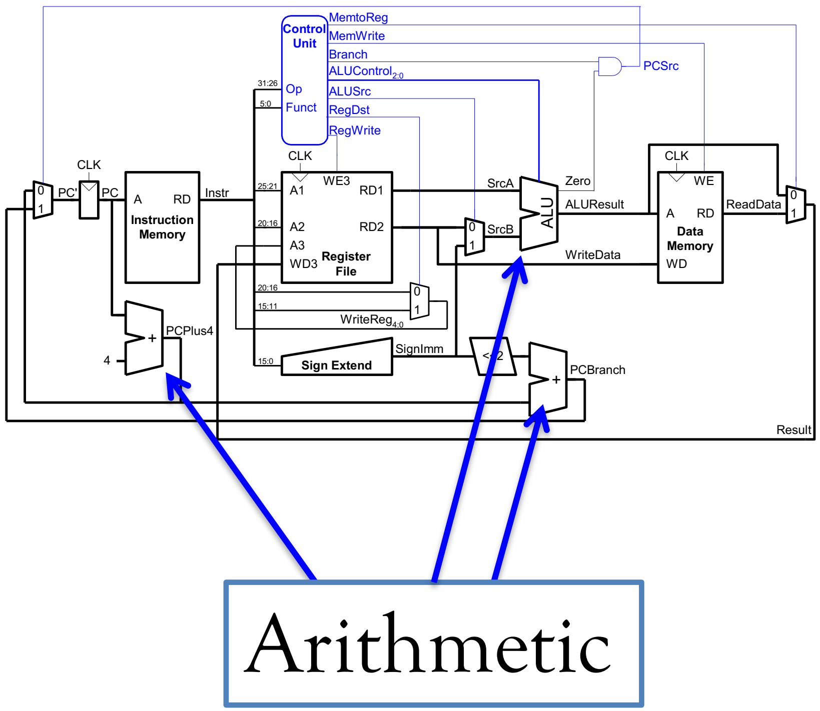
Arithmetic Circuits: Adders

Becker/Molitor, Chapter 9.2

Jan Reineke
Universität des Saarlandes

Roadmap: Computer architecture



1. Combinatorial circuits: Boolean Algebra/Functions/Expressions/Synthesis
2. Number representations
3. **Arithmetic Circuits:**
   Addition, Multiplication, Division, ALU

4. Sequential circuits: Flip-Flops, Registers, SRAM, Moore and Mealy automata
5. Verilog

6. Instruction Set Architecture
7. Data path & Control path

8. Performance: RISC vs. CISC, Pipelining, Memory Hierarchy

Representation of **natural numbers**

Let $a = a_{n-1} \ldots a_1 a_0$ be a sequence of numerals from the positional numeral system $(b, Z, \delta) = (2, \{0,1\}, id)$.

(We call such numbers binary numbers.)

Then the value $\langle a \rangle$ of a is:

$$< a > = < a_{n-1} \ldots a_1 a_0 > = \sum_{i=0}^{n-1} b^i \cdot \delta(a_i)$$

Adders

$$
\begin{tikzcd}
a = a_{n-1}\ldots a_0 \arrow[rr] \arrow[dd, "\langle\cdot\rangle"'] & & \boxed{\text{Adder}} \arrow[r] & s_n\ldots s_0 = s \arrow[dd, "\langle\cdot\rangle"'] \\
& b = b_{n-1}\ldots b_0 \arrow[ur] \arrow[dd, "\langle\cdot\rangle"'] & & \\
\langle a\rangle \arrow[rr] & & + \arrow[r] & \langle a\rangle + \langle b\rangle = \langle s\rangle \\
& \langle b\rangle \arrow[ur] & &
\end{tikzcd}
$$

Adder (with carry-in)

Given: 2 positive binary numbers

$$\langle a \rangle = \langle a_{n-1} \ldots a_0 \rangle,$$

$$\langle b \rangle = \langle b_{n-1} \ldots b_0 \rangle,$$

and a carry-in $c \in \{0,1\}$.

Wanted: Circuit computing the binary representation of

$$\langle s \rangle = \langle a \rangle + \langle b \rangle + c.$$

How many bits do we need to represent s?

Because $\langle a \rangle + \langle b \rangle + c \leq 2 \cdot (2^n - 1) + 1 = 2^{n+1} - 1$

$n+1$ bits suffice for s,

i.e., a circuit with $n+1$ outputs.

Definition: Adder

Definition (Adder):

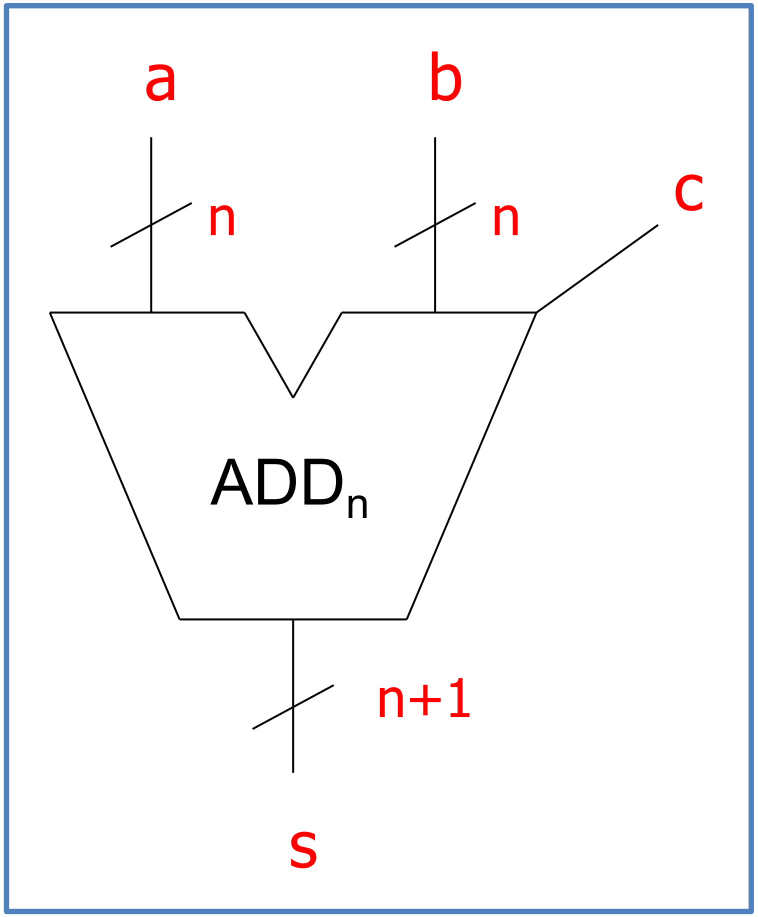
An **n-bit adder** is a circuit that computes the following Boolean function:

$+_n : \mathbf{B}^{2n+1} \rightarrow \mathbf{B}^{n+1}$,

$(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0, c) \rightarrow (s_n, \ldots, s_0)$ with

$\langle s \rangle = \langle s_n \ldots s_0 \rangle = \langle a_{n-1} \ldots a_0 \rangle + \langle b_{n-1} \ldots b_0 \rangle + c$

Schematic of an n-bit adder



Back to the basics: Grade school addition

Adding as you learned it
in grade school:

		1	0	1	1
+		0	1	1	0
+	1	1	1	0	0
	1	0	0	0	1

carry-in

Half adder (*HA*)

Half adders may be used to sum up two 1-Bit numbers *without* carry-in:

It computes the following function:

$$ha : \mathbf{B}^2 \rightarrow \mathbf{B}^2$$

with $ha(a_0, b_0) = (s_1, s_0)$

with $\langle s_1 s_0 \rangle = 2s_1 + s_0$

$= a_0 + b_0 \quad = \langle a_0 \rangle + \langle b_0 \rangle$

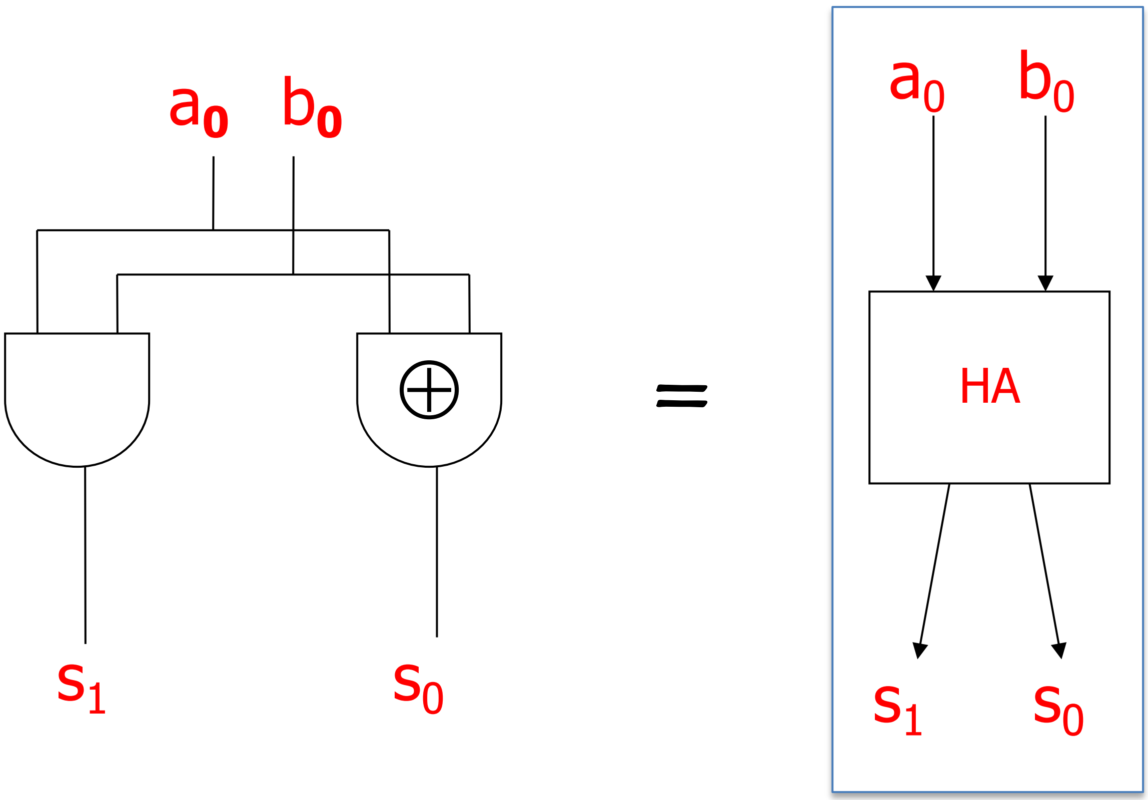
Truth table of the *HA*

a_0	b_0	ha_1	ha_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Thus:

$ha_0 = a_0 \oplus b_0$ $\quad$ $ha_1 = a_0 \wedge b_0$

Half adder circuit



Cost and depth of a half adder:

$$C(HA) = 2, \ \ \text{depth}(HA) = 1$$

Full adder (*FA*)

a_0	b_0	c	fa_1	fa_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

From the table we can derive:

$$fa_0 = a_0 \oplus b_0 \oplus c = ha_0(c, ha_0(a_0, b_0))$$

$$fa_1 = a_0 \wedge b_0 \vee c \wedge (a_0 \oplus b_0)$$

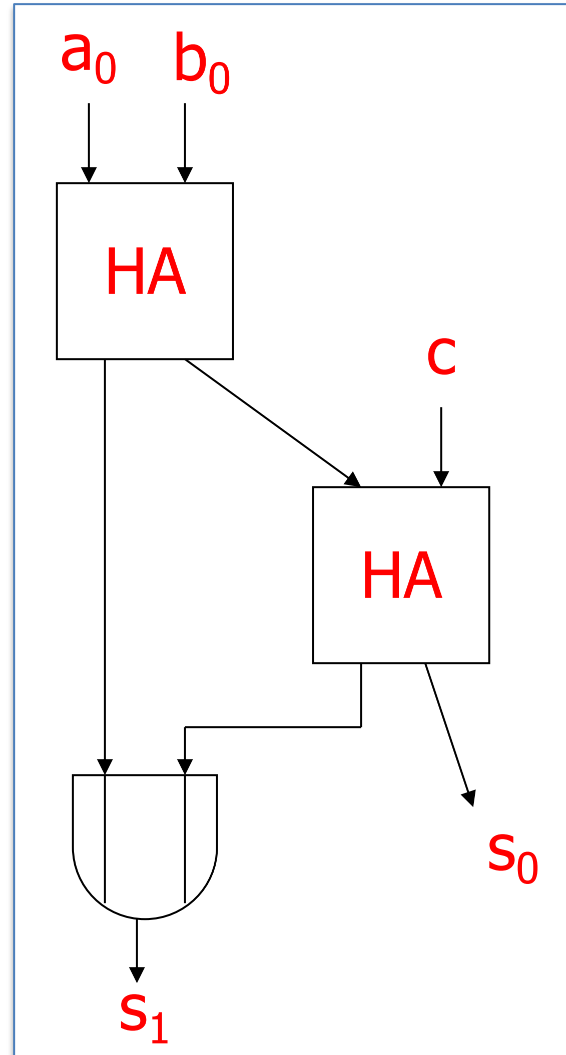
$$= ha_1(a_0, b_0) \vee ha_1(c, ha_0(a_0, b_0))$$

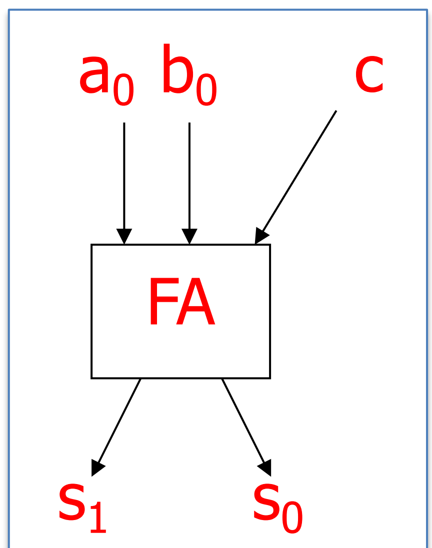
Full adder composed from *HAs*

From the table we can derive:

$$fa_0 = a_0 \oplus b_0 \oplus c = ha_0(c, ha_0(a_0, b_0))$$

$$\begin{aligned} fa_1 &= a_0 \wedge b_0 \vee c \wedge (a_0 \oplus b_0) \\ &= ha_1(a_0, b_0) \vee ha_1(c, ha_0(a_0, b_0)) \end{aligned}$$



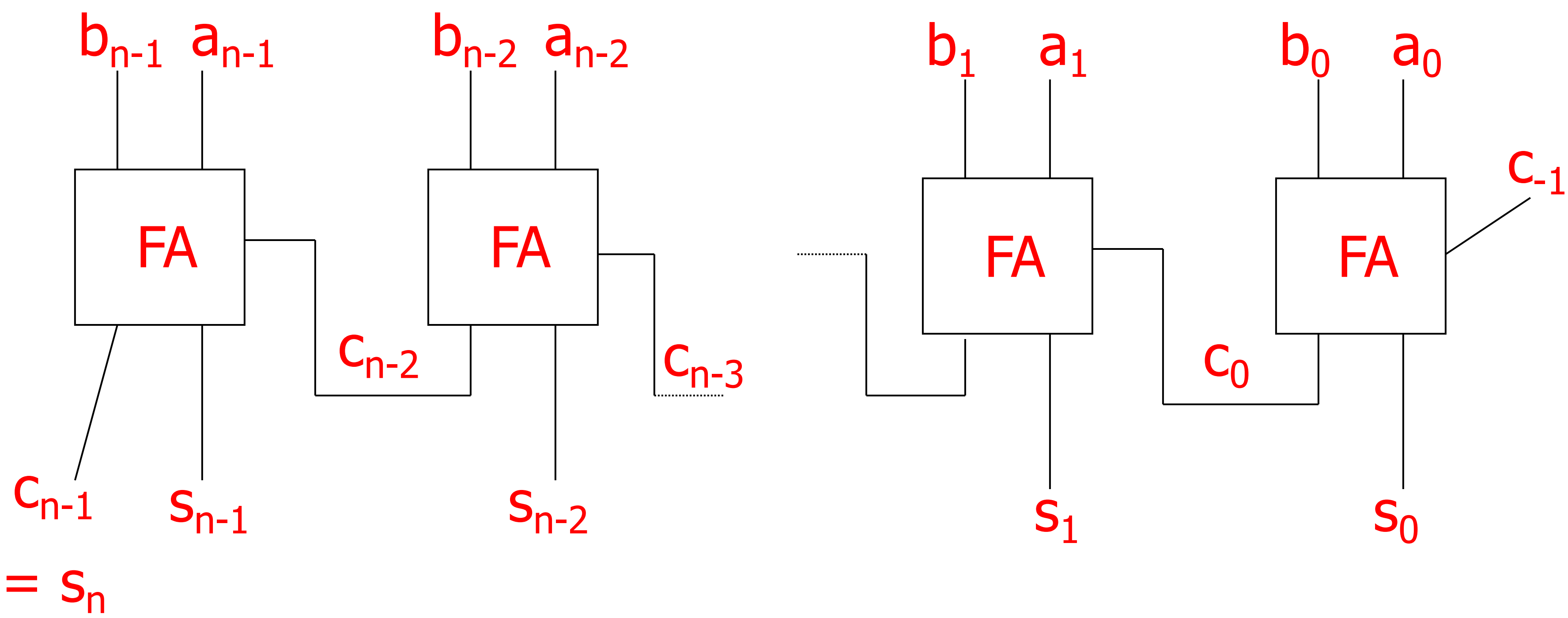


Cost and depth of a *FA*:

C(*FA*) = 5, depth(*FA*) = 3

Implementing the "school method": Ripple-carry adder (RC)

(also called Carry-chain adder)



Implementing the “school method”: Ripple-carry adder (RC)

Hierarchical construction:

(inductive definition)

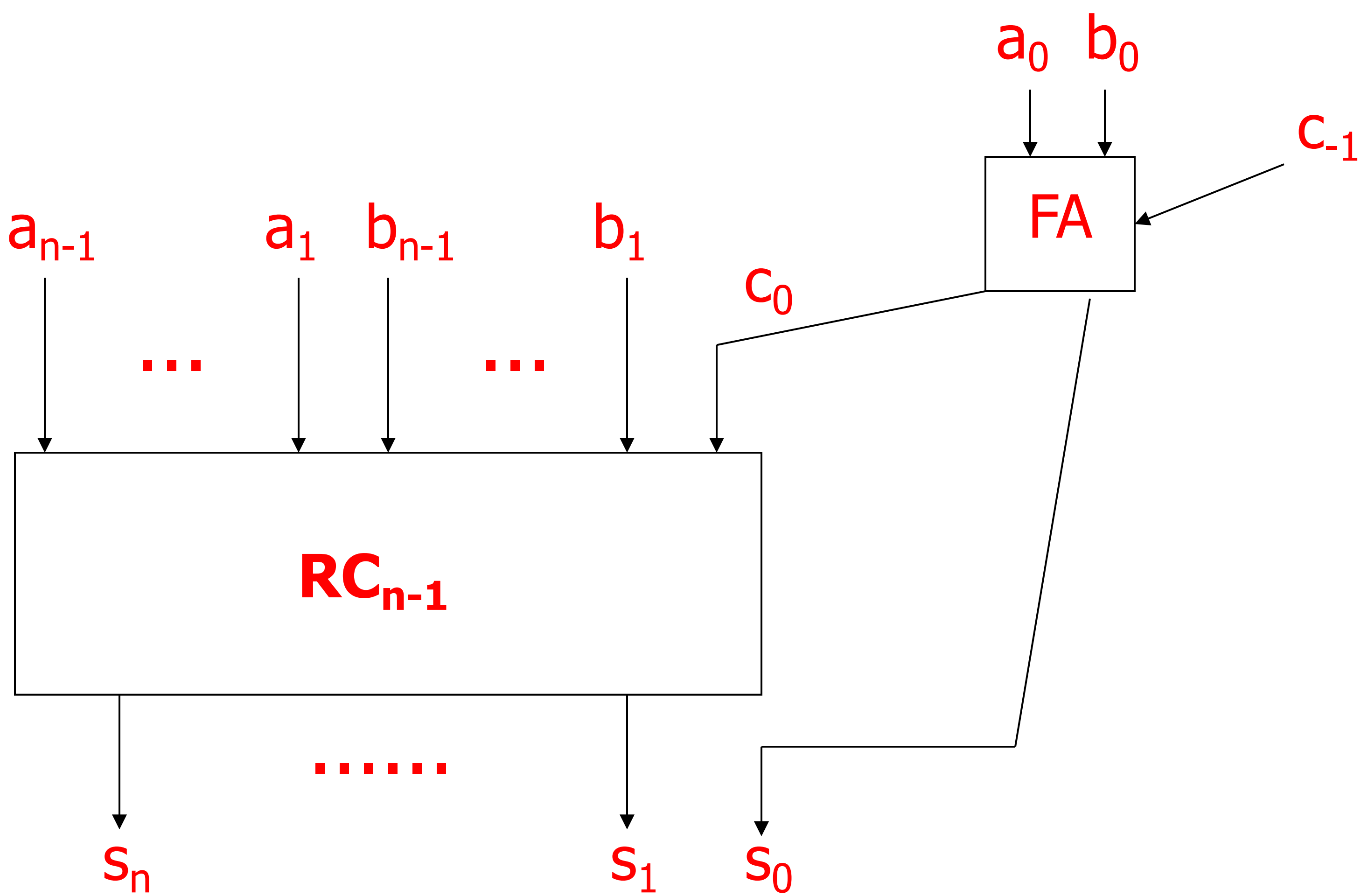
For n=1: $RC_1 = FA$

For n>1: Circuit RC_n is defined as follows

Notation:

We refer to the carry-in with c_{-1}, and the carry from position i to $i+1$ with c_i .

Recursive construction of an n-bit Ripple-carry adder (RC_n)



Correctness of the RC_n

Theorem: The RC_n circuit is an n-bit adder.

I.e., it computes the function

$+_n : \mathbf{B}^{2n+1} \rightarrow \mathbf{B}^{n+1}$,

$(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0, c) \rightarrow (s_n, \ldots, s_0)$ with

$\langle s \rangle = \langle s_n \ldots s_0 \rangle = \langle a_{n-1} \ldots a_0 \rangle + \langle b_{n-1} \ldots b_0 \rangle + c$

Correctness of the RC_n: Proof

Proof by induction:

- $n=1$: ✓
- $n-1 \rightarrow n$:

  Input to RC_n: $(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots b_0, c_{-1})$

  Show that the output $(s_n, \ldots, s_0)$ of RC_n satisfies

  $\langle s\rangle = \langle s_n \ldots s_0\rangle = \langle a_{n-1} \ldots a_0\rangle + \langle b_{n-1} \ldots b_0\rangle + c_{-1}$

We know that: $\langle c_0, s_0\rangle = a_0 + b_0 + c_{-1}$ (FA)

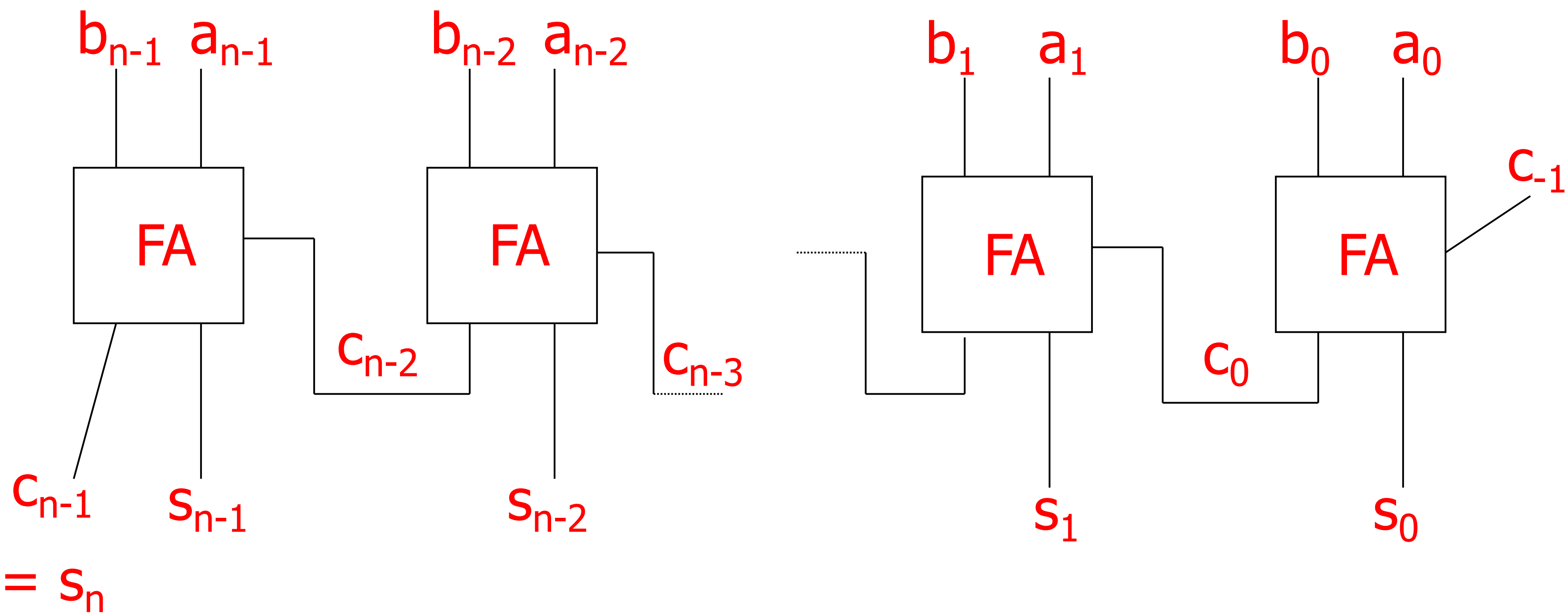
And by inductive hypothesis:

For RC_{n-1}: $\langle s_n \ldots s_1\rangle = \langle a_{n-1} \ldots a_1\rangle + \langle b_{n-1} \ldots b_1\rangle + c_0$

Putting it all together:

$$
\begin{aligned}
\langle s_n \ldots s_0\rangle \quad & = 2 \cdot \langle s_n \ldots s_1\rangle + s_0 \\
\text{(I.H.)} \quad & = 2 \cdot (\langle a_{n-1} \ldots a_1\rangle + \langle b_{n-1} \ldots b_1\rangle + c_0) + s_0 \\
\text{(FA)} \quad & = 2 \cdot \langle a_{n-1} \ldots a_1\rangle + a_0 + 2 \cdot \langle b_{n-1} \ldots b_1\rangle + b_0 + c_{-1} \\
& = \langle a\rangle + \langle b\rangle + c_{-1}
\end{aligned}
$$

Cost and depth of Ripple-carry adders



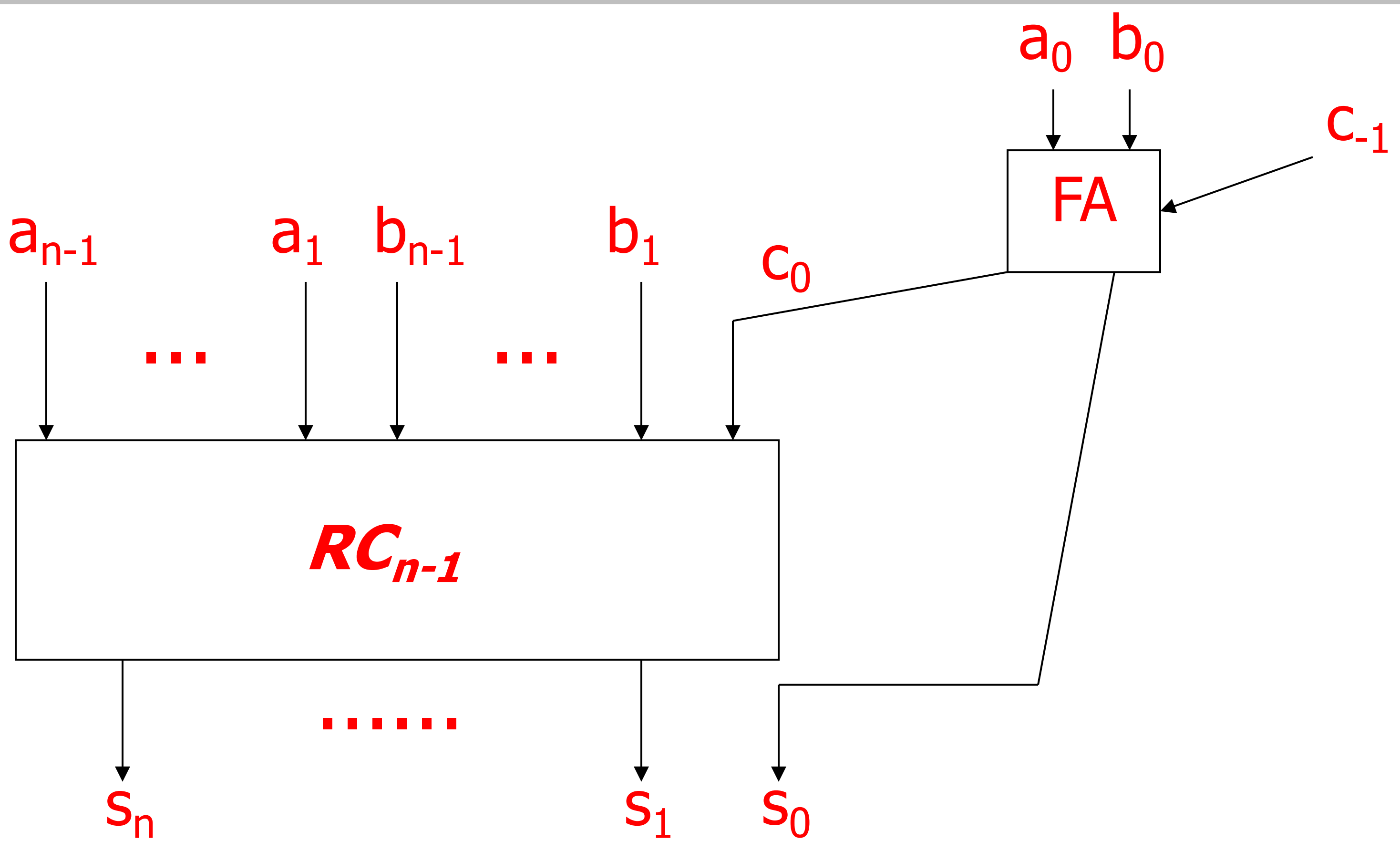
Cost of RC_n?

$C(RC_n) = n \cdot C(FA) = 5n$

Depth of RC_n?

$\text{depth}(RC_n) = 3 + 2(n-1) < 3n \text{ (for } n > 1)$

Cost and depth of Ripple-carry adders (recursive)



Cost of RC_n: $C(RC_n) = C(FA) + C(RC_{n-1}) = 5 + C(RC_{n-1})$

Depth of RC_n: $\text{depth}(RC_n) = 3 + \text{depth}(RC_{n-1})-1 = 3 + 2(n-1)$

Some more important circuits

- n-bit incrementer
- n-bit multiplexer

Definition: n-bit incrementer

An **n-bit incrementer** computes the following function:

$inc_n : \mathbf{B}^{n+1} \rightarrow \mathbf{B}^{n+1}$,

$(a_{n-1}, \ldots, a_0, c) \rightarrow (s_n, \ldots, s_0)$ with

$\langle s_n \ldots s_0 \rangle = \langle a \rangle + c$

Incrementer

An Incrementer is an adder with $b_i=0$ for all i.

→ Replaces the *FAs* in RC_n by *HAs*.

Cost and depth:

$C(INC_n) = n \cdot C(HA) = 2n$

$depth(INC_n) = n \cdot depth(HA) = n$

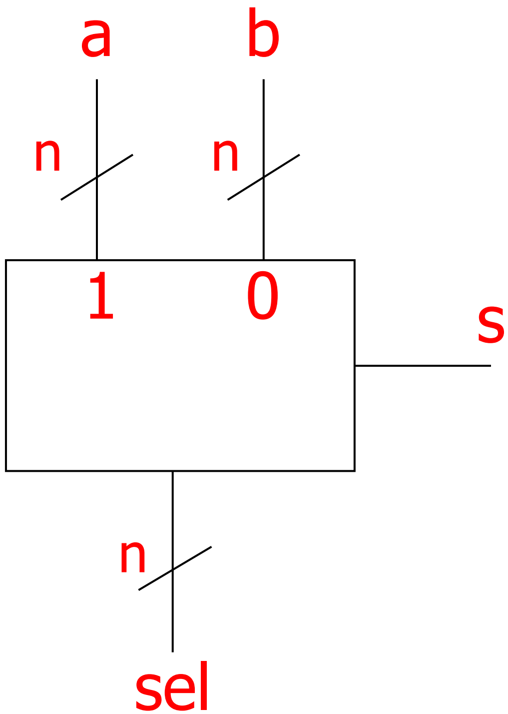
Definition: n-bit multiplexer

An **n-bit multiplexer** (MUX_n) is a circuit that computes the following function:

$sel_n : \mathbf{B}^{2n+1} \rightarrow \mathbf{B}^n$ with

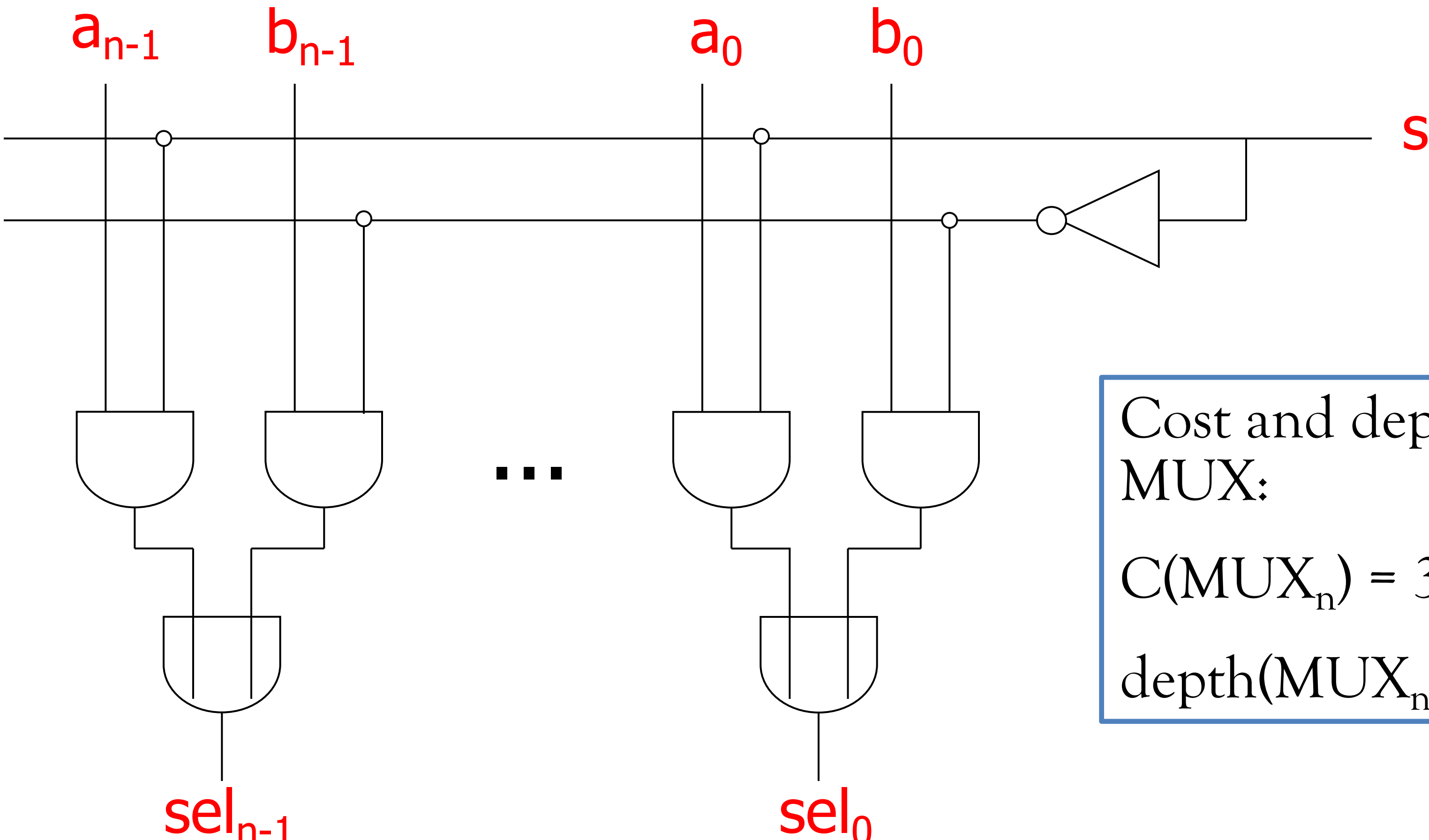
$$sel_n(a_{n-1}, \ldots, b_{n-1}, \ldots, b_0, s) = \begin{cases} (a_{n-1}, \ldots, a_0) : if\ s = 1 \\ (b_{n-1}, \ldots, b_0) : if\ s = 0 \end{cases}$$

$$(sel_n)_i = s \cdot \mathrm{a}_\mathrm{i} + \overline{s} \cdot \mathrm{b}_\mathrm{i}$$



Schematic of an n-bit multiplexer

Based on the equation: $(sel_n)_i = s \cdot \mathrm{a_i} + \bar{s} \cdot \mathrm{b_i}$



Cost and depth of a MUX:

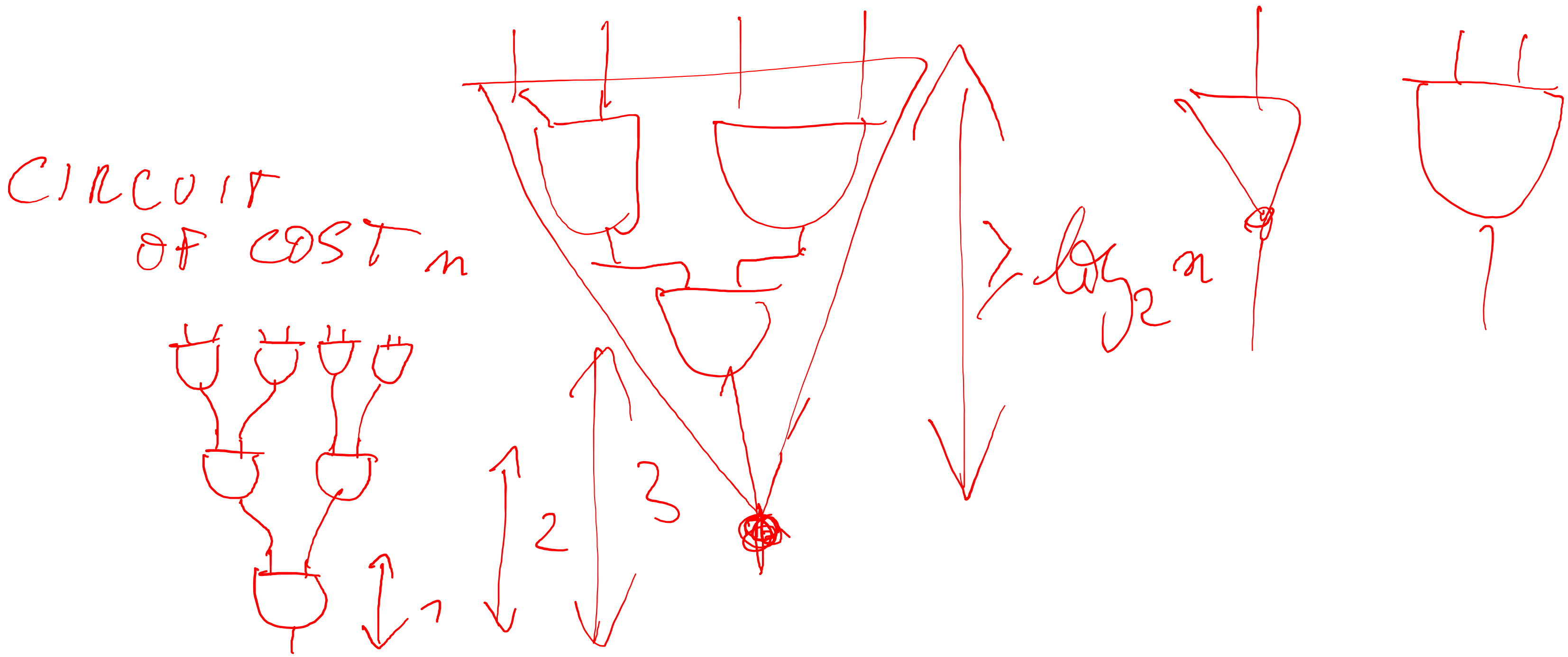
$C(MUX_n) = 3n + 1$

$depth(MUX_n) = 3$

Back to adders

Brainstorming:

- Are there cheaper and faster adders than RC_n?
- Can we construct a constant-depth adder, independently of *n*?



Back to adders

Brainstorming:

- Are there cheaper and faster adders than RC_n?
- Can we construct a constant-depth adder, independently of *n*?

a_{n-1} b_{n-1} a_{n-2} b_{n-2} … a_i b_i … a_0 b_0 c_{-1}

Lower bounds!

$C(+_n) \geq 2n$, $\quad depth(+_n) \geq \log(n) + 1$

Observation: Output s_n **depends** on all 2n+1 inputs!

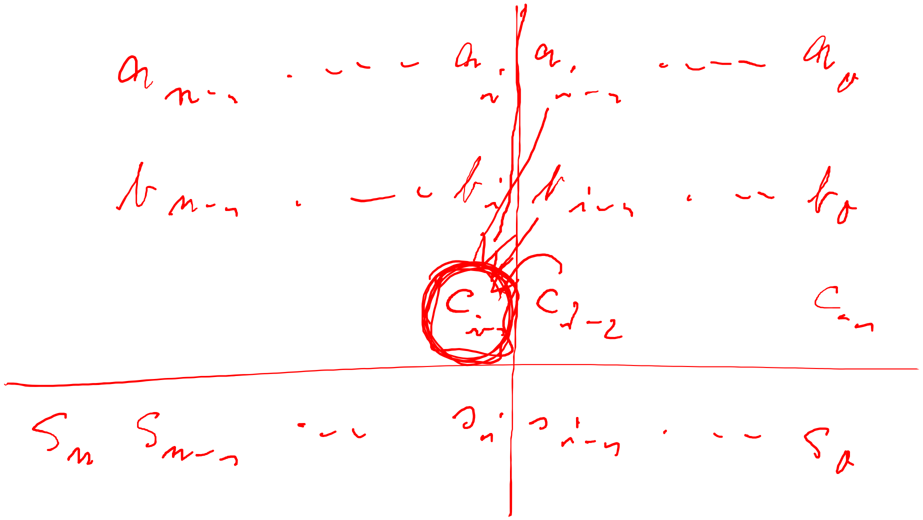
We use gates with at most 2 inputs.

Binary trees with 2n+1 leaves have 2n inner nodes.

Binary trees with n leaves have depth $\geq \lceil \log n \rceil$.

In the following, let $n = 2^k$.

Brainstorming: Faster adders



Idea: „Divide and Conquer“:
Employ parallel processing to reduce the depth!

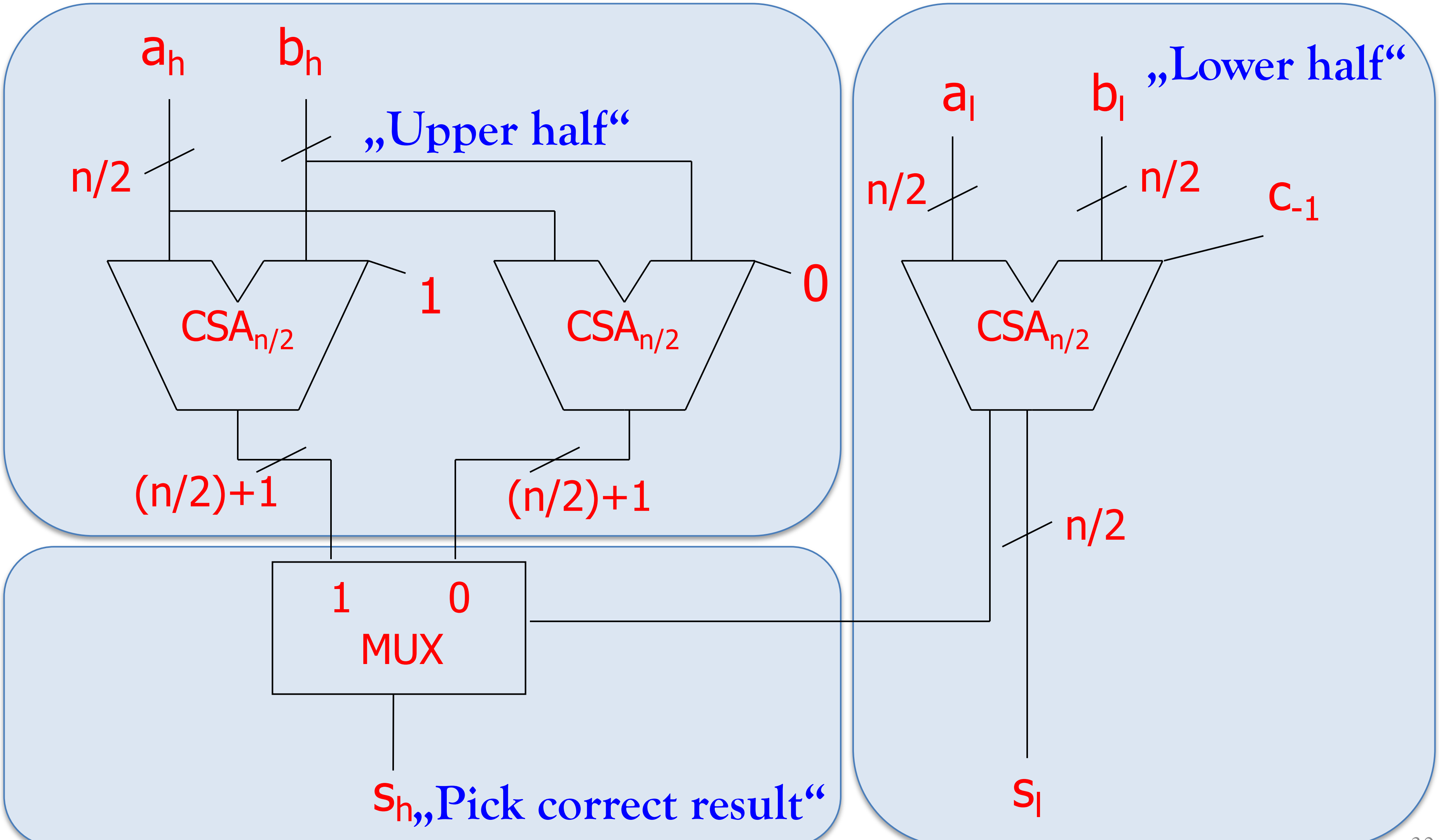
$$
\begin{array}{rcccc}
 & a_{n-1}\ldots a_{\frac{n}{2}} & \vert & a_{\frac{n}{2}-1}\ldots a_0 \\
+ & b_{n-1}\ldots b_{\frac{n}{2}} & \vert & b_{\frac{n}{2}-1}\ldots b_0 \\
+ & c_{\frac{n}{2}-1} & \vert & c_{-1} \\
\hline
s_n & s_{n-1}\ldots s_{\frac{n}{2}} & \vert & s_{\frac{n}{2}-1}\ldots s_0
\end{array}
$$

More precisely:
Compute **upper** and **lower** half of result in parallel.

Problem:
Dependency of the upper half of the result on carry from lower half.

Solution:
Computer upper half for both possible values of the carry and pick the correct one later.

Schematic of a conditional-sum adder (CSA_n)



On the complexity of the CSA_n

- We have: $CSA_1 = FA$.
- A CSA_n consists of 3 $CSA_{n/2}$.

Brainstorming: Depth of the CSA_n

How does depth(CSA_n) evolve depending on *n*?

$$\begin{aligned}
\text{depth}(CSA_n) &= \text{depth}(CSA_{n/2}) + \text{depth}(MUX_{(n/2)+1}) \\
&= \text{depth}(CSA_{n/2}) + 3 \\
&= \text{depth}(CSA_{n/4}) + 3 + 3 \\
&= \text{depth}(CSA_{n/8}) + 3 + 3 + 3 \\
&= \text{depth}(CSA_{n/2\wedge k}) + 3k \\
&= \text{depth}(CSA_1) + 3k \qquad (n = 2^k,\ k = \log_2 n) \\
&= \text{depth}(FA) + 3k \\
&= 3(k+1) \\
&= 3 \log_2 n + 3
\end{aligned}$$

Depth of the CSA_n

Theorem (Depth of the CSA_n):

$$depth(CSA_n) = 3 \log_2 n + 3$$

Proof:
By induction over n.

- Induction base (n=1):

  $depth(CSA_1) = depth(FA) = 3.$

- Induction step (n>1):

$$\begin{aligned} depth(CSA_n) &= depth(CSA_{n/2}) + depth(MUX_{(n/2)+1}) \\ &= 3 \log_2 (n/2) + 3 + depth(MUX_{(n/2)+1}) && \textit{(inductive hypothesis)} \\ &= 3 \log_2 (n/2) + 3 + 3 && \textit{(depth of the multiplexer)} \\ &= 3 ((\log_2 n)-(\log_2 2)) + 3 + 3 \\ &= 3 ((\log_2 n)-1) + 3 + 3 \\ &= 3 \log_2 n + 3 \end{aligned}$$

Reminder:
We assume that *n* is a power of two.

Remember:
$\log (a/b) = (\log a) - (\log b)$

Lower bound on the cost of the CSA_n

How does the cost $C(CSA_n)$ evolve depending on n?

$$C(CSA_1) = C(FA) = 5$$

$$\begin{aligned}
C(CSA_n) &= 3 \cdot C(CSA_{n/2}) + C(MUX_{(n/2)+1}) \\
&= 3 \cdot C(CSA_{n/2}) + 3 \cdot n/2 + 4 \\
&\overset{(*)}{\geq} 3 \cdot C(CSA_{n/2}) \\
&\geq 3 \cdot 3 \cdot C(CSA_{n/4}) \\
&\geq 3^k \cdot C(CSA_{n/2\text{^}k}) \\
&= 3^k \cdot C(CSA_1) \\
&= 5 \cdot 3^{\log n}
\end{aligned}$$

Reminder:
$C(MUX_n) = 3n + 1$

(*) To derive a lower bound we ignore the multiplexer.

$(k = \log_2 n)$

Lower bound on the cost of the CSA_n

Theorem (Cost of the CSA_n):

$$C(CSA_n) \geq 5 \cdot 3^{\log n}$$

Proof (by induction over n):

Induction base (n = 1):

$$C(CSA_1) = C(FA) = 5 \geq 5 = 5 \cdot 3^{\log 1}$$

Induction step (n > 1):

$$\begin{aligned} C(CSA_n) &= 3 \cdot C(CSA_{n/2}) + C(MUX_{(n/2)+1}) \\ &\geq 3 \cdot C(CSA_{n/2}) \\ &\geq 3 \cdot 5 \cdot 3^{\log (n/2)} && \text{(inductive hypothesis)} \\ &= 5 \cdot 3 \cdot 3^{(\log n)-1} \\ &= 5 \cdot 3^{\log n} \end{aligned}$$

Lower bound on the cost of the CSA_n

What is $3^{\log n}$?

$$3^{\log n} = (2^{\log 3})^{\log n} = 2^{\log 3 \cdot \log n} = (2^{\log n})^{\log 3} = n^{\log 3}$$

$$n^{\log 3} \approx n^{1.58}$$

For example:

$$64^{\log 3} = 3^{\log 64} = 3^6 = 729$$

Exact cost of the CSA_n

Taking into account the cost of the multiplexer, the exact cost of the CSA_n is:

$$C(CSA_n) = 10n^{\log 3} - 3n - 2$$

Thus, the conditional-sum adder is
very fast, but also **pretty expensive**!

Questions: Are there adders with

- **linear cost** (like the ripple-carry adder), and
- **logarithmic depth** (like the conditional-sum adder)?

Excursion: Addition of numbers in two's complement

Can we use **n-bit adders** for numbers in two's complement?

Observation:

$[d_{n-1}\ldots d_0]_2 = \langle d_{n-2}\ldots d_0\rangle - d_{n-1}\cdot 2^{n-1}$ and $\langle d_{n-1}\ldots d_0\rangle = \langle d_{n-2}\ldots d_0\rangle + d_{n-1}\cdot 2^{n-1}$

So $\langle d_{n-1}\ldots d_0\rangle - [d_{n-1}\ldots d_0]_2 = d_{n-1}(2^{n-1}+2^{n-1}) = d_{n-1}2^n$.

I.e. $\langle d_{n-1}\ldots d_0\rangle \equiv [d_{n-1}\ldots d_0]_2 \pmod{2^n}$.

Excursion:
Addition of numbers in two's complement

Theorem:
Let $a, b \in \mathbf{B}^n$, $c_{n-1}, c_{-1} \in \mathbf{B}$ and $s \in \mathbf{B}^n$,
such that $\langle c_{n-1}, s\rangle = \langle a\rangle + \langle b\rangle + c_{-1}$.

Then: $[s]_2 \equiv [a]_2 + [b]_2 + c_{-1} \pmod{2^n}$.

Proof:

1. $[a]_2 \equiv \langle a\rangle \pmod{2^n}$, $[b]_2 \equiv \langle b\rangle \pmod{2^n}$, $[s]_2 \equiv \langle s\rangle \pmod{2^n}$
2. $\langle a\rangle + \langle b\rangle + c_{-1} = \langle c_{n-1}, s\rangle \equiv \langle s\rangle \pmod{2^n}$
3. $[a]_2 + [b]_2 + c_{-1} \overset{(1.)}{\equiv} \langle a\rangle + \langle b\rangle + c_{-1} \overset{(2.)}{\equiv} \langle s\rangle \overset{(1.)}{\equiv} [s]_2 \pmod{2^n}$

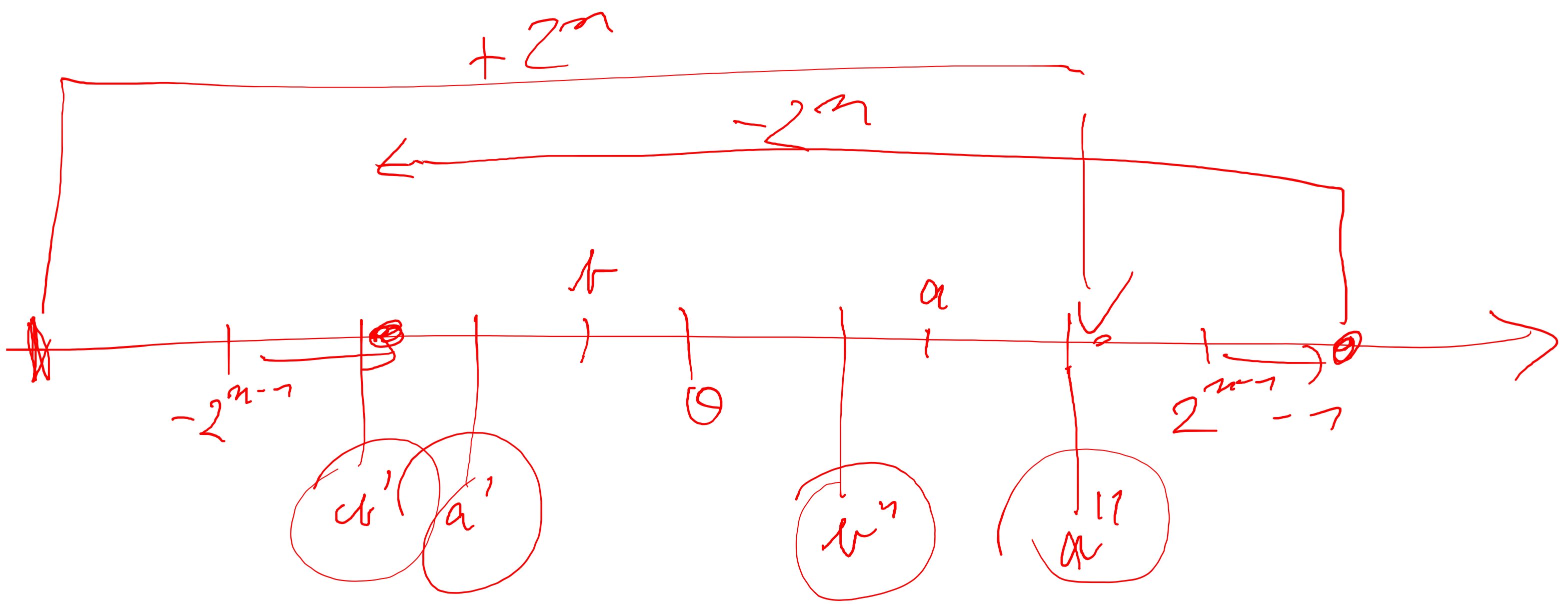
Excursion: Addition of numbers in two's complement

Observation:
The range of numbers covered by n-bit two's complement is $R_n = \{-2^{n-1}, \ldots, 2^{n-1}-1\}$
→ There are **no two** different values in R_n that are equal modulo 2^n.

Thus:
If the result of the addition is representable in n-bit two's complement, then it is computed correctly by an n-bit adder.

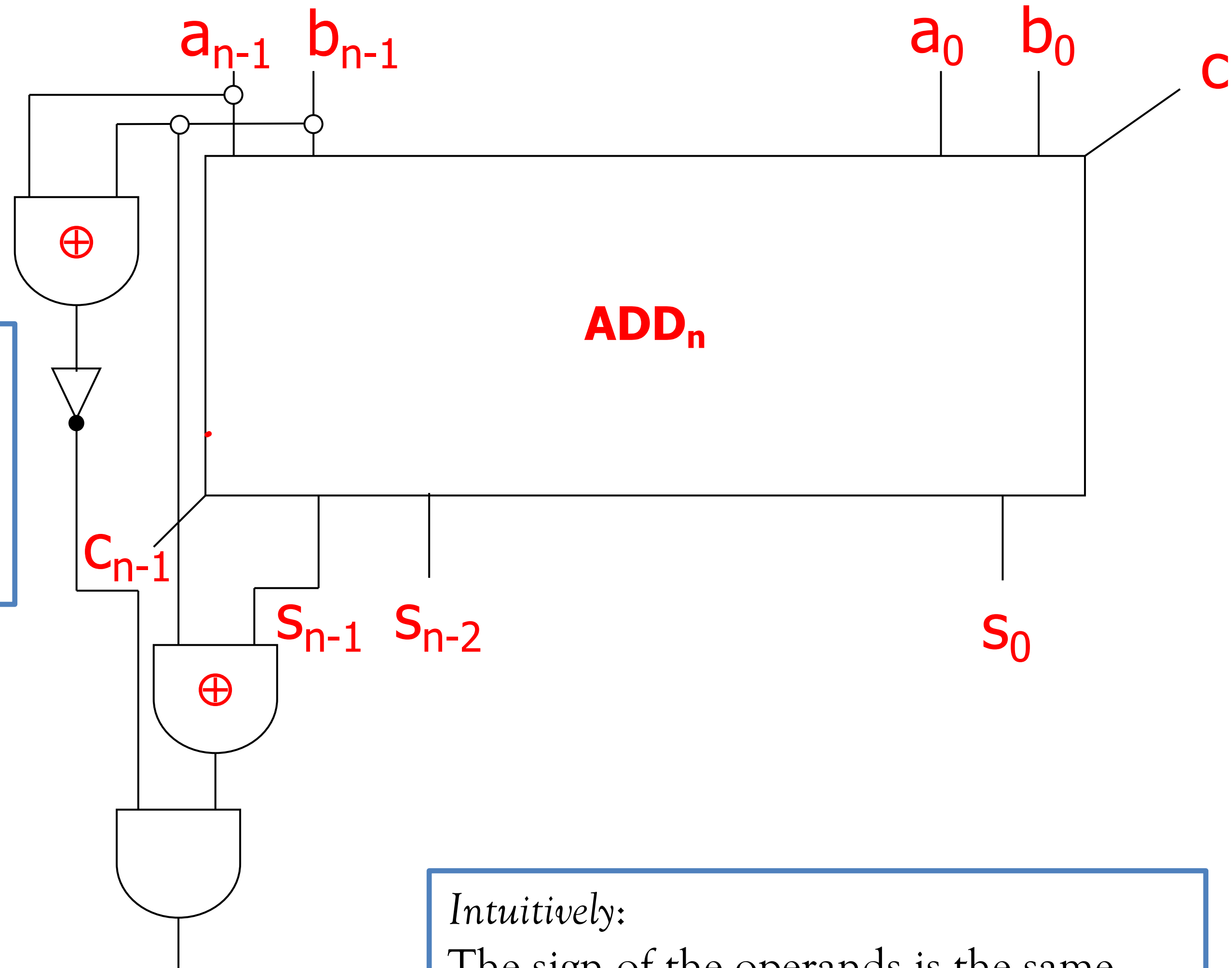
Excursion:
Addition of numbers in two's complement

Question: When is the result of the addition of two n-bit two's complement numbers **not** representable in n-bit two's complement?



Discovering an overflow of an n-bit adder

Circuit detects whether an overflow occurs:



$$a_{n-1} = b_{n-1} \neq s_{n-1}$$

Intuitively:
The sign of the operands is the same,
but different from the sign of the result.

Excursion: Addition of numbers in two's complement

Theorem:

Let $a,b \in \mathbf{B}^n$, c_{n-1}, $c_{-1} \in \mathbf{B}$ and $s \in \mathbf{B}^n$, such that $\langle c_{n-1},s\rangle = \langle a\rangle + \langle b\rangle + c_{-1}$.

Then:

1. $[a]_2+[b]_2+c_{-1} \notin R_n \Leftrightarrow (a_{n-1} = b_{n-1} \neq s_{n-1})$
2. $[a]_2+[b]_2+c_{-1} \in R_n \Rightarrow [a]_2+[b]_2+c_{-1} = [s]_2$

Proof of 1. via case distinction $[a]_2$, $[b]_2$ both positive, both negative, $[a]_2$ negative $[b]_2$ positive, $[a]_2$ positive $[b]_2$ negative.

Proof of 2. follows from the previous theorem.

Alternatively one can use the following overflow test:
$[a]_2+[b]_2+c_{-1} \notin R_n \Leftrightarrow c_{n-1} \neq c_{n-2}$

Carry-lookahead adder

Adder with

linear cost and logarithmic depth!

Approach: Fast precomputation of the carries c_i.

If the carries c_i are known, then s_i is simply $a_i \oplus b_i \oplus c_{i-1}$.

Computation of c_i via **parallel prefix computation**.

Parallel prefix computation

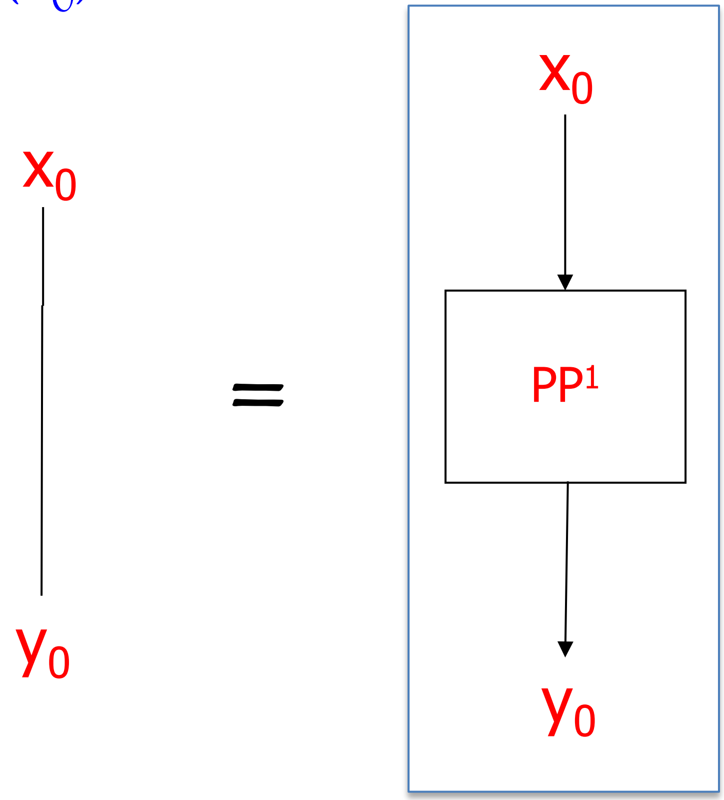
Definition:
Let M be a set and $\mathbf{o} : M \times M \rightarrow M$ an **associative** operation.
The **parallel prefix sum** PP^n: $M^n \rightarrow M^n$ is defined as follows:

$$PP^n\,(x_{n-1}, \ldots, x_0) = (x_{n-1} \mathbin{\mathbf{o}} x_{n-2} \ldots \mathbin{\mathbf{o}} x_0, \ldots, x_1 \mathbin{\mathbf{o}} x_0, x_0)$$

Parallel prefix computation: Recursive construction: Base case

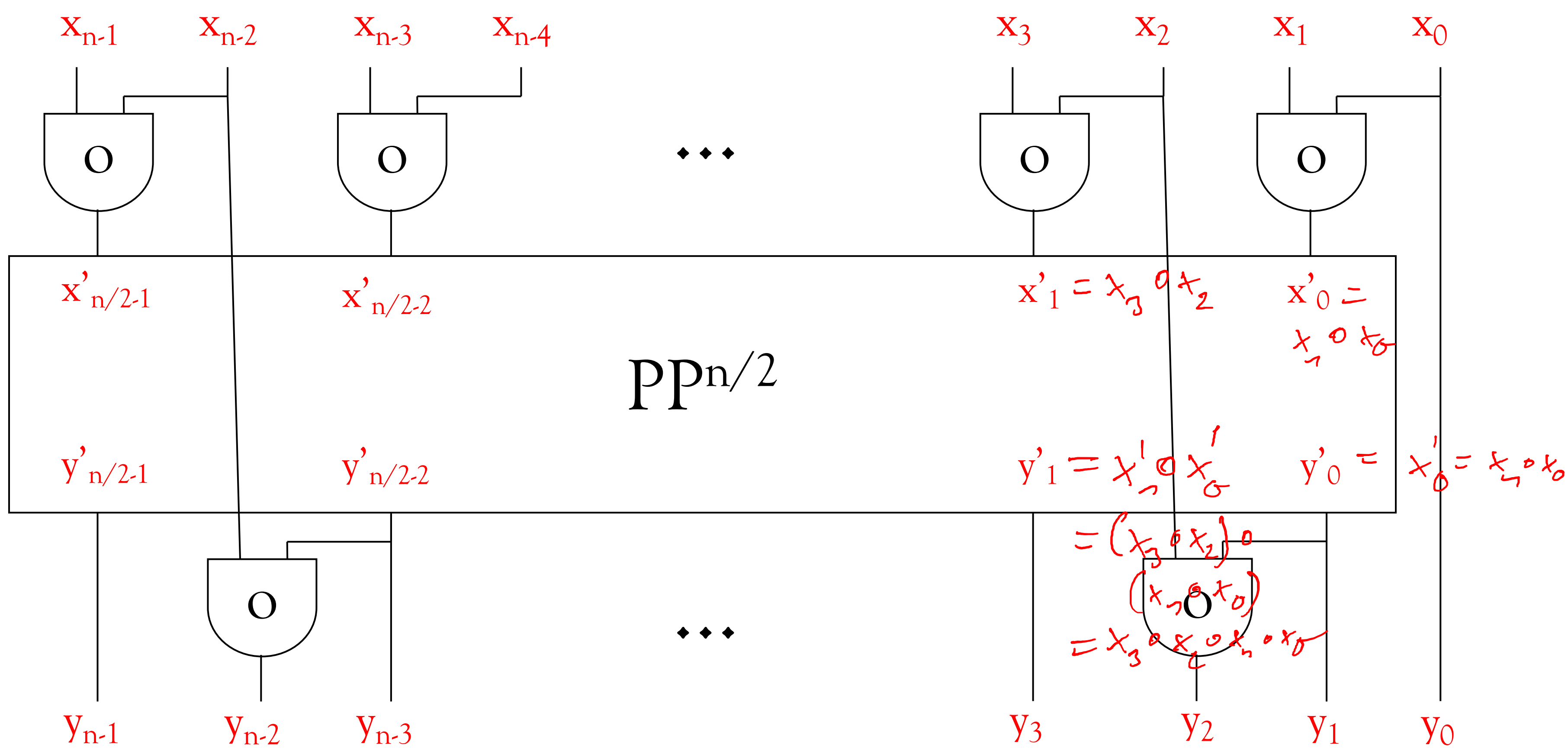
$PP^n (x_{n-1}, \ldots, x_0) = (x_{n-1} \text{ o } x_{n-2} \ldots \text{ o } x_0, \ldots, x_1 \text{ o } x_0, x_0)$

Base case: $PP^1 (x_0) = (x_0)$



Parallel prefix computation: PP^n based on PP$^{n/2}$

$PP^n(x_{n-1}, \dots, x_0) = (x_{n-1} \circ x_{n-2} \dots \circ x_0, x_{n-2} \circ x_{n-3} \circ \dots \circ x_0, \dots, x_0)$



Parallel prefix computation : Correctness (for $n = 2^i$)

Induction base ($i=0, n=1$): ✔

Induction step ($n/2 \rightarrow n$):

Inductive hypothesis: $y'_i = x'_i \circ x'_{i-1} \circ \ldots \circ x'_0$

For the *odd* outputs we have:

$$
\begin{aligned}
y_{2i+1} &= y'_i = x'_i \circ x'_{i-1} \circ \ldots \circ x'_0 && \text{(inductive hypothesis)} \\
&= (x_{2i+1} \circ x_{2i}) \circ \ldots \circ (x_1 \circ x_0) \\
&= x_{2i+1} \circ x_{2i} \circ \ldots \circ x_1 \circ x_0 && \text{(associativity)}
\end{aligned}
$$

For the *even* outputs (except $i = 0$) we have:

$$
\begin{aligned}
y_{2i} &= x_{2i} \circ y'_{i-1} = x_{2i} \circ (x'_{i-1} \circ \ldots \circ x'_0) && \text{(inductive hypothesis)} \\
&= x_{2i} \circ ((x_{2i-1} \circ x_{2i-2}) \circ \ldots \circ (x_1 \circ x_0)) \\
&= x_{2i} \circ x_{2i-1} \circ \ldots \circ x_1 \circ x_0 && \text{(associativity)}
\end{aligned}
$$

Cost of parallel prefix computation (for $n = 2^i$)

$\boxed{\textit{Cost}: C(PP^n) < 2n \cdot C(\circ)}$

Proof by induction over i:

- $i=0$, $n=1$:

$$C(PP^1) = 0 < 2 \cdot C(\circ)$$

- $n \rightarrow 2n$:

$$\begin{aligned} C(PP^{2n}) &= C(PP^n) + (2n-1) \cdot C(\circ) \\ &< 2n \cdot C(\circ) + (2n-1) \cdot C(\circ) \qquad \text{(I.H.)} \\ &< 2(2n) \cdot C(\circ) \end{aligned}$$

Depth of parallel prefix computation(for $n = 2^i$)

Depth: $\text{depth}(PP^n) < (2 \cdot \log_2 n) \cdot \text{depth}(\mathbf{o})$

Proof by induction over i:

- $i=0$, $n=1$: $\text{depth}(PP^1) = 0 < 2$
  $= (2 \cdot \log_2 2) \cdot \text{depth}(\mathbf{o})$

- $n \rightarrow 2n$: $\text{depth}(PP^{2n}) = \text{depth}(PP^n) + 2 \cdot \text{depth}(\mathbf{o})$
  $\leq^{(I.H.)} (2 \cdot \log n + 2) \cdot \text{depth}(\mathbf{o})$
  $= (2 \cdot (\log n + 1)) \cdot \text{depth}(\mathbf{o})$
  $= (2 \cdot \log (2n)) \cdot \text{depth}(\mathbf{o})$

Back to the adder: Precomputation of the carries

Distinguish **generated** and **propagated** carries:

$$
\begin{array}{ccccccc}
a_{n-1} & \ldots & a_j & \ldots & a_i & \ldots & a_0 \\
b_{n-1} & \ldots & b_j & \ldots & b_i & \ldots & b_0 \\
\ldots & \underline{c_j} & c_{j-1} & \ldots & \underline{c_{i-1}} & \ldots & c_{-1}
\end{array}
$$

$j \quad i$

1 1 1 1

0 0 0 0

$c_j \quad c_{i-1}$

$\boxed{p_{j,i}}$

1 0 1 1

0 0 0 1

$c_j \quad c_{i-1}$

$\cancel{p_{j,i}}$ $\cancel{g_{j,i}}$

Generated carry $g_{j,i}$ from i to j:

$c_j = 1$ *independently* of c_{i-1}.

Examples?

1 0 1 1

0 1 1 0

$c_j \quad c_{i-1}$

$\boxed{g_{j,i}}$

Propagated carry $p_{j,i}$ from i to j:

$c_j = 1$ *if and only if* also $c_{i-1} = 1$

Properties of generated and propagated carries

Carry c_j is obtained as follows:

$$c_j = g_{j,0} + p_{j,0} \cdot c_{-1}$$

For $i = j$ we have:

$$p_{i,i} = \underline{a_i \otimes b_i},$$

$$g_{i,i} = \underline{a_i \cdot b_i}.$$

For $i \neq j$ with $i \leq k < j$ we have:

$$g_{j,i} = \underline{g_{j,k+1}} + p_{j,k+1} \cdot g_{k,i},$$

$$p_{j,i} = p_{j,k+1} \cdot p_{k,i}.$$

Associative operator for the computation of $g_{j,i}$ and $p_{j,i}$

Define operator **o** as follows

$$(g, p) \text{ o } (g', p') = (g + p \cdot g', p \cdot p'),$$

so that

$$(g_{j,i}, p_{j,i}) = (g_{j,k+1}, p_{j,k+1}) \text{ o } (g_{k,i}, p_{k,i}).$$

Then we have:

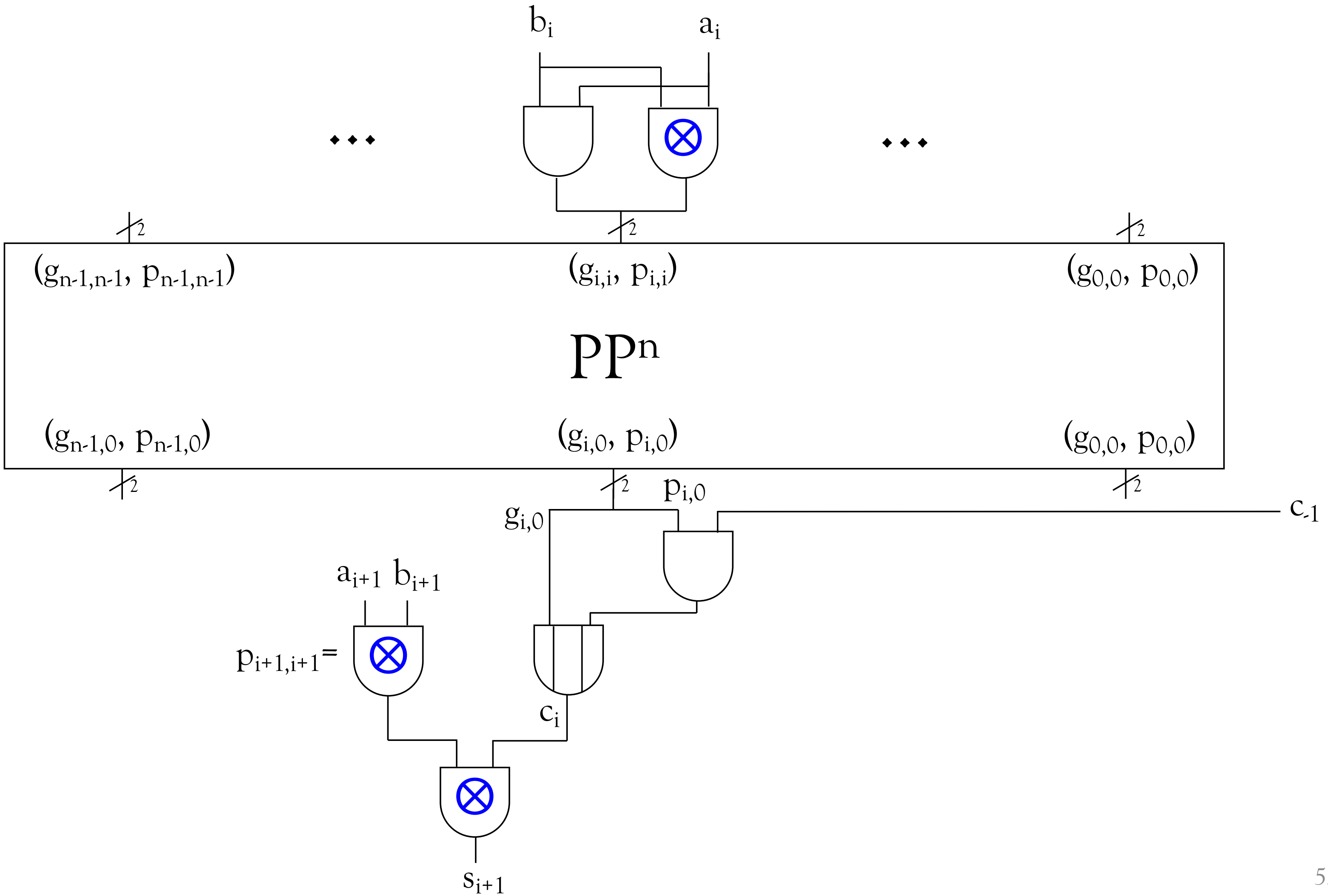
$$(g_{j,0}, p_{j,0}) = (g_{j,j}, p_{j,j}) \text{ o } \underbrace{\ldots}_{(g_{j-1,2},\, p_{j-1,2})} \text{ o } \underbrace{(g_{1,1}, p_{1,1}) \text{ o } (g_{0,0}, p_{0,0})}_{(g_{1,0},\, p_{1,0})}$$

The operator **o** is associative.

$(g_{j,0}, p_{j,0})$

→ Parallel prefix computation to determine $(g_{j,0}, p_{j,0})$

Carry-lookahead adder



Cost and depth of the CLA^n

Cost: $$\begin{aligned} C(CLA^n) &= C(PP^n) + 5n \\ &< 2n \cdot C(\circ) + 5n \\ &= 11n \end{aligned}$$

Depth: $$\begin{aligned} depth(CLA^n) &= depth(PP^n) + 4 \\ &\leq (2 \cdot \log n - 1) \cdot depth(\circ) + 4 \\ &= 4 \cdot \log n + 2 \end{aligned}$$

Summary: Circuits and their complexity

	Half adder	Full adder	Ripple-carry adder	Conditional-sum adder	Carry-lookahead adder
Cost	2	5	$5 \cdot n$	$10 \cdot n^{\log 3} - 3 \cdot n - 2$	$11 \cdot n$
Depth	1	3	$3+2 \cdot (n-1)$	$3 \cdot \log n + 3$	$4 \cdot \log n + 2$

	Incrementer	Multiplexer	arbitrary n-bit adder	Parallel prefix computation
Cost	$2 \cdot n$	$3 \cdot n+1$	$\geq 2 \cdot n$	$< 2 \cdot n \cdot C(\mathbf{o})$
Depth	n	3	$\geq \log n + 1$	$(2 \cdot \log n - 1) \cdot \text{depth}(\mathbf{o})$