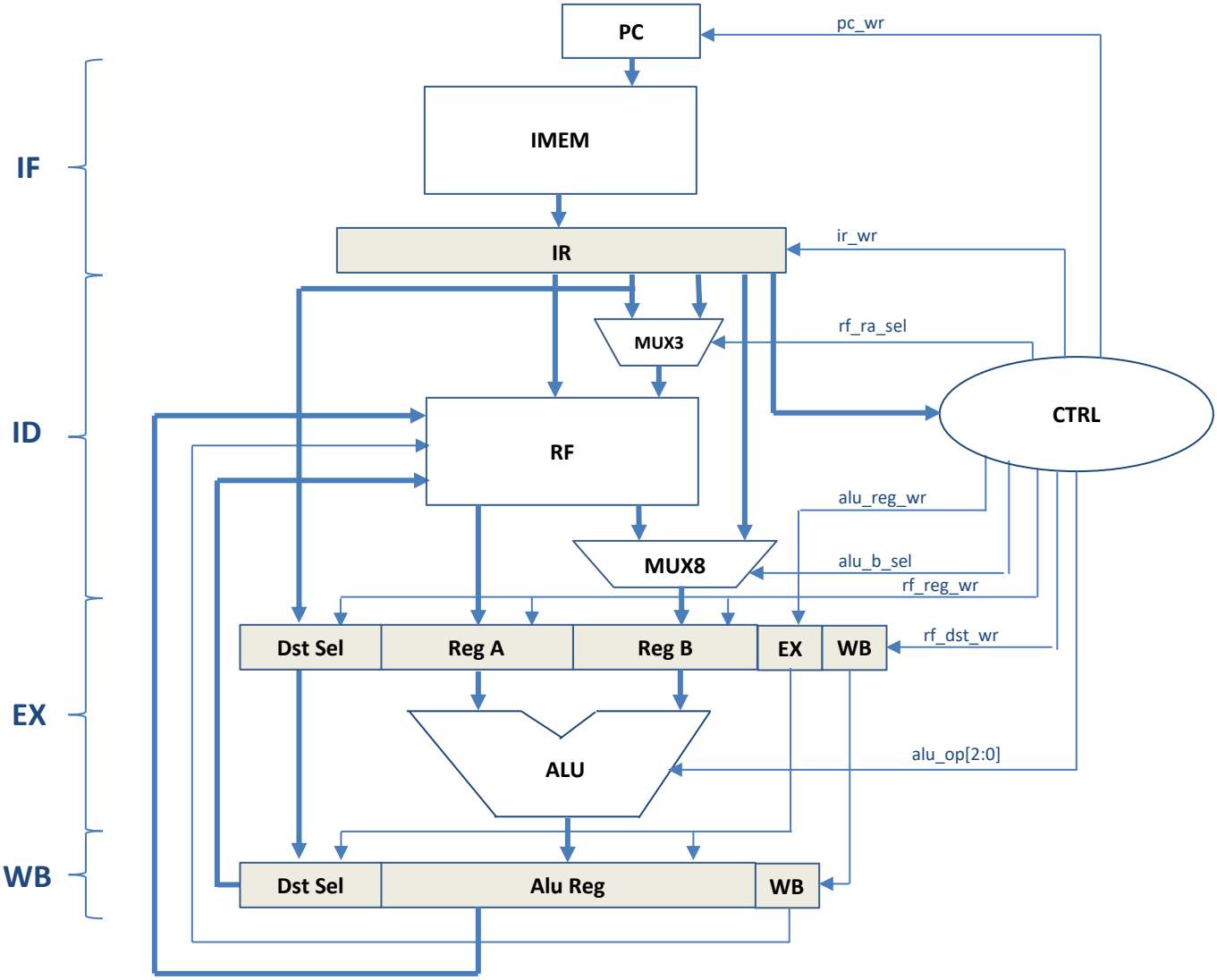

Abbildung 1.4. Befehlspipeline mit Datenpfad.

1. Erstellen Sie die Steuereinheit (***ctrl.vhd***), um die Pipeline-Ausführung zu ermöglichen.
2. Erweitern Sie die Pipelinestufen, um die Propagierung des Steuersignals zu ermöglichen.
3. Achten Sie auf die Propagierung von Konstanten für I-Typ-Anweisungen, indem Sie den Multiplexer (MUX8) vor der ‘Reg B‘ platzieren, wie in der Abbildung 1.4 gezeigt.
4. Importieren Sie ***imem.vhd***
5. Erstellen Sie das VHDL-Top-Modul für die Pipeline-Datenpfad-Architektur.

Das Top-Modul sollte folgende Ports haben:

- ***clk*** – Takteingang.
- ***rst_n*** – asynchrones Reset (low active).
- **pc_data[7:0]** – Ausgabedaten aus dem Programm Counter.
- ***ir_data[15:0]*** – Ausgabedaten aus dem Instruktionsregister (IF/ID Pipeline Stufe)
- ***rf_data[20:0]*** – Ausgabedaten aus der ID/EX Pipeline Stufe.
  rf_data = 'Dst Sel' & 'Reg A' & 'Reg B' & 'EX' & 'WB';
- ***alu_data[11:0]*** – Ausgabedaten aus der EX/WB Pipeline Stufe.
  alu_data = 'Dst Sel' & 'Alu Reg' & 'WB';
- ***ctrl_data[9:0]*** – Ausgabedaten aus der Steuereinheit.
  ctrl_data = pc_wr & ir_wr & rf_ra_sel & alu_reg_wr & alu_b_sel & rf_reg_wr & rf_dst_wr & alu_op;

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 den früheren Übungen ausgeführt. Das Programm ist in IMEM abgelegt. Es wurde um einige Nop-Instruktionen erweitert, um Datenhazards zu vermeiden:

```
addi 0, r0          -- r0 = 0
addi 1, r1          -- r1 = 1
addi 2, r2          -- r2 = 2
addi 3, r3          -- r3 = 3
addi 4, r4          -- r4 = 4
addi 5, r5          -- r5 = 5
addi 6, r6          -- r6 = 6
addi 7, r7          -- r7 = 7
add r0, r1, r7      -- r7 = r0 + r1 = 0 + 1 = 1
sub r5, r4, r6      -- r6 = r5 – r4 = 5 – 4 = 1
inc r0              -- r0 = r0 + 1 = 0 + 1 = 1
dec r2              -- r2 = r2 – r1 = 2 – 1 = 1
and r1, r5, r5      -- r5 = r1 and r5 = x"01" and x"03" = x"01"
xori 5, r4          -- r4 = r4 xor x"05" = x"04" xor x"05" = x"01"
andi 1, r3          -- r3 = r3 and x"01" = x"03" xor x"01" = x"01"
or r1, r1, r1       -- r1 = r1 or r1 = x"01" or x"01" = x"01"
addi x"FE", r0      -- r0 = r0 + x"FE" = x"01" + x"FE" = X"FF"
nop
nop
subi 1, r0          -- r0 = r0 – x"01" = x"FF" – x"01 = x"FE"
nop
nop
not r0, r0          -- r0 = not r0 = not x"FE" = x"01"
```

Überprüfen Sie die Simulation in der ‘Tcl-Console‘ auf Fehler. Analysieren Sie die Waveform, indem Sie die unterschiedlichen Signale im Waveform-Fenster beobachten. Wie ist die Programmlaufzeit?

1.5. Schritt 3: Synthetisieren des Top-Designs

- Führen Sie die Synthese des Top-Moduls 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 verhält sich diese Lösung im Vergleich zum Single-Cycle- und Multi-Cycle-Datenpfad?

Anhang (Port Deklarationen)

```
entity alu is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          alu_op : in STD_LOGIC_VECTOR (2 downto 0);
          alu_out : out STD_LOGIC_VECTOR (7 downto 0);
          clk : in STD_LOGIC;
          rst_n : in STD_LOGIC;
          alu_wr : in STD_LOGIC;
          rf_dst_wr_in : in STD_LOGIC;
          rf_dst_wr_out : out STD_LOGIC;
          rf_dst_sel_in : in STD_LOGIC_VECTOR (2 downto 0);
          rf_dst_sel_out : out STD_LOGIC_VECTOR (2 downto 0)
          );
end alu;
```

```
entity rf is
   Port ( clk : in STD_LOGIC;
          rst_n : in STD_LOGIC;
          wrdata : in STD_LOGIC_VECTOR (7 downto 0);
          sa : in STD_LOGIC_VECTOR (2 downto 0);
          sb : in STD_LOGIC_VECTOR (2 downto 0);
          dr : in STD_LOGIC_VECTOR (2 downto 0);
          rf_dstreg_wr : in STD_LOGIC;
          rega : out STD_LOGIC_VECTOR (7 downto 0);
          regb : out STD_LOGIC_VECTOR (7 downto 0);
          rf_outreg_wr : in STD_LOGIC;
          reg_b_sel : in STD_LOGIC;
          const : in STD_LOGIC_VECTOR (7 downto 0);
          alu_reg_wr_in : in STD_LOGIC;
          alu_reg_wr_out : out STD_LOGIC;
          rf_dst_wr_in : in STD_LOGIC;
          rf_dst_wr_out : out STD_LOGIC;
          rf_dst_sel_in : in STD_LOGIC_VECTOR (2 downto 0);
          rf_dst_sel_out : out STD_LOGIC_VECTOR (2 downto 0)
          );
end rf;
```

```
entity pc is
    Port ( clk : in STD_LOGIC;
           rst_n : in STD_LOGIC;
           pc_out : out STD_LOGIC_VECTOR (7 downto 0);
           pc_wr : in STD_LOGIC
           );
end pc;

entity mux3 is
    Port ( mux_a : in STD_LOGIC_VECTOR (2 downto 0);
           mux_b : in STD_LOGIC_VECTOR (2 downto 0);
           mux_sel : in STD_LOGIC;
           mux_out : out STD_LOGIC_VECTOR (2 downto 0));
end mux3;



entity ir is
    Port ( clk : in STD_LOGIC;
           rst_n : in STD_LOGIC;
           ir_in : in STD_LOGIC_VECTOR (15 downto 0);
           ir_out : out STD_LOGIC_VECTOR (15 downto 0);
           ir_wr : in STD_LOGIC);
end ir;

entity imem is
    Port ( addr : in STD_LOGIC_VECTOR (7 downto 0);
           dout : out STD_LOGIC_VECTOR (15 downto 0));
end imem;

entity datapath is
    Port ( clk : in STD_LOGIC;
           rst_n : in STD_LOGIC;
           pc_data : out STD_LOGIC_VECTOR (7 downto 0);
           imem_data : out STD_LOGIC_VECTOR (15 downto 0);
           ir_data : out STD_LOGIC_VECTOR (15 downto 0);
           rf_data : out STD_LOGIC_VECTOR (20 downto 0);
           alu_data : out STD_LOGIC_VECTOR (11 downto 0);
           ctrl_data : out STD_LOGIC_VECTOR (9 downto 0));
end datapath;

entity ctrl is
    Port ( instr : in STD_LOGIC_VECTOR (6 downto 0);
           alu_op : out STD_LOGIC_VECTOR (2 downto 0);
           alu_b_sel : out STD_LOGIC;
           alu_reg_wr : out STD_LOGIC;
           rf_ra_sel : out STD_LOGIC;
           rf_reg_wr : out STD_LOGIC;
           rf_dst_wr : out STD_LOGIC;
           pc_wr : out STD_LOGIC;
           ir_wr : out STD_LOGIC;
           clk : in STD_LOGIC;
           rst_n : in STD_LOGIC);
end ctrl;
```

```
entity rf is
    Port ( clk : in STD_LOGIC;
           rst_n : in STD_LOGIC;
           wrdata : in STD_LOGIC_VECTOR (7 downto 0);
           sa : in STD_LOGIC_VECTOR (2 downto 0);
           sb : in STD_LOGIC_VECTOR (2 downto 0);
           dr : in STD_LOGIC_VECTOR (2 downto 0);
           rf_dstreg_wr : in STD_LOGIC;
           rega : out STD_LOGIC_VECTOR (7 downto 0);
           regb : out STD_LOGIC_VECTOR (7 downto 0);
           rf_outreg_wr : in STD_LOGIC;
           reg_b_sel : in STD_LOGIC;
           const : in STD_LOGIC_VECTOR (7 downto 0);
           alu_reg_wr_in : in STD_LOGIC;
           alu_reg_wr_out : out STD_LOGIC;
           rf_dst_wr_in : in STD_LOGIC;
           rf_dst_wr_out : out STD_LOGIC;
           rf_dst_sel_in : in STD_LOGIC_VECTOR (2 downto 0);
           rf_dst_sel_out : out STD_LOGIC_VECTOR (2 downto 0)
           );
end rf;
```