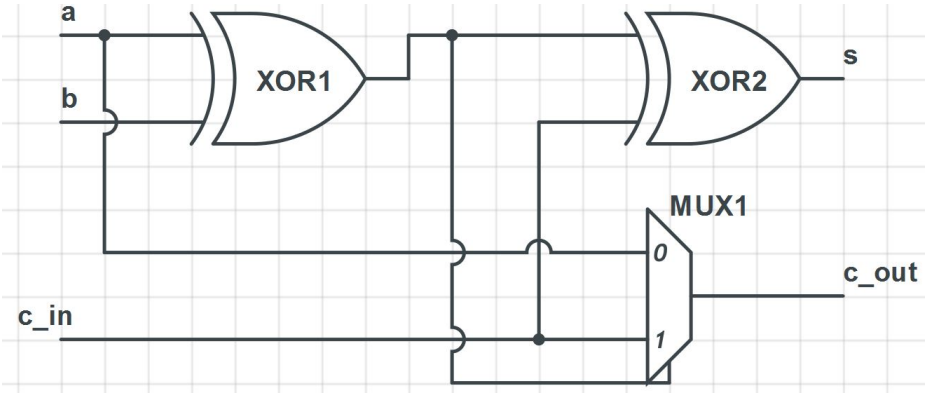
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

Systemarchitektur SS 2021

Lösungsskizze 5

Aufgabe 5.1: Verbesserter Volladdierer



Aufgabe 5.2: Carry-Lookahead Addierer

Zu zeigen: Die Operation $\circ$ ist assoziativ: $((g_1, p_1) \circ (g_2, p_2)) \circ (g_3, p_3) = (g_1, p_1) \circ ((g_2, p_2) \circ (g_3, p_3))$.

Beweis

$$
\begin{aligned}
& ((g_1, p_1) \circ (g_2, p_2)) \circ (g_3, p_3) \\
=& (g_1 + (p_1 * g_2), p_1 * p_2) \circ (g_3, p_3) && \text{(Def. } \circ\text{)} \\
=& ((g_1 + (p_1 * g_2)) + ((p_1 * p_2) * g_3), (p_1 * p_2) * p_3) && \text{(Def. } \circ\text{)} \\
=& ((g_1 + (p_1 * g_2)) + (p_1 * (p_2 * g_3)), (p_1 * p_2) * p_3) && \text{(Assoziativität } *\text{)} \\
=& ((g_1 + (p_1 * g_2)) + (p_1 * (p_2 * g_3)), p_1 * (p_2 * p_3)) && \text{(Assoziativität } *\text{)} \\
=& (g_1 + ((p_1 * g_2) + (p_1 * (p_2 * g_3))), p_1 * (p_2 * p_3)) && \text{(Assoziativität } +\text{)} \\
=& (g_1 + (p_1 * (g_2 + (p_2 * g_3)))), p_1 * (p_2 * p_3)) && \text{(Distributivität)} \\
=& (g_1, p_1) \circ (g_2 + (p_2 * g_3), p_2 * p_3) && \text{(Def. } \circ\text{)} \\
=& (g_1, p_1) \circ ((g_2, p_2) \circ (g_3, p_3)) && \text{(Def. } \circ\text{)}
\end{aligned}
$$

Aufgabe 5.3: Multiplikation von Zahlen in Zweier-Komplement-Darstellung

(a) Sei $a' = [\overline{a}] + 1$. a' benötigt eine Länge von n+1 Bits, weil sonst durch die Addition mit 1 ein Overflow auftreten könnte.

$$\begin{pmatrix} pp_0 \\ pp_1 \\ \vdots \\ pp_{n-1} \end{pmatrix} = \begin{pmatrix} a_{n-1}b_0 & a_{n-1}b_0 & \dots & a_{n-1}b_0 & a_{n-1}b_0 & a_{n-1}b_0 & a_{n-2}b_0 & \dots & a_1b_0 & a_0b_0 \\ a_{n-1}b_1 & a_{n-1}b_1 & \dots & a_{n-1}b_1 & a_{n-1}b_1 & a_{n-2}b_1 & a_{n-3}b_1 & \dots & a_0b_1 & 0 \\ \vdots & & & & \vdots & \vdots & \vdots & & & \vdots \\ a'_nb_{n-1} & a'_{n-1}b_{n-1} & \dots & a'_2b_{n-1} & a'_1b_{n-1} & a'_0b_{n-1} & 0 & \dots & 0 & 0 \end{pmatrix}$$

(b) Das Aufsummieren der n Partialprodukte kann, wie auf Folien 17 und 18 (Foliensatz 8) gezeigt, mit Hilfe von CSavAs und einem CLA mit Kosten $O(n^2)$ und Tiefe $O(\log_2(n))$ realisiert werden.

(c) $-10 = [a] = [110110], 18 = [b] = [010010], -[a] = [\overline{a}] + 1 = [001001] + 1 = [001010]$
Multiplikationsmatrix:

$$\begin{pmatrix} 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 0*0 & 1*0 & 1*0 & 0*0 \\ 1*1 & 1*1 & 1*1 & 1*1 & 1*1 & 1*1 & 1*1 & 0*1 & 1*1 & 1*1 & 0*1 & 0 \\ 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 0*0 & 1*0 & 1*0 & 0*0 & 0 & 0 \\ 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 0*0 & 1*0 & 1*0 & 0*0 & 0 & 0 & 0 \\ 1*1 & 1*1 & 1*1 & 1*1 & 0*1 & 1*1 & 1*1 & 0*1 & 0 & 0 & 0 & 0 \\ 0*0 & 0*0 & 0*0 & 1*0 & 0*0 & 1*0 & 0*0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Die Zeilen der Matrix müssen jetzt nur noch addiert werden:

$$\begin{aligned} &000000000000 \\ +&111111101100 \\ +&000000000000 \\ +&000000000000 \\ +&111101100000 \\ +&000000000000 \\ =\ &1111101001100 \end{aligned}$$

Aufgabe 5.4: Arithmetic Logic Unit

(a) *Schaltkreis:*

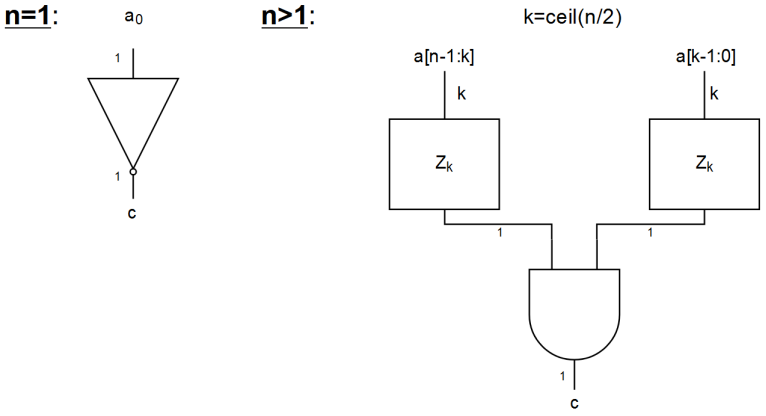


Abbildung 1: n-ZERO-Test

Basisfall n=1: $D(Z_1) = C(Z_1) = 1$, weil es nur ein einziges NOT-Gatter gibt.

Fall n>1:

$D(Z_n) = D(Z_{n/2}) + 1$
$C(Z_n) = 2 * C(Z_{n/2}) + 1$

- Behauptung:$D(Z_n) = \log(n) + 1$

  Beweis: per Induktion über n (Anzahl Bits)
  Induktionsanfang: n=1
  $\log(n) + 1 = \log(1) + 1 = \log(2^0) + 1 = 0 + 1 = 1 = D(Z_1)$
  Induktionsvoraussetzung:
  Gelte für ein bel., festes $n \in \mathbb{N}$ die Aussage $D(Z_n) = \log(n) + 1$.
  Induktionsschritt: $n \to 2n$
  $D(Z_{2n}) = D(Z_n) + 1$
  $= \log(n) + 1 + 1 \quad (I.V.)$
  $= \log(n) + \log(2) + 1$
  $= \log(2n) + 1$ ■

- Behauptung: $C(Z_n) = 2n - 1$

  Beweis: per Induktion über n (Anzahl Bits)
  Induktionsanfang: n=1
  $2 * 1 - 1 = 2 - 1 = 1 = C(Z_1)$
  Induktionsvoraussetzung:
  Gelte für ein bel., festes $n \in \mathbb{N}$ die Aussage $C(Z_n) = 2n - 1$.
  Induktionsschritt: $n \to 2n$
  $C(Z_{2n}) = 2 * C(Z_n) + 1$
  $= 2 * (2n - 1) + 1 \quad (I.V.)$
  $= 2 * 2n - 2 + 1$
  $= 2 * 2n - 1$ ■

(b) Test auf Gleichheit lässt sich durch Subtraktion der beiden Zahlen und anschließendes Testen auf 0 mit dem Tester aus Teilaufgabe (a) realisieren. Test auf Ungleichheit funktioniert nahezu analog. Das Ergebnis muss nur noch invertiert werden.

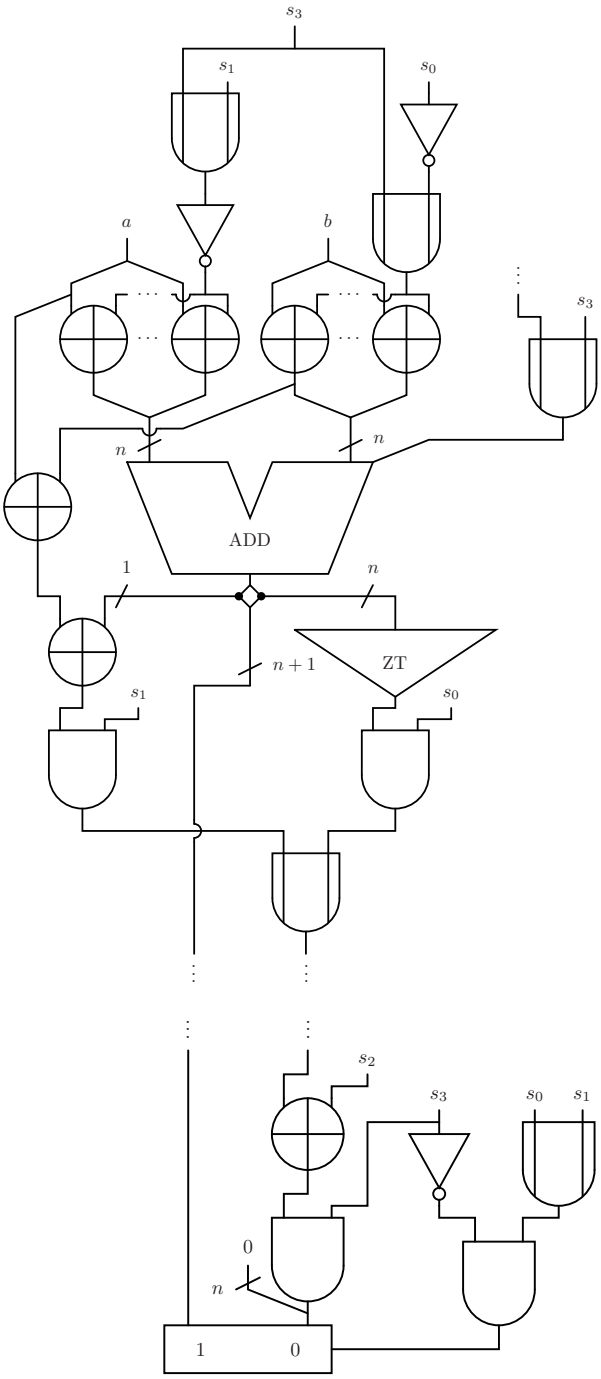
Man muss sich nun noch Gedanken machen, wie man den Operator $[a] < [b]$ am besten umsetzt. Auch hierfür kann man den Subtrahierer verwenden $[a] - [b] < 0$ mit anschließenden < 0 Vergleich. Für Zahlen in Zweikomplementdarstellung kann der < 0 Test durch $sign == 1$ realisiert werden wobei $sign$ das oberste (sprich sign-) Bit von $[a] - [b]$ meint. Zu beachten ist allerdings dass hier das oberste und nicht das 32. Bit genommen werden muss! Falls die Subtraktion nicht in die gegebenen 32 bit passt muss man das "33. Bit" aus dem carry berechnen. Dies geht zum Beispiel wie in der Grafik gezeigt indem man $a_n \oplus \neg b_n \oplus c_n$ berechnet (also quasi "von Hand" das oberste bit ausrechnet). $\leq$ ergibt sich dann als Kombination davon mit dem Gleichheitstest und die übrigen Operatoren aus der Vertauschung der Operanden.

Es bietet sich an die Tabelle um ein select-Bit zu erweitern, so dass man nun s_3, s_2, s_1, s_0 hat. Die alten Befehle lässt man so wie sie sind und setzt bei diesen s_3 jeweils auf 0.

Folgende zusätzliche Operationen werden realisiert:

s_3	s_2	s_1	s_0	
1	0	0	0	0
1	0	0	1	$=$
1	0	1	0	$<$
1	0	1	1	$\leq$
1	1	0	0	1
1	1	0	1	$\neq$
1	1	1	0	$\geq$
1	1	1	1	$>$

Um das korrekte Ergebnis zur Ausgabe zu leiten, müssen zusätzlich geeignete Multiplexer eingefügt werden. Ein Übersichtsbild zu den für die Vergleiche nötigen Teil der ALU finden Sie im Folgenden.


s_3
s_1
s_0
a
b
s_3
n
n
ADD
1
n
$n+1$
ZT
s_1
s_0
s_2
s_3
s_0
s_1
0
n
1
0

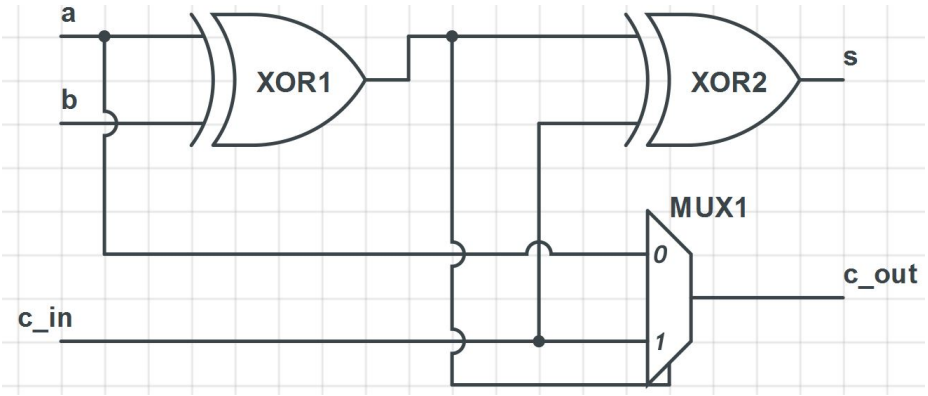
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

System Architecture SS 2021

Solution Sketch 5

Problem 5.1: Improved Full Adder



Problem 5.2: Carry-Lookahead Adder

To show: The operation $\circ$ is associative: $((g_1, p_1) \circ (g_2, p_2)) \circ (g_3, p_3) = (g_1, p_1) \circ ((g_2, p_2) \circ (g_3, p_3))$.

Beweis

$$
\begin{aligned}
& ((g_1, p_1) \circ (g_2, p_2)) \circ (g_3, p_3) \\
=& (g_1 + (p_1 * g_2), p_1 * p_2) \circ (g_3, p_3) && \text{(Def. } \circ\text{)} \\
=& ((g_1 + (p_1 * g_2)) + ((p_1 * p_2) * g_3), (p_1 * p_2) * p_3) && \text{(Def. } \circ\text{)} \\
=& ((g_1 + (p_1 * g_2)) + (p_1 * (p_2 * g_3)), (p_1 * p_2) * p_3) && \text{(Associativity } *\text{)} \\
=& ((g_1 + (p_1 * g_2)) + (p_1 * (p_2 * g_3)), p_1 * (p_2 * p_3)) && \text{(Associativity } *\text{)} \\
=& (g_1 + ((p_1 * g_2) + (p_1 * (p_2 * g_3))), p_1 * (p_2 * p_3)) && \text{(Associativity } +\text{)} \\
=& (g_1 + (p_1 * (g_2 + (p_2 * g_3)))), p_1 * (p_2 * p_3)) && \text{(Distributivity)} \\
=& (g_1, p_1) \circ (g_2 + (p_2 * g_3), p_2 * p_3) && \text{(Def. } \circ\text{)} \\
=& (g_1, p_1) \circ ((g_2, p_2) \circ (g_3, p_3)) && \text{(Def. } \circ\text{)}
\end{aligned}
$$

Problem 5.3: Multiplication of Numbers in Two's Complement Representation

(a) Let $a' = [\overline{a}] + 1$. a' requires $n+1$ bits, because otherwise, an overflow could occur due to the addition with 1.

$$\begin{pmatrix} pp_0 \\ pp_1 \\ \vdots \\ pp_{n-1} \end{pmatrix} = \begin{pmatrix} a_{n-1}b_0 & a_{n-1}b_0 & \dots & a_{n-1}b_0 & a_{n-1}b_0 & a_{n-1}b_0 & a_{n-2}b_0 & \dots & a_1b_0 & a_0b_0 \\ a_{n-1}b_1 & a_{n-1}b_1 & \dots & a_{n-1}b_1 & a_{n-1}b_1 & a_{n-2}b_1 & a_{n-3}b_1 & \dots & a_0b_1 & 0 \\ \vdots & & & & \vdots & \vdots & \vdots & & & \vdots \\ a'_nb_{n-1} & a'_{n-1}b_{n-1} & \dots & a'_2b_{n-1} & a'_1b_{n-1} & a'_0b_{n-1} & 0 & \dots & 0 & 0 \end{pmatrix}$$

(b) Summing up the n partial products can be realized using CSavAs and a CLA with a cost of $O(n^2)$ and a depth of $O(\log_2(n))$, as shown on slides 17 and 18 (slide deck 8).

(c) $-10 = [a] = [110110], 18 = [b] = [010010], -[a] = [\overline{a}] + 1 = [001001] + 1 = [001010]$
Multiplication matrix:

$$\begin{pmatrix} 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 0*0 & 1*0 & 1*0 & 0*0 \\ 1*1 & 1*1 & 1*1 & 1*1 & 1*1 & 1*1 & 1*1 & 0*1 & 1*1 & 1*1 & 0*1 & 0 \\ 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 0*0 & 1*0 & 1*0 & 0*0 & 0 & 0 \\ 1*0 & 1*0 & 1*0 & 1*0 & 1*0 & 0*0 & 1*0 & 1*0 & 0*0 & 0 & 0 & 0 \\ 1*1 & 1*1 & 1*1 & 1*1 & 0*1 & 1*1 & 1*1 & 0*1 & 0 & 0 & 0 & 0 \\ 0*0 & 0*0 & 0*0 & 1*0 & 0*0 & 1*0 & 0*0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Now, we have to add up the rows of the matrix:

$$\begin{aligned} &000000000000 \\ +&111111101100 \\ +&000000000000 \\ +&000000000000 \\ +&111101100000 \\ +&000000000000 \\ =\ &1111101001100 \end{aligned}$$

Problem 5.4: Arithmetic Logic Unit

(a) *Circuit:*

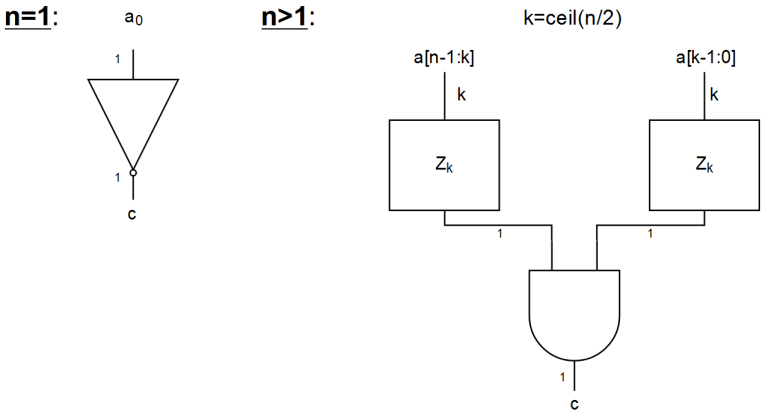


Figure 2: n-ZERO-Test

Base case n=1: $D(Z_1) = C(Z_1) = 1$, because there is only a single NOT gate.

Case n>1:

$D(Z_n) = D(Z_{n/2}) + 1$
$C(Z_n) = 2 * C(Z_{n/2}) + 1$

- Claim:$D(Z_n) = \log(n) + 1$

  Proof: by induction on n (number of bits)
  Base case: n=1
  $\log(n) + 1 = \log(1) + 1 = \log(2^0) + 1 = 0 + 1 = 1 = D(Z_1)$
  Induction hypothesis (IH):
  Assume the claim $D(Z_n) = \log(n) + 1$ holds for some $n \in \mathbb{N}$.
  Induction step: $n \to 2n$
  $D(Z_{2n}) = D(Z_n) + 1$
  $= \log(n) + 1 + 1 \quad (IH)$
  $= \log(n) + \log(2) + 1$
  $= \log(2n) + 1$ ■

- Claim: $C(Z_n) = 2n - 1$

  Proof: by induction on n (number of bits)
  Base case: n=1
  $2 * 1 - 1 = 2 - 1 = 1 = C(Z_1)$
  Induction hypothesis (IH):
  Assume the claim $C(Z_n) = 2n - 1$ holds for some $n \in \mathbb{N}$.
  Induction step: $n \to 2n$
  $C(Z_{2n}) = 2 * C(Z_n) + 1$
  $= 2 * (2n - 1) + 1 \quad (IH)$
  $= 2 * 2n - 2 + 1$
  $= 2 * 2n - 1$ ■

(b) Testing for equality can be realized by subtracting the two numbers and then testing for 0 with the circuit from subtask (a). Testing for inequality works almost analogously. The only difference is that the result has to be inverted.

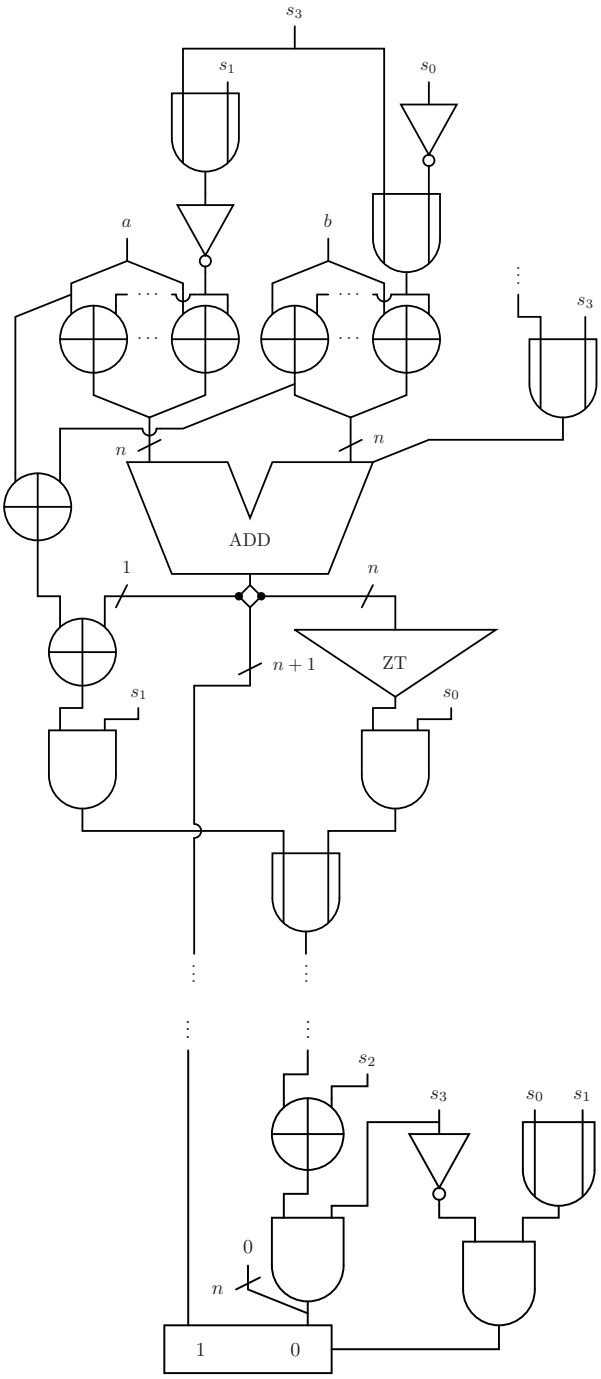
Now, we have to think about the best way to implement the operator $[a] < [b]$. For this, we can also use the subtractor $[a] - [b]$ with a subsequent < 0 comparison. For numbers in two's complement representation, the < 0 test can be realized by $sign == 1$, where $sign$ is the most significant (i.e. the sign) bit of $[a] - [b]$. Note however that we need to use the most significant bit and not the 32nd bit! If the subtraction does not fit into 32 bits, we have to calculate the "33rd bit" from the carry bit. This can be done, for example, as shown in the figure by computing $a_n \oplus \neg b_n \oplus c_n$ (i.e., computing the most significant bit "by hand"). We can then obtain $\leq$ as a combination of this with the equality test, and the remaining operators by exchanging the operands.

We can add one select bit to the table so that we now have s_3, s_2, s_1, s_0. We do not modify the existing functions, and we set s_3 to 0 for these functions.

We add the following additional functions:

s_3	s_2	s_1	s_0	
1	0	0	0	0
1	0	0	1	$=$
1	0	1	0	$<$
1	0	1	1	$\leq$
1	1	0	0	1
1	1	0	1	$\neq$
1	1	1	0	$\geq$
1	1	1	1	$>$

In order to route the correct result to the output, we have to insert additional multiplexers. An overview of the part of the ALU that is necessary for the comparisons can be found below.


s_3
s_1
s_0
a
b
s_3
n
n
ADD
1
n
$n+1$
ZT
s_1
s_0
s_2
s_3
s_0
s_1
0
n
1
0