Praktikum zur Vorlesung

Prozessorarchitektur

SS 2018

Übungsblatt 5

Entwurf eines Mehrzyklen-Datenpfads

Abgabefrist: Mittwoch 16.05.2018, 10:00 Uhr

1.1. Einführung

In Übung 4 haben Sie einen Einzyklen-Datenpfad entworfen, der einen Befehl pro Taktzyklus ausführt. Als nächstes soll ein Mehrzyklen-Datenpfad entworfen werden. Dafür müssen Registerstufen zwischen den Funktionseinheiten des Datenpfades vorbereitet werden. Das macht die Ausführung von Instruktionen in mehreren Taktzyklen möglich. In einem einfachen Mehrzyklen-Datenpfad erfordert jeder Schritt der Befehlsverarbeitung (Instruction Fetch, Instruction Decode, Execution und Write Back) einen Taktzyklus zur Ausführung.

In dieser Übung sollen Sie einen Mehrzyklen-Datenpfad entwickeln. Als Vorlage werden Sie den Einzyklen-Datenpfad aus der Übung 4 verwenden. Sie werden eine Steuereinheit erstellen, die eine Mehrzyklenausführung ermöglicht.

1.2. Erstellen der Datenpfadblöcke in VHDL

1. In Vivado werden Sie ein neues Projekt mit den Namen ***lab5*** erstellen und alle VHDL-Quelldateien aus Lab 4 in das neue Projekt importieren. Stellen Sie sicher, dass "Copy sources into projects" beim Import ausgewählt ist. Sie werden vorhandene VHDL-Dateien ändern und neu benötigte VHDL-Dateien erstellen. Sie müssen auch die Funktionalität der Steuerlogik aktualisieren, um eine Mehrzyklenausführung zu unterstützen. Das Blockschaltbild des Zielmoduls ist in Abbildung 1.1 dargestellt. Sie werden das Zielmodul simulieren und synthetisieren.

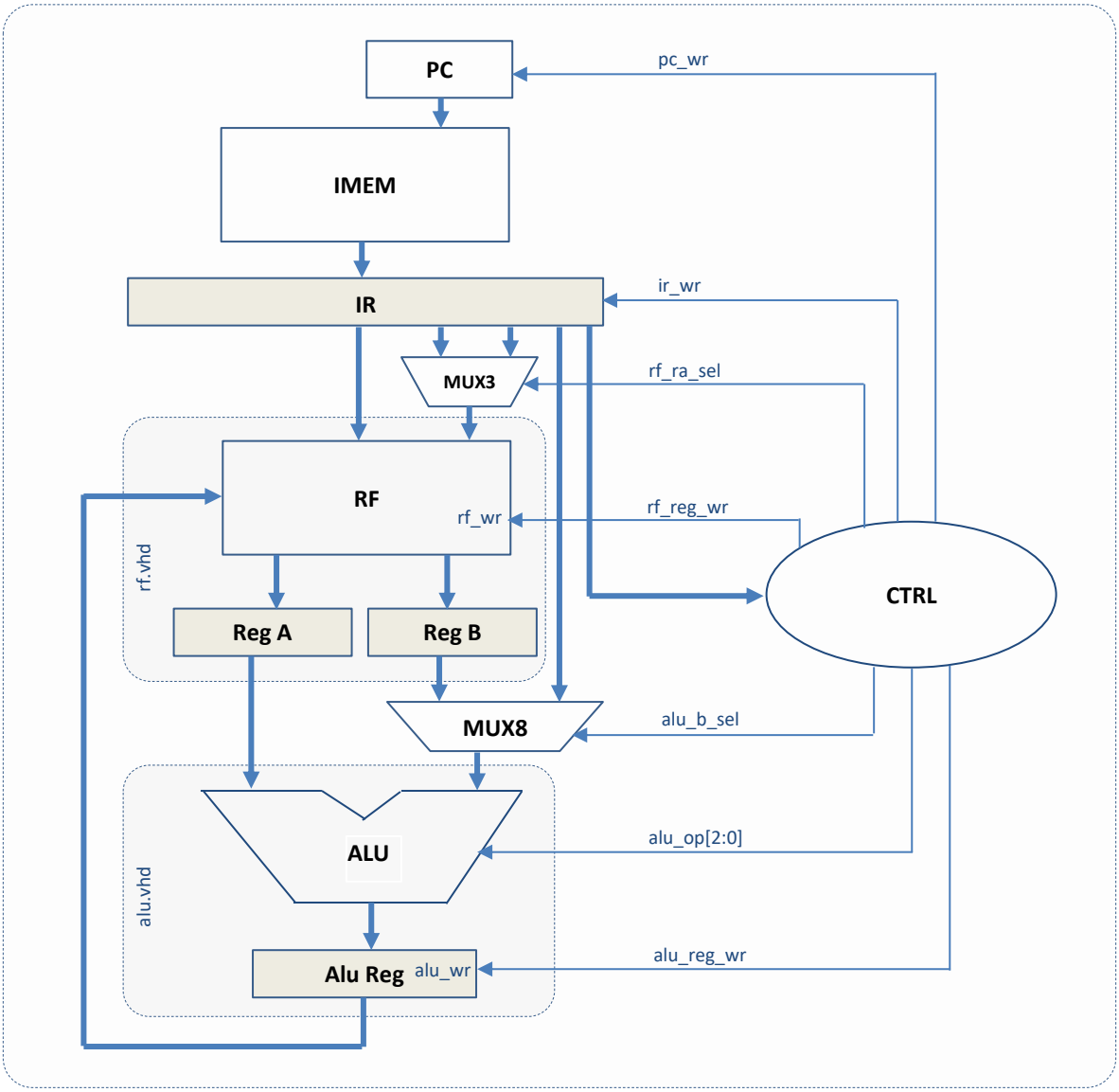

Abbildung 1.1. Mehrzyklen-Datenpfad, CLK und Reset-Signal sind nicht eingezeichnet. Port für Datapath und Testbench beachten!

Beachten Sie folgende Hinweise:

- Erstellen Sie die VHDL-Quelle (***ir.vhd***) des Instruktionsregisters (IR) als 16-Bit-Register mit asynchronem Reset und Load-Enable-Signal (***ir_wr***).
- Fügen Sie ein Load-Enable-Signal (***pc_wr***) in den Programmzähler (PC) ein. Ist pc_wr = "1", so wird pc_out bei steigender Flanke von clk inkrementiert, ansonsten hält pc_out seinen Wert. Verändern Sie ***pc.vhd*** entsprechend.
- Fügen Sie die Ausgaberegister 'Reg A' und 'Reg B' und ein Load-Enable-Signal (***rf_wr***) in die Registerbank (RF) ein, welches das Schreiben der neuen Ausgaberegister steuert. Reg A und Reg B übernehmen die Werte von ehemals ra und rb bei rf_wr = "1". Verändern Sie ***rf.vhd*** entsprechend.

- Fügen Sie ein Ausgaberegister in die ALU durch editieren der Datei ***alu.vhd*** ein. Fügen Sie ebenfalls ein Steuersignal (***alu_wr***) ins Modul ein, welches das Schreiben des ALU-Ausgangsregisters steuert.
- **Alle hinzugefügten Register sollen beim Reset mit "0" initialisiert werden. Das Reset-Signal ist asynchron und low aktiv.**

1.3. Erstellen der Steuerlogik in VHDL

Sie werden in die Steuereinheit (***ctrl.vhd***), die in Übung 4 entwickelt wurde, einen Zustandsautomat (FSM) einfügen. Die FSM ist in der Abbildung 1.2. beschrieben.

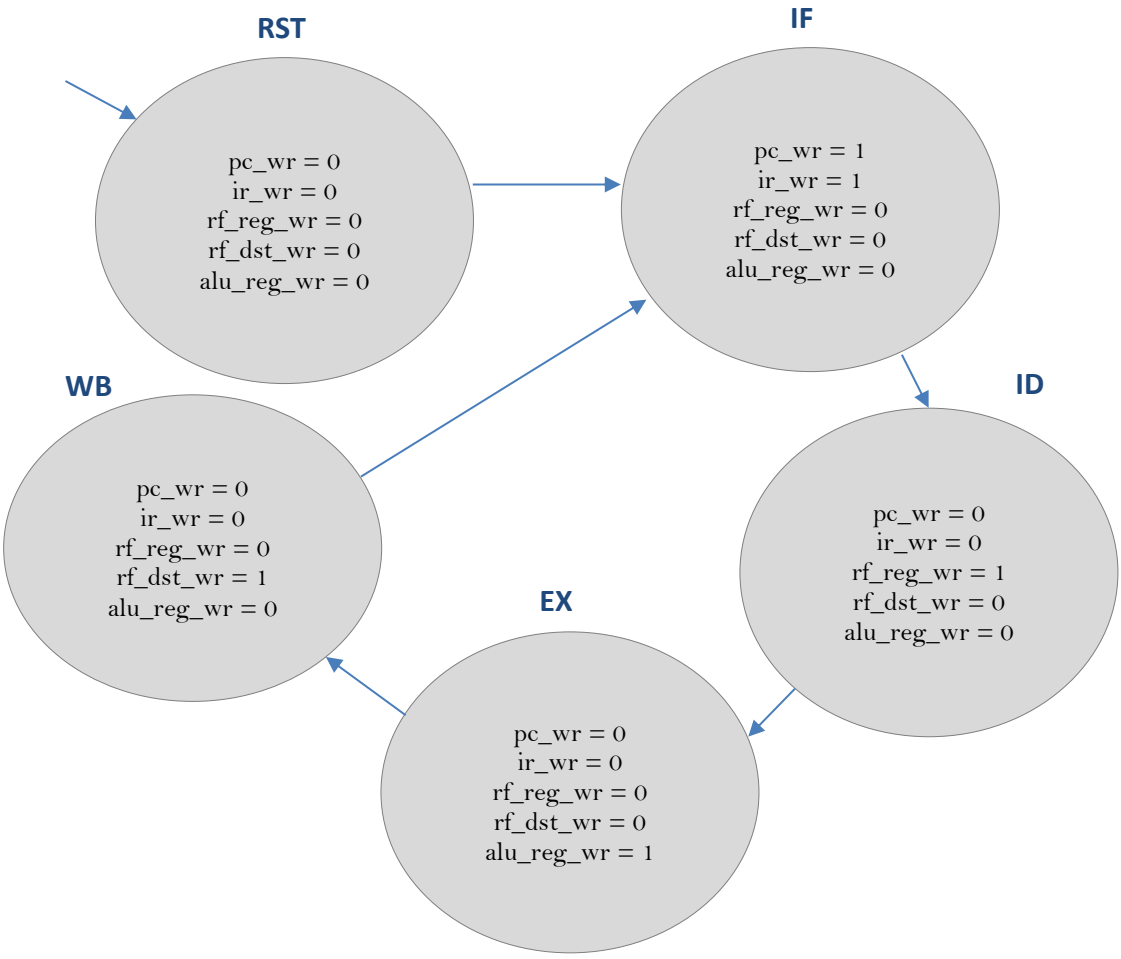

Abbildung 1.2. Zustandsautomat des Steuerlogik.

Die FSM liefert die Signale, die das Schreiben von Registerstufen nach jedem Takt der Befehlsverarbeitung (Abbildung 1.3.) steuert. Die Signale für ALU- (***alu_op***) und die Multiplexer-Steuerung (***alu_b_sel***, ***rf_ra_sel***) sollen in einem separaten VHDL-Prozess beschrieben werden.

Splitten Sie hier die Instruktion in ***op_type, r_opcode und i_opcode*** um diese besser behandeln zu können.

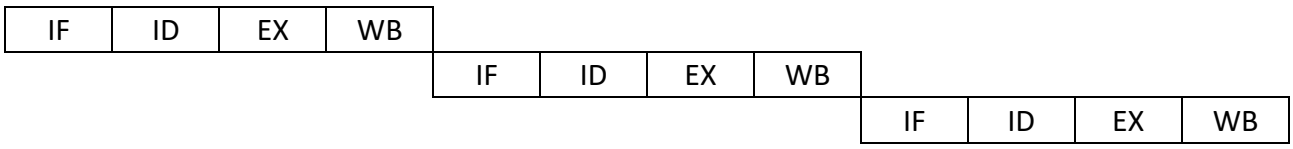

Abbildung 1.3. Ausführungssequenzen des Mehrzyklendatenpfads.

1.4. Erstellen des Datenpfad Top Moduls in VHDL

- Erstellen Sie das Top Modul (***datapath.vhd***) gemäß Abbildung 1.1.
  Das Top-Design sollte die folgenden Ein- und Ausgangsports enthalten (Reihenfolge in der VHDL-Beschreibung einhalten!):

  clk – Takt Eingang.
  rst_n – asynchron Reset (low active).
  pc_data[7:0] – Ausgabe aus PC.
  imem_data[15:0] – Ausgabe aus IMEM.
  ir_data[15:0] – Ausgabe aus IR.
  alu_data[7:0] – Ausgabe aus ALU-Register.
  rf_we – RF write-back Steuersignal (***rf_dst_wr***); Ausgabe aus CTRL.
  pc_incr – PC load Steuersignal (***pc_wr***); Ausgabe aus CTRL.

- Führen Sie die RTL-Analyse durch, um festzustellen, ob das Top-Modul synthetisierbar ist. Prüfen Sie das generierte Blockdiagramm.

1.5. Simulieren des Top Moduls

- Importieren Sie die Testbench-Datei ***datapath_tb*** und führen Sie die Simulation durch. Die Simulation führt das gleiche Programm wie in Übung 4 aus. Überprüfen Sie die Meldungen und korrigieren Sie eventuelle Fehler. Analysieren Sie die Waveform, indem die unterschiedlichen Signale in das Waveform-Fenster eingefügt werden. Wie lange ist die Programmlaufzeit?

1.6. Synthese des Entwurfs

- Führen Sie die Synthese durch.
- Generieren Sie die Timing-Summary und überprüfen Sie die Ergebnisse des Setup-Delays unter ‘Unconstrained Paths‘. Wie groß ist das Total-Delay? Welchen Unterschied können Sie zum Total-Delay zum Einzyklen-Datenpfad feststellen?

Anhang

```
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);
        we : in STD_LOGIC;
        rega : out STD_LOGIC_VECTOR (7 downto 0);
        regb : out STD_LOGIC_VECTOR (7 downto 0);
        rf_wr : in STD_LOGIC
      );
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 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 alu is Port (
          clk : in STD_LOGIC;
          rst_n : in STD_LOGIC;
          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);
          alu_wr : in STD_LOGIC
         );
end alu;

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;
```