7. Blatt	Fachgebiet Architektur eingebetteter Systeme **Rechnerorganisation Praktikum**	AES Embedded Systems Architecture WS2023/24

Ausgabe: 11. Dezember 2023

Abgaben
- Theorie: entfällt
- Praxis: 07. Januar 2024
- Rücksprache: 08./09. Januar 2024

Aufgabe 1: ALU Control (2 Punkte)

Entwerfen Sie die Steuereinheit für die ALU, wie sie in [1, ab Seite 246] vorgestellt wird. Wie im Buch beschrieben lässt sich die Steuerfunktion durch ganz einfache Logik abbilden (siehe Abbildung 1). Implementieren Sie die Funktionalität in der Datei `aluCtrl.vhd`.

Name	Typ	Art	Beschreibung
aluOp	std_logic_vector (1 downto 0)	in	Betriebsart der ALU
f	std_logic_vector (5 downto 0)	in	ALU-Optionen (sind Bestandteil der Assemblerinstruktion)
operation	std_logic_vector (3 downto 0)	out	Ansteuerungsausgang zur ALU

Tabelle 1: Entity: Ports

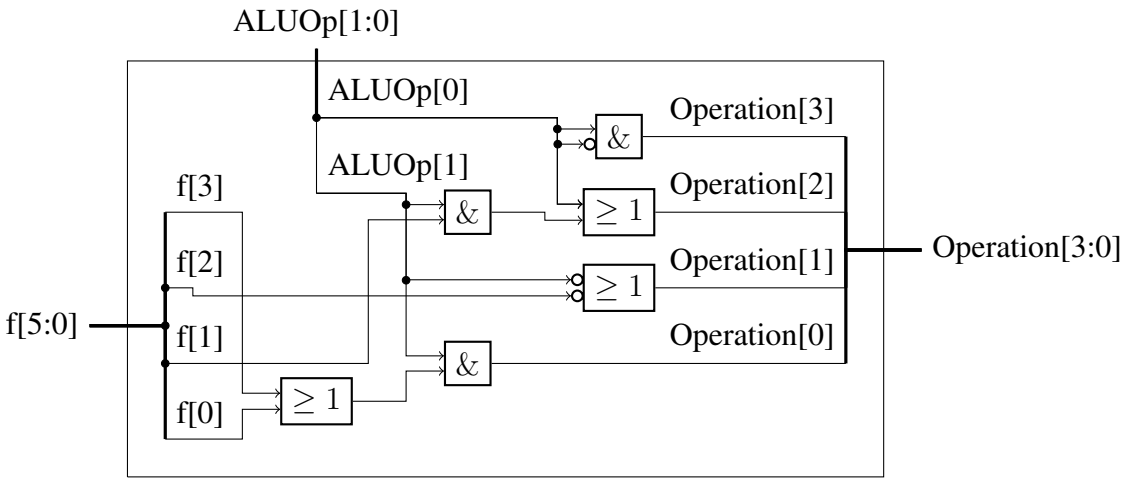

Abbildung 1: Schaltung für die Steuersignale der ALU aus [1]

Aufgabe 2: 1 Bit ALU (3 Punkte)

Die ALU ist das Herzstück der MIPS-CPU, sie führt fast alle in der CPU auftretenden Berechnungen durch. Zuerst soll eine 1-Bit-ALU in der Datei `alu_1bit.vhd` implementiert werden, jene ist eine leicht abgewandelte Variante der in [1] vorgestellten ALU. Implementieren Sie die Funktionalität in der `architecture structural` und testen Sie die Implementierung mit der Testbench `alu_1bit_tb`. Nutzen Sie dabei den in der Datei `adder_1bit.vhd` vorgegebenen Volladdierer.

Name	**Typ**	**Art**	**Beschreibung**
operation	std_logic_vector (1 downto 0)	in	durchzuführende Operation (bzw. Auswahl der richtigen Operation)
a	std_logic	in	erster Dateneingang
aInvert	std_logic	in	Flag, ob der erste Dateneingang invertiert werden soll
b	std_logic	in	zweiter Dateneingang
bInvert	std_logic	in	Flag, ob der zweite Dateneingang invertiert werden soll
carryIn	std_logic	in	Carry-In für den Addierer
less	std_logic	in	Eingang für das Ergebnis der *slt*-Operation
result	std_logic	out	Ergebnis der Operation
carryOut	std_logic	out	Carry-Out des Addierers
set	std_logic	out	Ausgang für das Ergebnis der *slt*-Operation

Tabelle 2: Entity: Ports

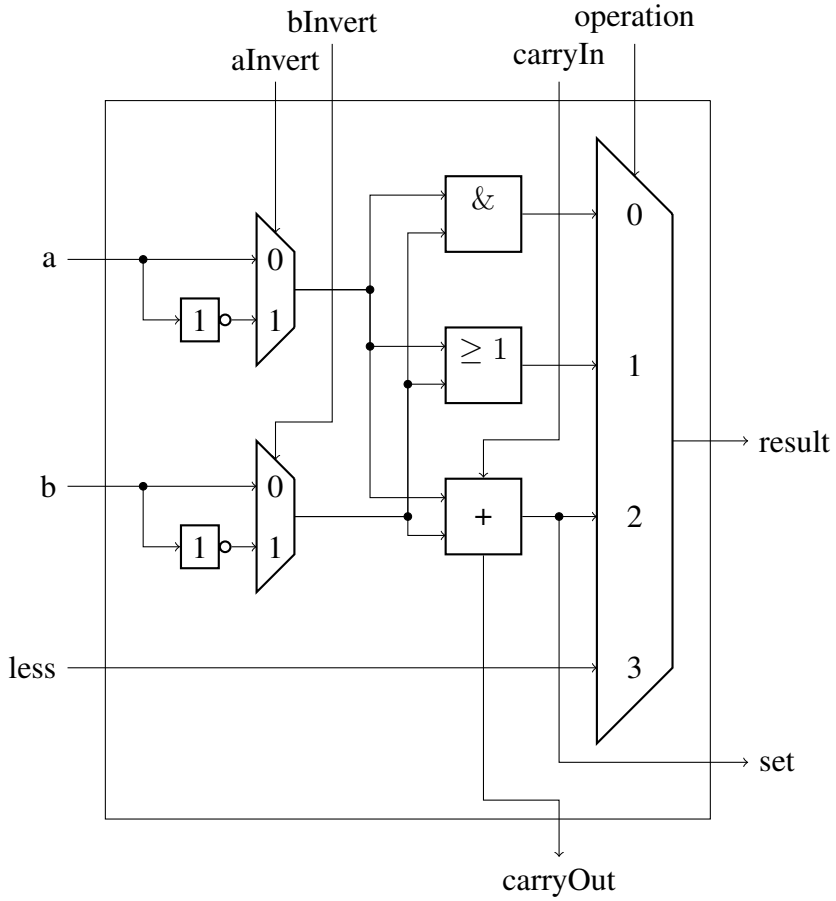

Abbildung 2: Aufbau der 1-Bit-ALU

Aufgabe 3: MIPS ALU (5 Punkte)

Damit auch n Bit breite Vektoren verarbeitet werden können, muss aus mehreren 1-Bit-ALUs eine Schaltkette gebauen werden. Desweiteren werden einige Ports nicht mehr benötigt, da die gebaute Schaltketten neu "verpackt" wird. Neu hinzu kommen jedoch weitere Aussagen (Flags) über die ausgeführte Operation. Die Funktionalität der n-Bit-ALU soll in der `architecture behavioral` der Datei `mipsAlu.vhd` implementiert werden.

Name	Typ	Art	Beschreibung
ctrl	std_logic_vector(3 downto 0)	in	durchzuführende Operation
a	std_logic_vector(WIDTH-1 downto 0)	in	erster Dateneingang
b	std_logic_vector(WIDTH-1 downto 0)	in	zweiter Dateneingang
result	std_logic_vector(WIDTH-1 downto 0)	out	Ergebnis der Operation
overflow	std_logic	out	Flag, ob es einen Überlauf gab
zero	std_logic	out	Flag, ob $result = 0$

Tabelle 3: Entity: Ports

Name	Typ	Art	Beschreibung
WIDTH	integer	generic	Breite der Eingangsvektoren

Tabelle 4: Entity: Generics

ctrl(X)	Steuerleitung
alu_ctrl(3)	Ainvert
alu_ctrl(2)	Binvert **Hinweis:** Sie können davon ausgehen, dass mit $Binvert = 1$ immer eine Subtraktion gemeint ist
alu_ctrl(1 downto 0)	AluOp

Tabelle 5: Aufschlüsselung des ALU `ctrl` Ports

1. Implementieren Sie eine generische Schaltkette aus den 1-Bit-ALUs um eine n-Bit-ALU zu realisieren.
   Hinweis:
   Für die slt-Operation wird der `set`-Port des MSB mit dem `less`-Port des LSB verbunden. Alle anderen `less`-Ports werden mit einer "0" gespeist, die entsprechenden `set`-Ports werden auf "open" gesetzt ("open" erklärt einen Ausgangsport für nicht verbunden).
2. Implementieren Sie eine `overflow`-Detektion.
   Hinweis:
   Wie funktioniert die Erkennung eines Überlaufs? Wie kann man dies formal beschreiben? Sie könnnen entweder die bekannte Formel auf Basis der Überträge nutzen oder sich aus den Vorzeichenbits von a, b und $result$ selbst für Addition und Subtraktion entsprechende Bedingungen herleiten.
3. Schreiben Sie zusätzliche Logik, welche das `zero`-Flag setzt, wenn $result = 0$ auftritt.
4. Fehlerfortpflanzung muss nicht implementiert werden. Bei der `overflow`-Detektion und dem `zero`-Flag darf keine Fehlerfortpflanzung erfolgen.
5. Validieren Sie das Verhalten Ihrer Implementation mithilfe der vorgegebenen Testbench `mipsAlu_tb`.

Literatur

[1] David A. Patterson and John L. Hennessy. *Rechnerorganisation und -entwurf.* Spektrum Akademischer Verlag, September 2005.