Entwurf digitaler Systeme
Übungsblatt 4

27. Juni 2017

In dieser Übung soll ein kleiner Prozessor implementiert werden. Der Prozessor soll mit 8 Registern und einer 3-stufigen Pipeline ausgestattet werden. Alle Berechnungen werden über einen Akkumulator ausgeführt. Der Controller soll über einen Adressraum von 16-Bit verfügen. Näheres dazu in der folgenden Beschreibung der Instruktionen:

Syntax	Kodierung	Beschreibung
`nop`	`0000 0000`	Keine Operation.
`rac rd`	`0100 0 rd=0bxxx`	Verschiebt den Inhalt des Akkumulators in ein Register.
`wac rd`	`0100 1 rd=0bxxx`	Verschiebt den Inhalt eines Registers in den Akkumulators.
`mal/mah`	`0000 00 {01/10}`	Schreibt den Inhalt des 8-Bit Akkumulators in die untere/obere Hälfte des 16-Bit Memory-Address-Registers (MAR).
`jmp`	`0000 0011`	Sprung zu der Programmadresse, die im MAR gespeichert ist.
`lds/ldu` *imm4*	`00{11/10}` *imm4*`=0bxxxx`	Lädt die Vorzeichen-behaftete/-lose 4-Bit Konstante in den Akkumulator.
`add/sub rd`	`0101 {0/1} rd=0bxxx`	Addiert/Subtrahiert den Inhalt des Registers `rd`zum/vom Akkumulator.
`lui` *imm4*	`0001` *imm4*`=0bxxxx`	Lädt die 4-Bit Konstante in die obere Hälfte der Bits des Akkumulators.
`ori` *imm4*	`0110` *imm4*`=0bxxxx`	Bitweise Disjunktion der vorzeichenlosen Konstante mit dem Akkumulator.
`addi` *imm4*	`0111` *imm4*`=0bxxxx`	Addition des Akkumulators mit der Vorzeichen-behafteten 4-Bit Konstante.

Die Pipeline des Prozessors soll die folgenden Schritte umfassen:

- Fetch - Lädt die im Programmspeicher and der durch den Programmzähler definierten Adresse gespeicherte Instruktion in das Instruktionsregister.
- Decode - dekodiert die Instruktion und setzt Steuersignale.
- Execute - führt die Operation anhand der Steuersignale aus.

1. Implementieren Sie eine Komponente für einen Speicher. Definieren Sie dazu zunächst einen Arraytyp. Die Schnittstelle umfasst einen Dateneingang, einen Datenausgang, einen Adresseingang, und Kontrollsignale zum Ansteuern als Schnittstelle.

2. Implementieren Sie analog eine Komponente für das Registerfile. Das Registerfile besteht dabei aus den 8-Registern, die über einen Multiplexer mit dem Ausgang verbunden sind. Definieren Sie hier selbst alle weiteren Eingangs und Ausgangssignale.

3. Es soll nun die ALU mit dem Akkumulator realisiert werden. Verwenden Sie dazu das Grundgerüst `alu.vhd`. Achten Sie darauf, dass sich die Additionsoperationen mittels eines Addierers realisieren lassen.

4. Integrieren Sie nun die ALU zusammen mit dem Registerfile in eine Komponente Execute. Verwenden Sie dazu das Grundgerüst `execute.vhd`.

5. Realisieren Sie nun die Komponente für die Dekodierung. Verwenden Sie dazu das Grundgerüst `decode.vhd`. Diese liefert im Wesentlichen die Steuersignale für die Execute, aber auch für die später angelegte Fetch-Stufe (bspw. für den Sprungbefehl). Diese Steuersignale müssen anhand der Instruktionskodierung gewählt werden.

6. Implementieren Sie nun die Fetch-Stufe. Verwenden Sie dazu das Grundgerüst `fetch.vhd`. Instanziieren Sie hier den Programmspeicher. Legen Sie entsprechende Ressourcen für Instruktions- und das Speicheradressregister sowie den Programmzähler an.

7. Integrieren Sie alle Komponenten im Grundgerüst `proc.vhd`.