

Entwurf digitaler Systeme

Übungsblatt 5

13. Dezember 2019

In dieser Übung soll ein einfacher 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 |
|---------------------------|-----------------------|--|
| <code>nop</code> | 0000 0000 | Keine Operation. |
| <code>rac rd</code> | 0100 0 rd=0bxxx | Verschiebt den Inhalt des Akkumulators in ein Register. |
| <code>wac rd</code> | 0100 1 rd=0bxxx | Verschiebt den Inhalt eines Registers in den Akkumulators. |
| <code>mal/mah</code> | 0000 00 {01/10} | Schreibt den Inhalt des 8-Bit Akkumulators in die untere/obere Hälfte des 16-Bit Memory-Address-Registers (MAR). |
| <code>jmp</code> | 0000 0011 | Sprung zu der Programmadresse, die im MAR gespeichert ist. |
| <code>lds/ldu imm4</code> | 00{11/10} imm4=0bxxxx | Lädt die Vorzeichen-behaftete/-lose 4-Bit Konstante in den Akkumulator. |
| <code>add/sub rd</code> | 0101 {0/1} rd=0bxxx | Addiert/Subtrahiert den Inhalt des Registers rdzum/vom Akkumulator. |
| <code>lui imm4</code> | 0001 imm4=0bxxxx | Lädt die 4-Bit Konstante in die obere Hälfte der Bits des Akkumulators. |
| <code>ori imm4</code> | 0110 imm4=0bxxxx | Bitweise Disjunktion der vorzeichenlosen Konstante mit dem Akkumulator. |
| <code>addi imm4</code> | 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 das Registerfile. Nutzen Sie dazu das Grundgerüst `regfile.vhd`. Das Registerfile soll über 8-Registern, die über einen Multiplexer mit dem Ausgang verbunden sind, verfügen. Definieren Sie zunächst einen Array-Typ und beschreiben Sie anschließend das Verhalten.
2. 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.
3. Integrieren Sie nun die ALU zusammen mit dem Registerfile in eine Komponente `Execute`. Verwenden Sie dazu das Grundgerüst `execute.vhd`.
4. Realisieren Sie die Komponente für die Dekodierung. Verwenden Sie dazu das Grundgerüst `decode.vhd`. Diese Komponente liefert die Steuersignale für die `Execute`- und die `Fetch`-Stufe, welche anhand der Instruktionkodierung gesetzt werden müssen.
5. Implementieren Sie nun die `Fetch`-Stufe. Verwenden Sie dazu das Grundgerüst `fetch.vhd`. Instanzieren Sie hier den Programmspeicher. Legen Sie entsprechende Ressourcen für Instruktions- und das Speicheradressregister, sowie den Programmzähler an.
6. Integrieren Sie alle Komponenten im Grundgerüst `proc.vhd`.