

Please turn off your phones



Chapter 10: Signed Addition

Computer Structure - Spring 2004

© Dr. Guy Even

Tel-Aviv Univ.

Goals

- represent negative numbers

Goals

- represent negative numbers
- two's complement representation

Goals

- represent negative numbers
- two's complement representation
- add & subtract two's complement numbers

Goals

- represent negative numbers
- two's complement representation
- add & subtract two's complement numbers
- identify overflow and negative result

Signed numbers

- unsigned numbers - non-negative integers

Signed numbers

- **unsigned numbers** - non-negative integers
- **signed numbers** - positive/negative numbers

Signed numbers

- **unsigned numbers** - non-negative integers
- **signed numbers** - positive/negative numbers
- Many ways to represent signed numbers

Representation of signed numbers

- The number represented in **sign-magnitude** representation by $A[n - 1 : 0] \in \{0, 1\}^n$ and $S \in \{0, 1\}$ is

$$(-1)^S \cdot \langle A[n - 1 : 0] \rangle.$$

Representation of signed numbers

- The number represented in **sign-magnitude** representation by $A[n - 1 : 0] \in \{0, 1\}^n$ and $S \in \{0, 1\}$ is

$$(-1)^S \cdot \langle A[n - 1 : 0] \rangle.$$

- The number represented in **one's complement** representation by $A[n - 1 : 0] \in \{0, 1\}^n$ is

$$-(2^{n-1} - 1) \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

Representation of signed numbers

- The number represented in **sign-magnitude** representation by $A[n - 1 : 0] \in \{0, 1\}^n$ and $S \in \{0, 1\}$ is

$$(-1)^S \cdot \langle A[n - 1 : 0] \rangle.$$

- The number represented in **one's complement** representation by $A[n - 1 : 0] \in \{0, 1\}^n$ is

$$-(2^{n-1} - 1) \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

- The number represented in **two's complement** representation by $A[n - 1 : 0] \in \{0, 1\}^n$ is

$$-2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

Two's complement - examples

- We denote the number represented in two's complement representation by $A[n - 1 : 0]$ as follows:

$$[A[n - 1 : 0]] \triangleq -2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

Two's complement - examples

- We denote the number represented in two's complement representation by $A[n - 1 : 0]$ as follows:

$$[A[n - 1 : 0]] \triangleq -2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

- **Examples:**

Two's complement - examples

- We denote the number represented in two's complement representation by $A[n - 1 : 0]$ as follows:

$$[A[n - 1 : 0]] \triangleq -2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

- **Examples:**
 - $[0^n] = 0.$

Two's complement - examples

- We denote the number represented in two's complement representation by $A[n - 1 : 0]$ as follows:

$$[A[n - 1 : 0]] \triangleq -2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

- **Examples:**

- $[0^n] = 0.$

- $[0 \cdot x[n - 2 : 0]] = \langle x[n - 2 : 0] \rangle.$

Two's complement - examples

- We denote the number represented in two's complement representation by $A[n - 1 : 0]$ as follows:

$$[A[n - 1 : 0]] \triangleq -2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

- **Examples:**

- $[0^n] = 0.$

- $[0 \cdot x[n - 2 : 0]] = \langle x[n - 2 : 0] \rangle.$

- $[1 \cdot x[n - 2 : 0]] = -2^{n-1} + \langle x[n - 2 : 0] \rangle < 0.$

Two's complement - examples

- We denote the number represented in two's complement representation by $A[n - 1 : 0]$ as follows:

$$[A[n - 1 : 0]] \triangleq -2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

- **Examples:**

- $[0^n] = 0.$

- $[0 \cdot x[n - 2 : 0]] = \langle x[n - 2 : 0] \rangle.$

- $[1 \cdot x[n - 2 : 0]] = -2^{n-1} + \langle x[n - 2 : 0] \rangle < 0.$

- \Rightarrow MSB indicates the **sign**.

Two's complement - examples

- We denote the number represented in two's complement representation by $A[n - 1 : 0]$ as follows:

$$[A[n - 1 : 0]] \triangleq -2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

- **Examples:**

- $[0^n] = 0.$

- $[0 \cdot x[n - 2 : 0]] = \langle x[n - 2 : 0] \rangle.$

- $[1 \cdot x[n - 2 : 0]] = -2^{n-1} + \langle x[n - 2 : 0] \rangle < 0.$

- \Rightarrow MSB indicates the **sign**.

- $[1^n] = -1.$

Two's complement - examples

- We denote the number represented in two's complement representation by $A[n - 1 : 0]$ as follows:

$$[A[n - 1 : 0]] \triangleq -2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle.$$

- **Examples:**

- $[0^n] = 0.$

- $[0 \cdot x[n - 2 : 0]] = \langle x[n - 2 : 0] \rangle.$

- $[1 \cdot x[n - 2 : 0]] = -2^{n-1} + \langle x[n - 2 : 0] \rangle < 0.$

- \Rightarrow MSB indicates the **sign**.

- $[1^n] = -1.$

- $[1 \cdot 0^{n-1}] = -2^{n-1}.$

Two's complement - story

- The most common method for representing signed numbers is two's complement.

Two's complement - story

- The most common method for representing signed numbers is two's complement.
- Why? adding, subtracting, and multiplying signed numbers represented in two's complement representation is almost as easy as performing these computations on unsigned (binary) numbers.

Two's complement - story

- The most common method for representing signed numbers is two's complement.
- Why? adding, subtracting, and multiplying signed numbers represented in two's complement representation is almost as easy as performing these computations on unsigned (binary) numbers.
- We will discuss addition & subtraction.

Two's complement - story

- The most common method for representing signed numbers is two's complement.
- Why? adding, subtracting, and multiplying signed numbers represented in two's complement representation is almost as easy as performing these computations on unsigned (binary) numbers.
- We will discuss addition & subtraction.

DEF: Suppose that the string A represents the value x .

Negation means computing the string B that represents $-x$.

Question: Suggest circuit for negation with respect to sign-magnitude representation and one's complement representation.

Two's complement - notation

Two's complement - notation

T_n - the set of signed numbers that are representable in two's complement representation using n -bit binary strings.

Two's complement - notation

T_n - the set of signed numbers that are representable in two's complement representation using n -bit binary strings.

Claim:

$$T_n \triangleq \{-2^{n-1}, -2^{n-1} + 1, \dots, 2^{n-1} - 1\}.$$

Two's complement - notation

T_n - the set of signed numbers that are representable in two's complement representation using n -bit binary strings.

Claim:

$$T_n \triangleq \{-2^{n-1}, -2^{n-1} + 1, \dots, 2^{n-1} - 1\}.$$

Question: Prove the claim.

Two's complement - notation

T_n - the set of signed numbers that are representable in two's complement representation using n -bit binary strings.

Claim:

$$T_n \triangleq \{-2^{n-1}, -2^{n-1} + 1, \dots, 2^{n-1} - 1\}.$$

Question: Prove the claim.

Remark: T_n is not closed under negation: $-2^{n-1} \in T_n$ but $2^{n-1} \notin T_n$.

Two's complement - negation

Two's complement - negation

Claim:

$$- [A[n - 1 : 0]] = [\text{INV}(A[n - 1 : 0])] + 1.$$

Two's complement - negation

Claim:

$$- [A[n - 1 : 0]] = [\text{INV}(A[n - 1 : 0])] + 1.$$

Proof: Note that $\text{INV}(A[i]) = 1 - A[i]$. Hence,

$$\begin{aligned} [\text{INV}(A[n - 1 : 0])] &= -2^{n-1} \cdot \text{INV}(A[n - 1]) + \langle \text{INV}(A[n - 2 : 0]) \rangle \\ &= -2^{n-1} \cdot (1 - A[n - 1]) + \sum_{i=0}^{n-2} (1 - A[i]) \cdot 2^i \\ &= \underbrace{-2^{n-1} + \sum_{i=0}^{n-2} 2^i}_{=-1} + \underbrace{2^{n-1} \cdot A[n - 1] - \sum_{i=0}^{n-2} A[i] \cdot 2^i}_{=-[A[n-1:0]]} \\ &= -1 - [A[n - 1 : 0]]. \end{aligned}$$

QED.

A circuit for negating a two's complement number

A circuit for negating a two's complement number

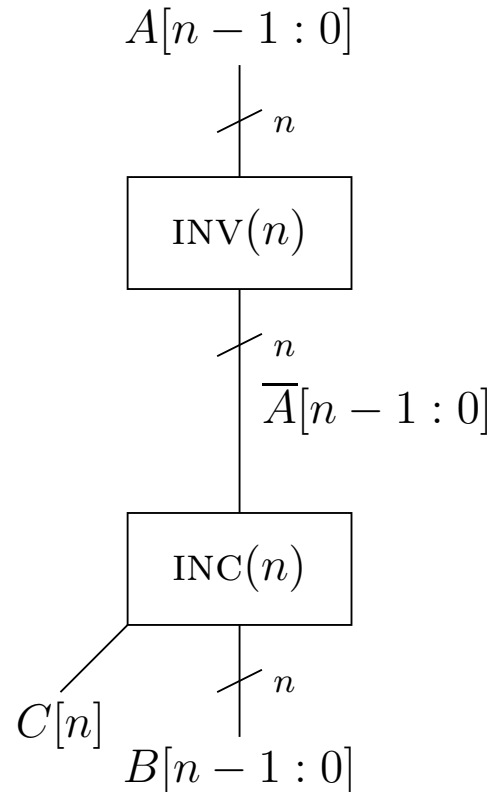
Claim:

$$- [A[n - 1 : 0]] = [\text{INV}(A[n - 1 : 0])] + 1.$$

A circuit for negating a two's complement number

Claim:

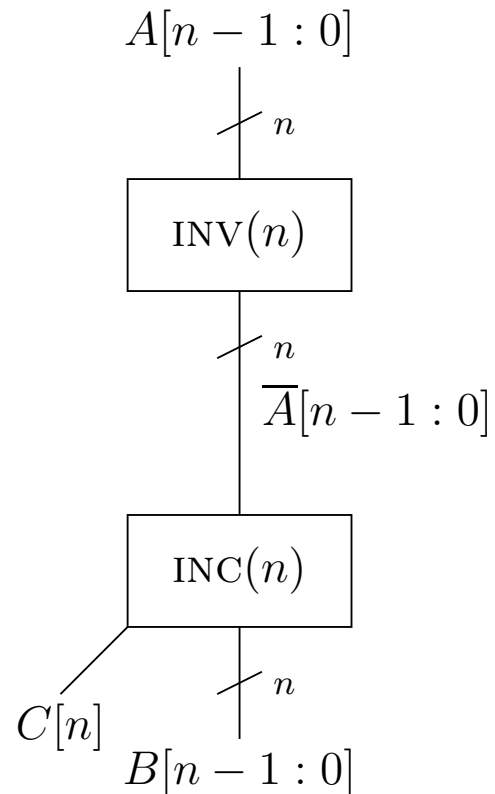
$$- [A[n - 1 : 0]] = [\text{INV}(A[n - 1 : 0])] + 1.$$



A circuit for negating a two's complement number

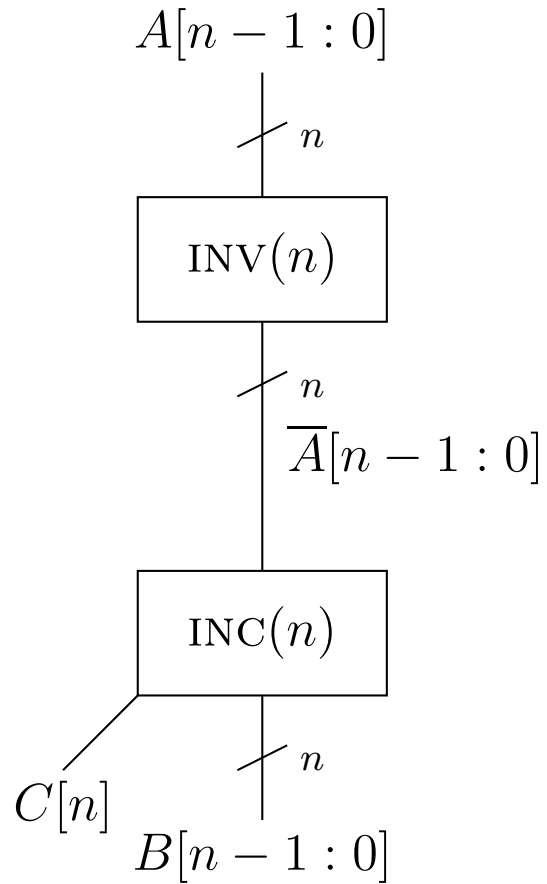
Claim:

$$- [A[n - 1 : 0]] = [\text{INV}(A[n - 1 : 0])] + 1.$$



Question: $[B[n - 1 : 0]] \stackrel{?}{=} - [A[n - 1 : 0]]$

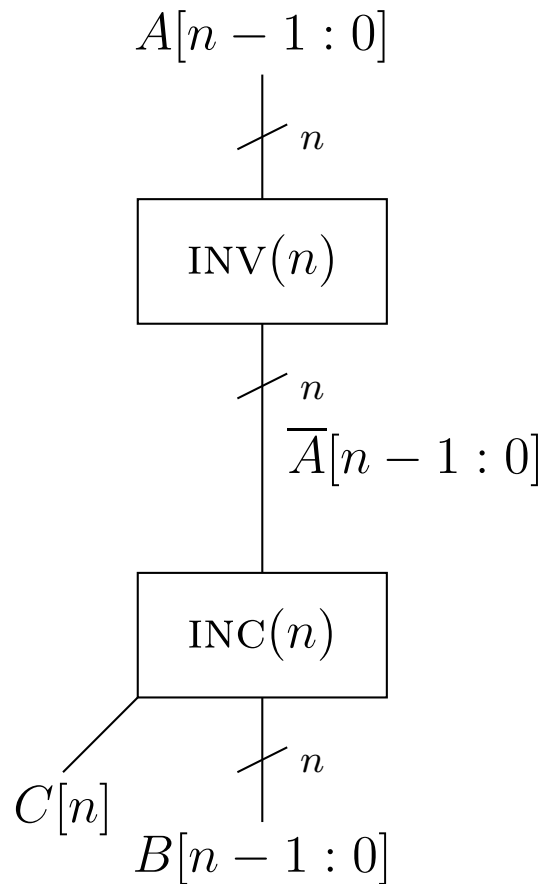
A circuit for negating a two's complement number - cont.



A circuit for negating a two's complement number - cont.

The increment circuit computes:

$$\langle \bar{A}[n-1:0] \rangle + 1.$$



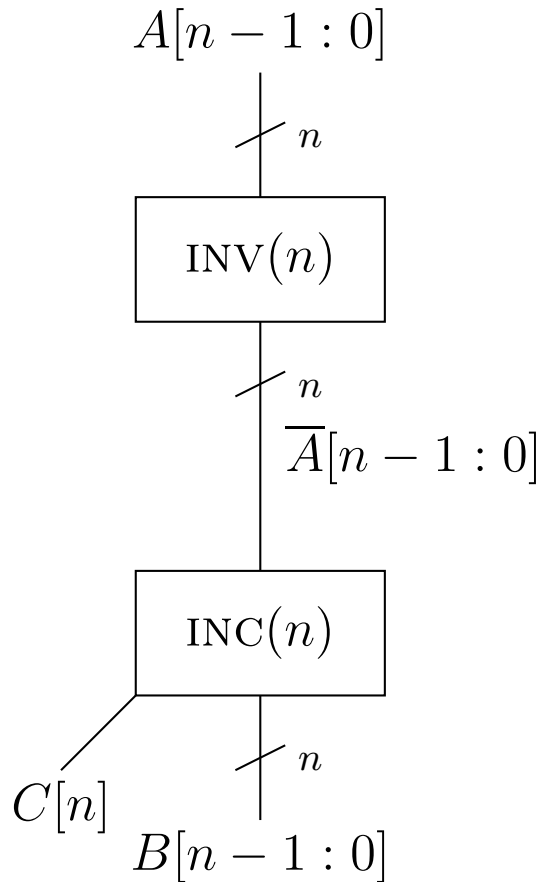
A circuit for negating a two's complement number - cont.

The increment circuit computes:

$$\langle \bar{A}[n-1:0] \rangle + 1.$$

However, we should compute

$$[\bar{A}[n-1:0]] + 1.$$



A circuit for negating a two's complement number - cont.

The increment circuit computes:

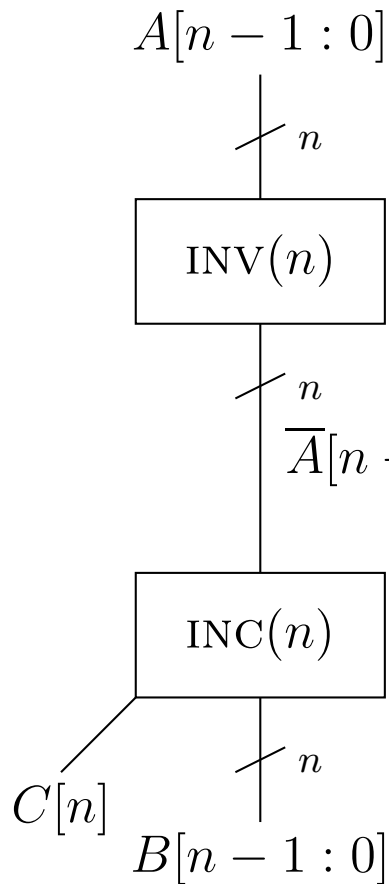
$$\langle \bar{A}[n-1:0] \rangle + 1.$$

However, we should compute

$$\langle \bar{A}[n-1:0] \rangle + 1.$$

We know that

$$\langle C[n] \cdot B[n-1:0] \rangle = \langle \bar{A}[n-1:0] \rangle + 1.$$



A circuit for negating a two's complement number - cont.

The increment circuit computes:

$$\langle \bar{A}[n - 1 : 0] \rangle + 1.$$

However, we should compute

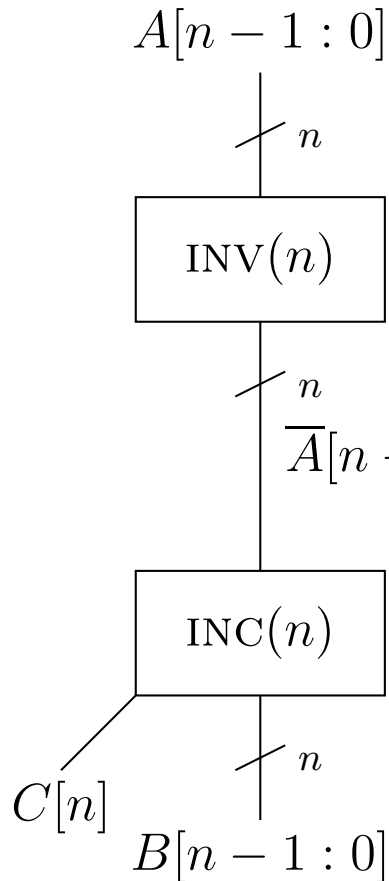
$$[\bar{A}[n - 1 : 0]] + 1.$$

We know that

$$\langle C[n] \cdot B[n - 1 : 0] \rangle = \langle \bar{A}[n - 1 : 0] \rangle + 1.$$

Suppose we are “lucky” and $C[n] = 0$.

$$\langle B[n - 1 : 0] \rangle = \langle \bar{A}[n - 1 : 0] \rangle + 1.$$



A circuit for negating a two's complement number - cont.

The increment circuit computes:

$$\langle \bar{A}[n - 1 : 0] \rangle + 1.$$

However, we should compute

$$[\bar{A}[n - 1 : 0]] + 1.$$

We know that

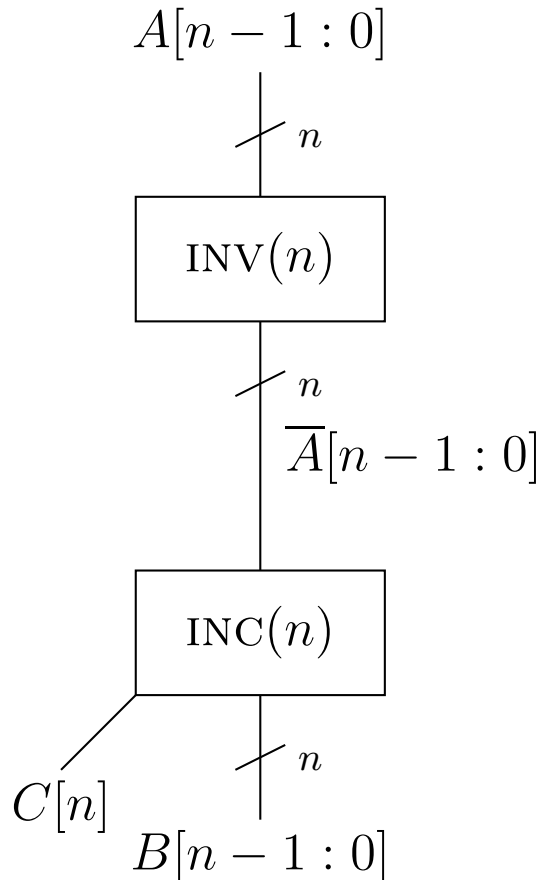
$$\langle C[n] \cdot B[n - 1 : 0] \rangle = \langle \bar{A}[n - 1 : 0] \rangle + 1.$$

Suppose we are “lucky” and $C[n] = 0$.

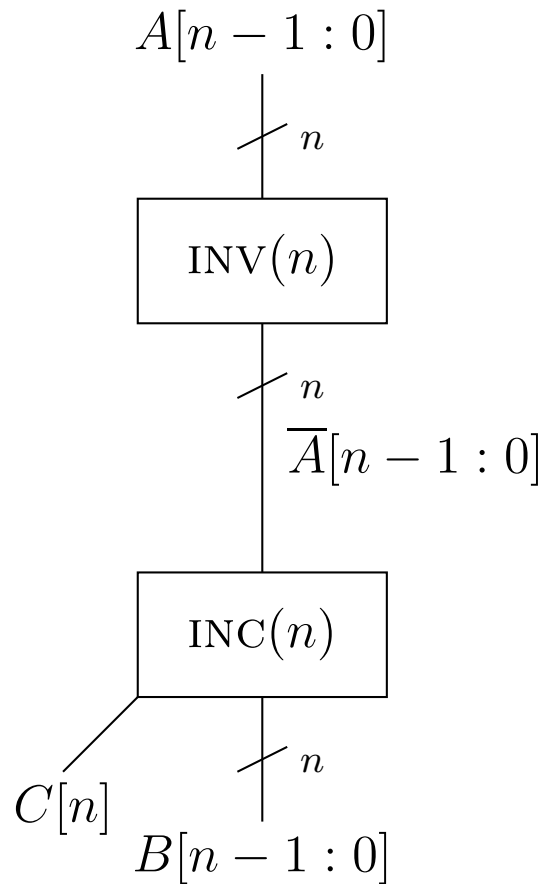
$$\langle B[n - 1 : 0] \rangle = \langle \bar{A}[n - 1 : 0] \rangle + 1.$$

Why should this imply that

$$[B[n - 1 : 0]] = [\bar{A}[n - 1 : 0]] + 1?$$



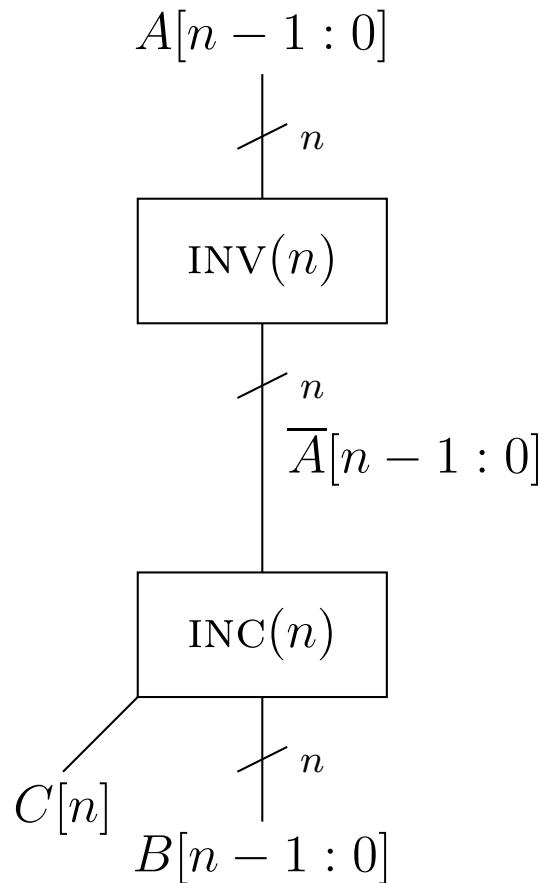
A circuit for negating a two's complement number - cont.



A circuit for negating a two's complement number - cont.

Counter example:

$$A[n - 1 : 0] = 1 \cdot 0^{n-1}.$$

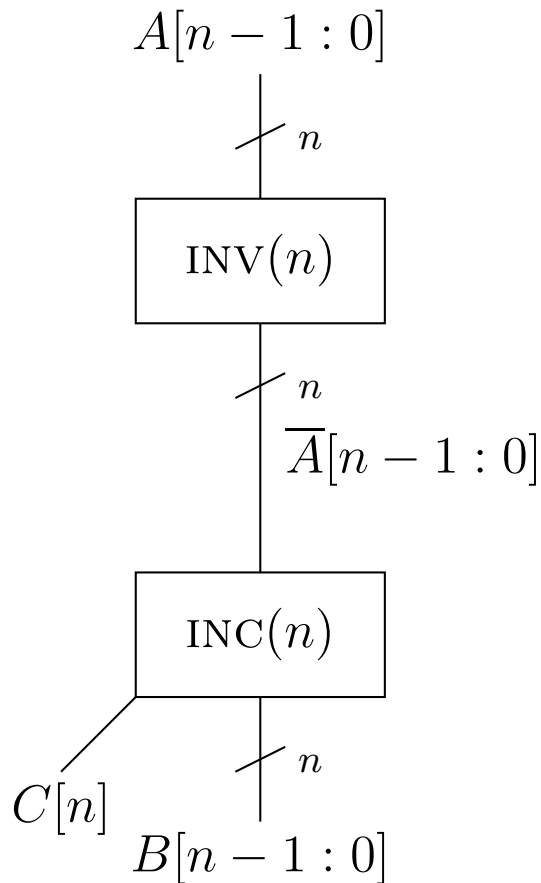


A circuit for negating a two's complement number - cont.

Counter example:

$$A[n - 1 : 0] = 1 \cdot 0^{n-1}.$$

$$\overline{A}[n - 1 : 0] = 0 \cdot 1^{n-1}.$$



A circuit for negating a two's complement number - cont.

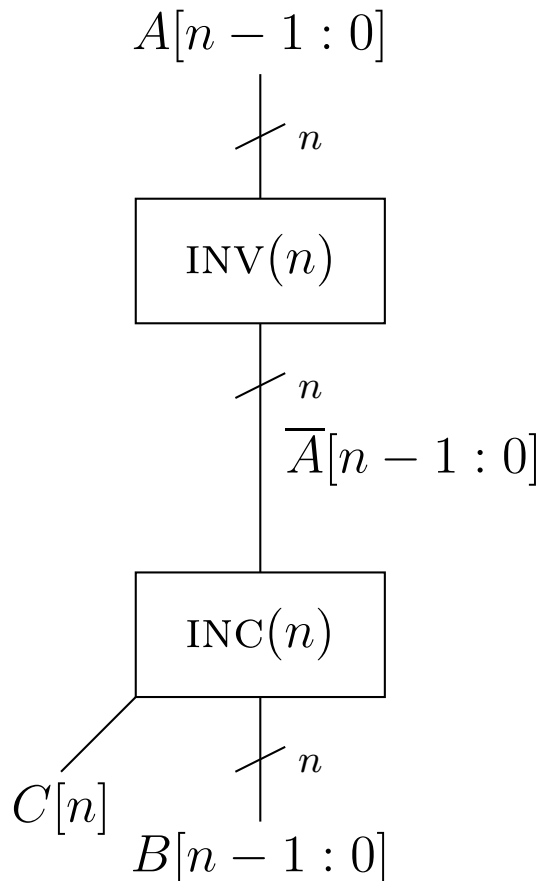
Counter example:

$$A[n - 1 : 0] = 1 \cdot 0^{n-1}.$$

$$\overline{A}[n - 1 : 0] = 0 \cdot 1^{n-1}.$$

Increment yields $C[n] = 0$ and

$$B[n - 1 : 0] = 1 \cdot 0^{n-1} = A[n - 1 : 0].$$



A circuit for negating a two's complement number - cont.

Counter example:

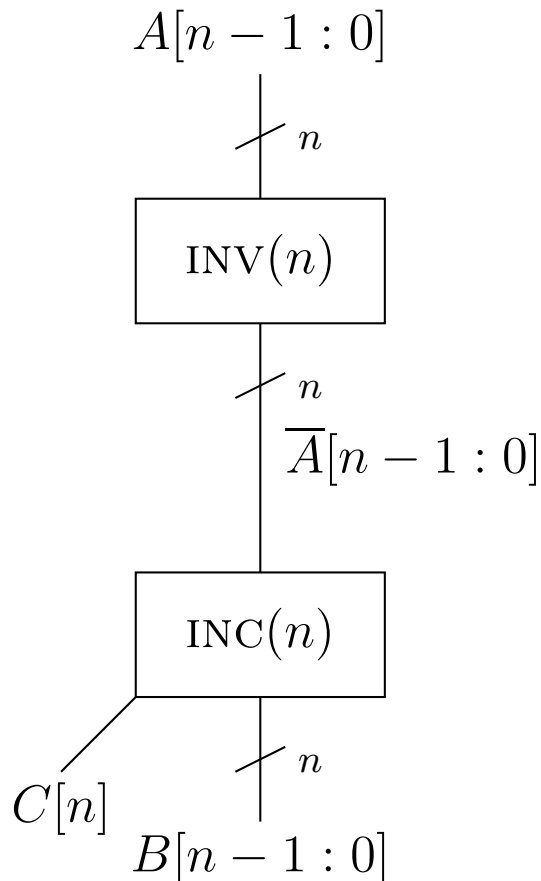
$$A[n - 1 : 0] = 1 \cdot 0^{n-1}.$$

$$\bar{A}[n - 1 : 0] = 0 \cdot 1^{n-1}.$$

Increment yields $C[n] = 0$ and

$$B[n - 1 : 0] = 1 \cdot 0^{n-1} = A[n - 1 : 0].$$

$$\Rightarrow [\vec{B}] \neq -[\vec{A}].$$



A circuit for negating a two's complement number - cont.

Counter example:

$$A[n - 1 : 0] = 1 \cdot 0^{n-1}.$$

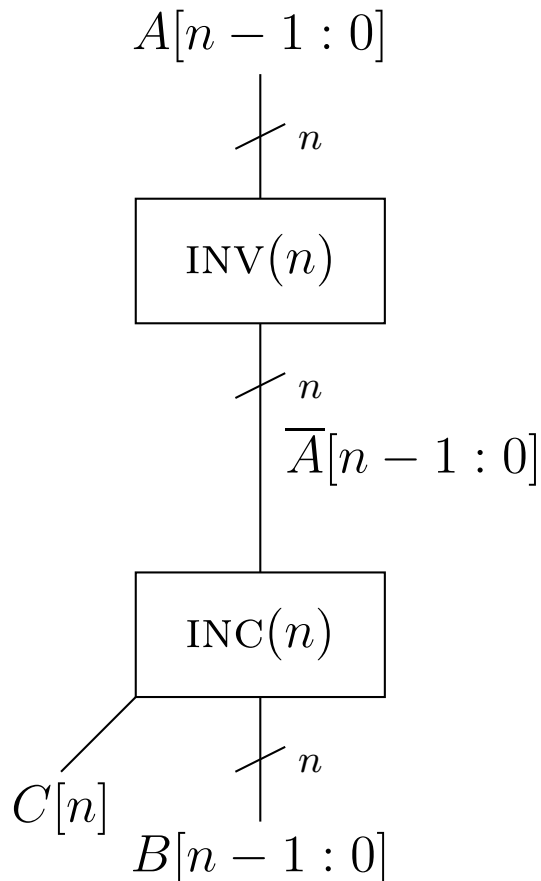
$$\bar{A}[n - 1 : 0] = 0 \cdot 1^{n-1}.$$

Increment yields $C[n] = 0$ and

$$B[n - 1 : 0] = 1 \cdot 0^{n-1} = A[n - 1 : 0].$$

$$\Rightarrow [\vec{B}] \neq -[\vec{A}].$$

Reason? binary increment is not a two's complement increment.



A circuit for negating a two's complement number - cont.

Counter example:

$$A[n - 1 : 0] = 1 \cdot 0^{n-1}.$$

$$\bar{A}[n - 1 : 0] = 0 \cdot 1^{n-1}.$$

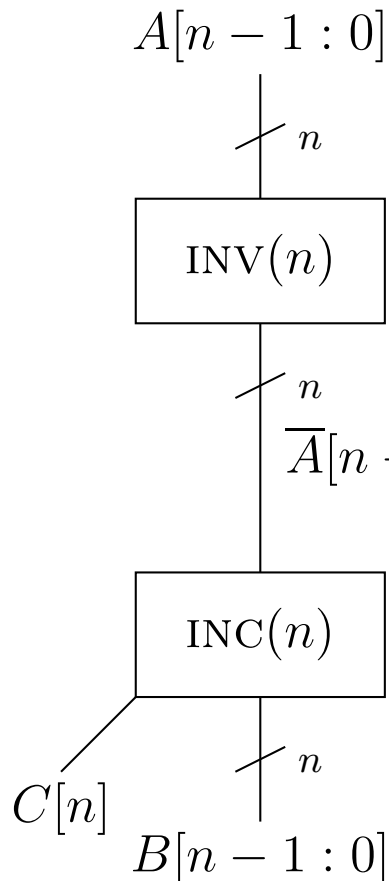
Increment yields $C[n] = 0$ and

$$B[n - 1 : 0] = 1 \cdot 0^{n-1} = A[n - 1 : 0].$$

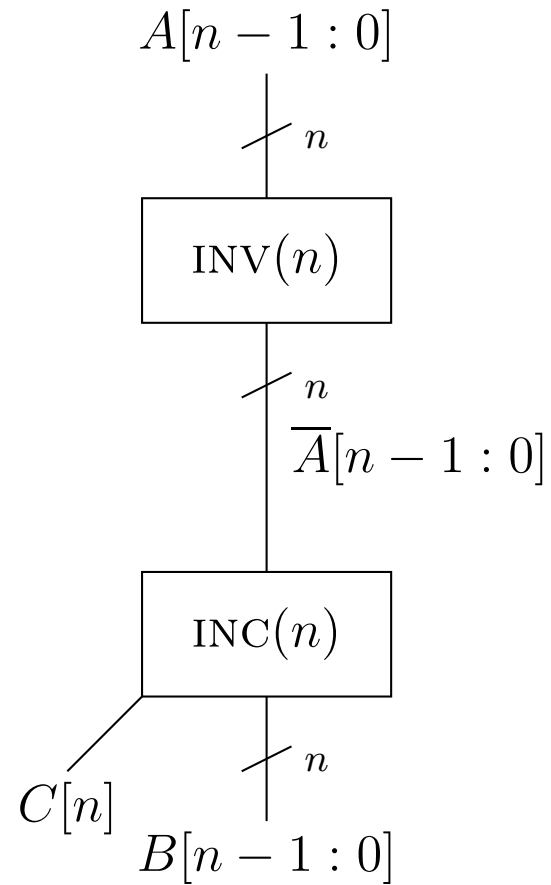
$$\Rightarrow [\vec{B}] \neq -[\vec{A}].$$

Reason? binary increment is not a two's complement increment.

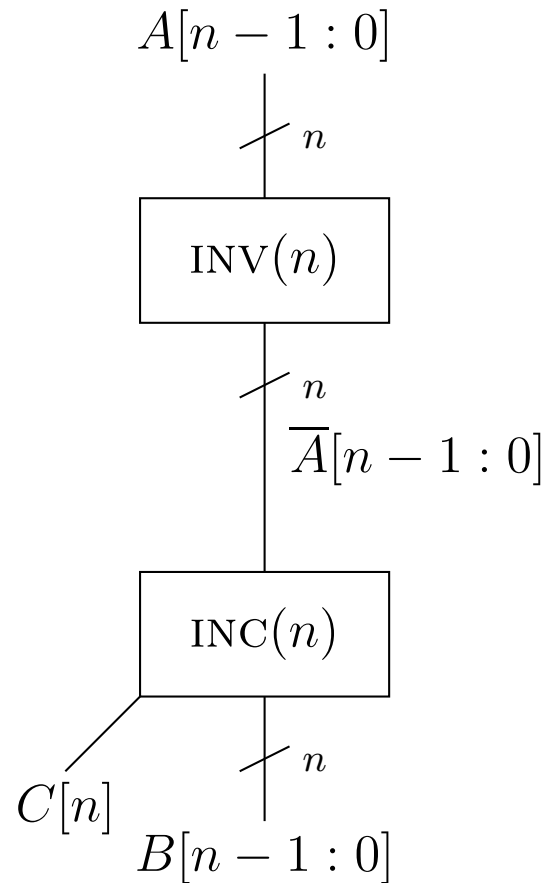
Had to err: $-\vec{A} \notin T_n.$



A circuit for negating a two's complement number - cont.



A circuit for negating a two's complement number - cont.



We will prove a theorem that will help us formulate and prove the correctness of the negation circuit.

Two's complement - mod 2^n property

Two's complement - mod 2^n property

Claim: For every $A[n - 1 : 0] \in \{0, 1\}^n$

$$\text{mod}(\langle \vec{A} \rangle, 2^n) = \text{mod}(\lceil \vec{A} \rceil, 2^n).$$

Two's complement - mod 2^n property

Claim: For every $A[n-1:0] \in \{0, 1\}^n$

$$\text{mod}(\langle \vec{A} \rangle, 2^n) = \text{mod}(\lceil \vec{A} \rceil, 2^n).$$

Note that

$$\langle \vec{A} \rangle \in [0, 2^n - 1]$$

$$\lceil \vec{A} \rceil \in [-2^{n-1}, 2^{n-1} - 1].$$

Two's complement - mod 2^n property

Claim: For every $A[n-1:0] \in \{0, 1\}^n$

$$\text{mod}(\langle \vec{A} \rangle, 2^n) = \text{mod}(\lceil \vec{A} \rceil, 2^n).$$

Note that

$$\langle \vec{A} \rangle \in [0, 2^n - 1]$$

$$\lceil \vec{A} \rceil \in [-2^{n-1}, 2^{n-1} - 1].$$

Remark: Alternative definition of two's complement representation based on Claim. Namely, represent $x \in [-2^{n-1}, 2^{n-1} - 1]$ by $x' \in [0, 2^n - 1]$, where $\text{mod}(x, 2^n) = \text{mod}(x', 2^n)$.

Claim: $\text{mod}(\langle \vec{A} \rangle, 2^n) = \text{mod}(\left[\vec{A} \right], 2^n)$

Claim: $\text{mod}(\langle \vec{A} \rangle, 2^n) = \text{mod}(\left[\vec{A} \right], 2^n)$

Proof:

$$\begin{aligned}\text{mod}(\langle \vec{A} \rangle, 2^n) &= \text{mod}(2^{n-1} \cdot A[n-1] + \langle A[n-2:0] \rangle, 2^n) \\ &= \text{mod}((2^{n-1} - 2^n) \cdot A[n-1] + \langle A[n-2:0] \rangle, 2^n) \\ &= \text{mod}(-2^{n-1} \cdot A[n-1] + \langle A[n-2:0] \rangle, 2^n) \\ &= \text{mod}(\left[\vec{A} \right], 2^n).\end{aligned}$$

□

Two's complement - sign extension

Two's complement - sign extension

Claim: If $A[n] = A[n - 1]$, then

$$[A[n : 0]] = [A[n - 1 : 0]] .$$

Two's complement - sign extension

Claim: If $A[n] = A[n - 1]$, then

$$[A[n : 0]] = [A[n - 1 : 0]].$$

Proof:

$$\begin{aligned} [A[n : 0]] &= -2^n \cdot A[n] + \langle A[n - 1 : 0] \rangle \\ &= -2^n \cdot A[n] + 2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle \\ &= -2^n \cdot A[n - 1] + 2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle \\ &= -2^{n-1} \cdot A[n - 1] + \langle A[n - 2 : 0] \rangle \\ &= [A[n - 1 : 0]]. \end{aligned}$$

QED

Two's complement - sign extension

Two's complement - sign extension

Claim: If $A[n] = A[n - 1]$, then

$$[A[n : 0]] = [A[n - 1 : 0]] .$$

Two's complement - sign extension

Claim: If $A[n] = A[n - 1]$, then

$$[A[n : 0]] = [A[n - 1 : 0]] .$$

Corollary:

$$[A[n - 1]^* \cdot A[n - 1 : 0]] = [A[n - 1 : 0]] .$$

Two's complement - sign extension

Claim: If $A[n] = A[n - 1]$, then

$$[A[n : 0]] = [A[n - 1 : 0]] .$$

Corollary:

$$[A[n - 1]^* \cdot A[n - 1 : 0]] = [A[n - 1 : 0]] .$$

sign-extension - duplicating the most significant bit does not affect the value represented in two's complement representation. This is similar to padding zeros from the left in binary representation.

Theorem: signed addition \mapsto binary addition

Theorem: signed addition \mapsto binary addition

- Binary addition: assume that

$$\langle C[n] \cdot S[n - 1 : 0] \rangle = \langle A[n - 1 : 0] \rangle + \langle B[n - 1 : 0] \rangle + C[0].$$

Theorem: signed addition \mapsto binary addition

- Binary addition: assume that

$$\langle C[n] \cdot S[n - 1 : 0] \rangle = \langle A[n - 1 : 0] \rangle + \langle B[n - 1 : 0] \rangle + C[0].$$

- $C[n - 1]$ - carry-bit in position $[n - 1]$ associated with this binary addition.

Theorem: signed addition \mapsto binary addition

- Binary addition: assume that

$$\langle C[n] \cdot S[n - 1 : 0] \rangle = \langle A[n - 1 : 0] \rangle + \langle B[n - 1 : 0] \rangle + C[0].$$

- $C[n - 1]$ - carry-bit in position $[n - 1]$ associated with this binary addition.



$$z \triangleq [A[n - 1 : 0]] + [B[n - 1 : 0]] + C[0].$$

Theorem: signed addition \mapsto binary addition

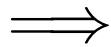
- Binary addition: assume that

$$\langle C[n] \cdot S[n - 1 : 0] \rangle = \langle A[n - 1 : 0] \rangle + \langle B[n - 1 : 0] \rangle + C[0].$$

- $C[n - 1]$ - carry-bit in position $[n - 1]$ associated with this binary addition.



$$z \triangleq [A[n - 1 : 0]] + [B[n - 1 : 0]] + C[0].$$



$$C[n - 1] - C[n] = 1 \quad \implies \quad z > 2^{n-1} - 1$$

$$C[n] - C[n - 1] = 1 \quad \implies \quad z < -2^{n-1}$$

$$z \in T_n \quad \iff \quad C[n] = C[n - 1]$$

$$z \in T_n \quad \implies \quad z = [S[n - 1 : 0]].$$

Theorem - proof

Theorem - proof

functionality of FA_{n-1} in $\text{RCA}(n) \implies$

$$A[n-1] + B[n-1] + C[n-1] = 2C[n] + S[n-1]$$

$$\implies A[n-1] + B[n-1] = 2C[n] - C[n-1] + S[n-1].$$

Theorem - proof

functionality of FA_{n-1} in $\text{RCA}(n) \implies$

$$\begin{aligned} A[n-1] + B[n-1] + C[n-1] &= 2C[n] + S[n-1] \\ \implies A[n-1] + B[n-1] &= 2C[n] - C[n-1] + S[n-1]. \end{aligned}$$

We now expand z as follows:

$$\begin{aligned} z &= [A[n-1 : 0]] + [B[n-1 : 0]] + C[0] \\ &= -2^{n-1} \cdot (A[n-1] + B[n-1]) \\ &\quad + \langle A[n-2 : 0] \rangle + \langle B[n-2 : 0] \rangle + C[0] \\ &= -2^{n-1} \cdot (2C[n] - C[n-1] + S[n-1]) + \langle C[n-1] \cdot S[n-2 : 0] \rangle \\ &= -2^{n-1} \cdot (2C[n] - C[n-1] - C[n-1]) + [S[n-1] \cdot S[n-2 : 0]] \\ &= -2^n \cdot (C[n] - C[n-1]) + [S[n-1 : 0]]. \end{aligned}$$

Theorem - proof - cont

$$z = -2^n \cdot (C[n] - C[n - 1]) + [S[n - 1 : 0]] .$$

Theorem - proof - cont

$$z = -2^n \cdot (C[n] - C[n - 1]) + [S[n - 1 : 0]] .$$

We distinguish between three cases:

Theorem - proof - cont

$$z = -2^n \cdot (C[n] - C[n - 1]) + [S[n - 1 : 0]].$$

We distinguish between three cases:

1. If $C[n] - C[n - 1] = 1$, then

$$\begin{aligned} z &= -2^n + [S[n - 1 : 0]] \\ &\leq -2^n + 2^{n-1} - 1 = -2^{n-1} - 1. \end{aligned}$$

Theorem - proof - cont

$$z = -2^n \cdot (C[n] - C[n - 1]) + [S[n - 1 : 0]].$$

We distinguish between three cases:

1. If $C[n] - C[n - 1] = 1$, then

$$\begin{aligned} z &= -2^n + [S[n - 1 : 0]] \\ &\leq -2^n + 2^{n-1} - 1 = -2^{n-1} - 1. \end{aligned}$$

2. If $C[n] - C[n - 1] = -1$, then

$$\begin{aligned} z &= 2^n + [S[n - 1 : 0]] \\ &\geq 2^n - 2^{n-1} = 2^{n-1}. \end{aligned}$$

Theorem - proof - cont

$$z = -2^n \cdot (C[n] - C[n - 1]) + [S[n - 1 : 0]].$$

We distinguish between three cases:

1. If $C[n] - C[n - 1] = 1$, then

$$\begin{aligned} z &= -2^n + [S[n - 1 : 0]] \\ &\leq -2^n + 2^{n-1} - 1 = -2^{n-1} - 1. \end{aligned}$$

2. If $C[n] - C[n - 1] = -1$, then

$$\begin{aligned} z &= 2^n + [S[n - 1 : 0]] \\ &\geq 2^n - 2^{n-1} = 2^{n-1}. \end{aligned}$$

3. If $C[n] = C[n - 1]$, then $z = [S[n - 1 : 0]]$, and obviously $z \in T_n$.

QED

Overflow

Overflow

DEF: Let $z \triangleq [A[n-1:0]] + [B[n-1:0]] + C[0]$. The signal **OVF** is defined as follows:

$$\text{OVF} \triangleq \begin{cases} 1 & \text{if } z \notin T_n \\ 0 & \text{otherwise.} \end{cases}$$

Overflow

DEF: Let $z \triangleq [A[n-1:0]] + [B[n-1:0]] + C[0]$. The signal **OVF** is defined as follows:

$$\text{OVF} \triangleq \begin{cases} 1 & \text{if } z \notin T_n \\ 0 & \text{otherwise.} \end{cases}$$

- overflow - sum is either too large or too small.

Overflow

DEF: Let $z \triangleq [A[n-1:0]] + [B[n-1:0]] + C[0]$. The signal **OVF** is defined as follows:

$$\text{OVF} \triangleq \begin{cases} 1 & \text{if } z \notin T_n \\ 0 & \text{otherwise.} \end{cases}$$

- overflow - sum is either too large or too small.
- better term - **out-of-range** - not the common term.

Overflow

DEF: Let $z \triangleq [A[n-1:0]] + [B[n-1:0]] + C[0]$. The signal **OVF** is defined as follows:

$$\text{OVF} \triangleq \begin{cases} 1 & \text{if } z \notin T_n \\ 0 & \text{otherwise.} \end{cases}$$

- overflow - sum is either too large or too small.
- better term - **out-of-range** - not the common term.
- By Theorem

$$\text{OVF} = \text{XOR}(C[n-1], C[n]).$$

Detecting Overflow

- The signal $C[n - 1]$ may not be available if one uses a “black-box” binary-adder (e.g., a library component in which $C[n - 1]$ is an internal signal).

Detecting Overflow

- The signal $C[n - 1]$ may not be available if one uses a “black-box” binary-adder (e.g., a library component in which $C[n - 1]$ is an internal signal).
- In this case we detect overflow based on the following claim.

Detecting Overflow

- The signal $C[n - 1]$ may not be available if one uses a “black-box” binary-adder (e.g., a library component in which $C[n - 1]$ is an internal signal).
- In this case we detect overflow based on the following claim.

Claim:

$$\text{XOR}(C[n - 1], C[n]) = \text{XOR}_4(A[n - 1], B[n - 1], S[n - 1], C[n]).$$

Detecting Overflow

- The signal $C[n - 1]$ may not be available if one uses a “black-box” binary-adder (e.g., a library component in which $C[n - 1]$ is an internal signal).
- In this case we detect overflow based on the following claim.

Claim:

$$\text{XOR}(C[n - 1], C[n]) = \text{XOR}_4(A[n - 1], B[n - 1], S[n - 1], C[n]).$$

Proof: Recall that

$$C[n - 1] = \text{XOR}_3(A[n - 1], B[n - 1], S[n - 1]).$$

□

Determining the sign of the sum

- How do we determine the sign of the sum z ?

Determining the sign of the sum

- How do we determine the sign of the sum z ?
- Obviously, if $z \in T_n$, then the sign-bit $S[n - 1]$ indicates whether z is negative.

Determining the sign of the sum

- How do we determine the sign of the sum z ?
- Obviously, if $z \in T_n$, then the sign-bit $S[n - 1]$ indicates whether z is negative.
- What happens if overflow occurs?

Determining the sign of the sum

- How do we determine the sign of the sum z ?
- Obviously, if $z \in T_n$, then the sign-bit $S[n - 1]$ indicates whether z is negative.
- What happens if overflow occurs?

Question: Provide an example in which the sign of z is not signaled correctly by $S[n - 1]$.

Determining the sign of the sum

- How do we determine the sign of the sum z ?
- Obviously, if $z \in T_n$, then the sign-bit $S[n - 1]$ indicates whether z is negative.
- What happens if overflow occurs?

Question: Provide an example in which the sign of z is not signaled correctly by $S[n - 1]$.

We would like to be able to know whether z is negative regardless of whether overflow occurs.

Determining the sign of the sum - cont.

Determining the sign of the sum - cont.

DEF: The signal **NEG** is defined as follows:

$$\text{NEG} \triangleq \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{if } z \geq 0. \end{cases}$$

Determining the sign of the sum - cont.

DEF: The signal **NEG** is defined as follows:

$$\text{NEG} \triangleq \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{if } z \geq 0. \end{cases}$$

Theorem implies that:

$$\text{NEG} = \begin{cases} S[n-1] & \text{if no overflow} \\ 1 & \text{if } C[n] - C[n-1] = 1 \\ 0 & \text{if } C[n-1] - C[n] = 1. \end{cases}$$

Determining the sign of the sum - cont.

DEF: The signal **NEG** is defined as follows:

$$\text{NEG} \triangleq \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{if } z \geq 0. \end{cases}$$

Theorem implies that:

$$\text{NEG} = \begin{cases} S[n-1] & \text{if no overflow} \\ 1 & \text{if } C[n] - C[n-1] = 1 \\ 0 & \text{if } C[n-1] - C[n] = 1. \end{cases}$$

An even simpler method...

Claim: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$.

Claim: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$.

Proof:

The proof is based on playing the following “mental game”:

Claim: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$.

Proof:

The proof is based on playing the following “mental game”:

- “extend” the computation to $n + 1$ bits.

Claim: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$.

Proof:

The proof is based on playing the following “mental game”:

- “extend” the computation to $n + 1$ bits.
- \implies overflow does not occur in extended precision.

Claim: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$.

Proof:

The proof is based on playing the following “mental game”:

- “extend” the computation to $n + 1$ bits.
- \implies overflow does not occur in extended precision.
- \implies the sum bit in position n indicates correctly the sign of the sum z .

Claim: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$.

Proof:

The proof is based on playing the following “mental game”:

- “extend” the computation to $n + 1$ bits.
- \implies overflow does not occur in extended precision.
- \implies the sum bit in position n indicates correctly the sign of the sum z .
- express this sum bit using n -bit addition signals.

Proof: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$ - **cont.**

Proof: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$ - **cont.**

Sign extension to $n+1$ bits:

$$\tilde{A}[n:0] \triangleq A[n-1] \cdot A[n-1:0]$$

$$\tilde{B}[n:0] \triangleq B[n-1] \cdot B[n-1:0]$$

$$\langle \tilde{C}[n+1] \cdot \tilde{S}[n:0] \rangle \triangleq \langle \tilde{A}[n:0] \rangle + \langle \tilde{B}[n:0] \rangle + C[0].$$

Proof: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$ - **cont.**

Sign extension to $n+1$ bits:

$$\tilde{A}[n:0] \triangleq A[n-1] \cdot A[n-1:0]$$

$$\tilde{B}[n:0] \triangleq B[n-1] \cdot B[n-1:0]$$

$$\langle \tilde{C}[n+1] \cdot \tilde{S}[n:0] \rangle \triangleq \langle \tilde{A}[n:0] \rangle + \langle \tilde{B}[n:0] \rangle + C[0].$$

Since sign-extension preserves value, it follows that

$$z = \left[\tilde{A}[n:0] \right] + \left[\tilde{B}[n:0] \right] + C[0].$$

Proof: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$ - **cont.**

Proof: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$ - **cont.**

We claim that $z \in T_{n+1}$. This follows from

$$\begin{aligned} z &= [A[n-1 : 0]] + [B[n-1 : 0]] + C[0] \\ &\leq 2^{n-1} - 1 + 2^{n-1} - 1 + 1 \\ &\leq 2^n - 1. \end{aligned}$$

Proof: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$ - **cont.**

We claim that $z \in T_{n+1}$. This follows from

$$\begin{aligned} z &= [A[n-1:0]] + [B[n-1:0]] + C[0] \\ &\leq 2^{n-1} - 1 + 2^{n-1} - 1 + 1 \\ &\leq 2^n - 1. \end{aligned}$$

Similarly $z \geq 2^{-n}$.

Proof: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$ - **cont.**

We claim that $z \in T_{n+1}$. This follows from

$$\begin{aligned} z &= [A[n-1:0]] + [B[n-1:0]] + C[0] \\ &\leq 2^{n-1} - 1 + 2^{n-1} - 1 + 1 \\ &\leq 2^n - 1. \end{aligned}$$

Similarly $z \geq 2^{-n}$.

Since sign-extension preserves value and $z \in T_{n+1}$:

$$z \stackrel{\text{sign-ext}}{=} [\tilde{A}[n:0]] + [\tilde{B}[n:0]] + C[0] \stackrel{\text{no OVF}}{=} [\tilde{S}[n:0]].$$

Proof: $\text{NEG} = \text{XOR}_3(A[n-1], B[n-1], C[n])$ - **cont.**

We claim that $z \in T_{n+1}$. This follows from

$$\begin{aligned} z &= [A[n-1:0]] + [B[n-1:0]] + C[0] \\ &\leq 2^{n-1} - 1 + 2^{n-1} - 1 + 1 \\ &\leq 2^n - 1. \end{aligned}$$

Similarly $z \geq 2^{-n}$.

Since sign-extension preserves value and $z \in T_{n+1}$:

$$z \stackrel{\text{sign-ext}}{=} [\tilde{A}[n:0]] + [\tilde{B}[n:0]] + C[0] \stackrel{\text{no OVF}}{=} [\tilde{S}[n:0]].$$

$$\implies \text{NEG} = \tilde{S}[n].$$

Proof: $\text{NEG} = \text{XOR}_3(A[n - 1], B[n - 1], C[n])$ - **cont.**

$$\begin{aligned}\text{NEG} &= \tilde{S}[n] \\ &= \text{XOR}_3(\tilde{A}[n], \tilde{B}[n], \tilde{C}[n]) \\ &= \text{XOR}_3(A[n - 1], B[n - 1], C[n]).\end{aligned}$$

QED

More on NEG

Question: Prove that $\text{NEG} = \text{XOR}(\text{OVF}, S[n - 1])$.

A two's-complement adder - S-ADDER(n)

A two's-complement adder - S-ADDER(n)

DEF:

Input: $A[n - 1 : 0], B[n - 1 : 0] \in \{0, 1\}^n$, and $C[0] \in \{0, 1\}$.

Output: $S[n - 1 : 0] \in \{0, 1\}^n$ and $\text{NEG}, \text{OVF} \in \{0, 1\}$.

Functionality: Define z as follows:

$$z \triangleq [A[n - 1 : 0]] + [B[n - 1 : 0]] + C[0].$$

The functionality is defined as follows:

$$z \in T_n \implies [S[n - 1 : 0]] = z$$

$$z \in T_n \iff \text{OVF} = 0$$

$$z < 0 \iff \text{NEG} = 1.$$

A two's-complement adder - S-ADDER(n)

DEF:

Input: $A[n - 1 : 0], B[n - 1 : 0] \in \{0, 1\}^n$, and $C[0] \in \{0, 1\}$.

Output: $S[n - 1 : 0] \in \{0, 1\}^n$ and $\text{NEG}, \text{OVF} \in \{0, 1\}$.

Functionality: Define z as follows:

$$z \triangleq [A[n - 1 : 0]] + [B[n - 1 : 0]] + C[0].$$

The functionality is defined as follows:

$$z \in T_n \implies [S[n - 1 : 0]] = z$$

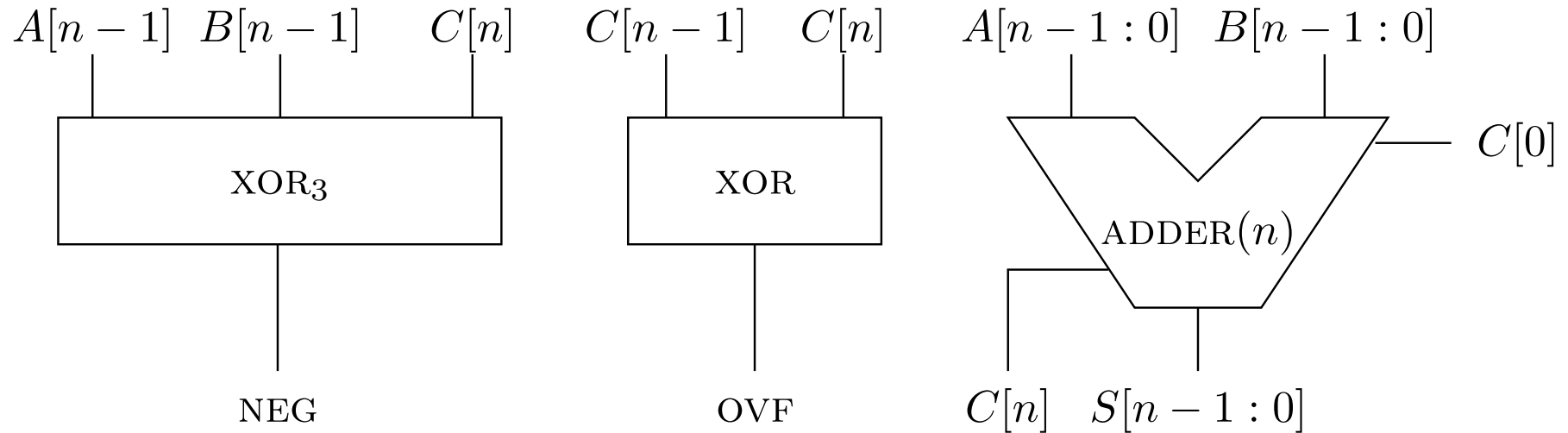
$$z \in T_n \iff \text{OVF} = 0$$

$$z < 0 \iff \text{NEG} = 1.$$

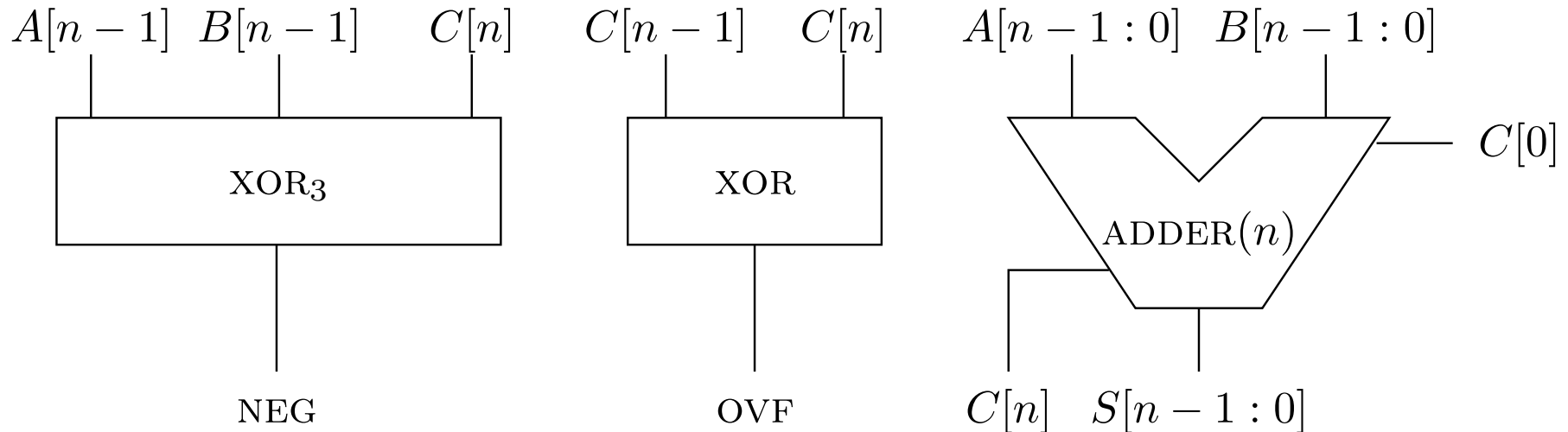
■ Note that no carry-out $C[n]$ is output.

S-ADDER(n) - **implementation**

S-ADDER(n) - implementation

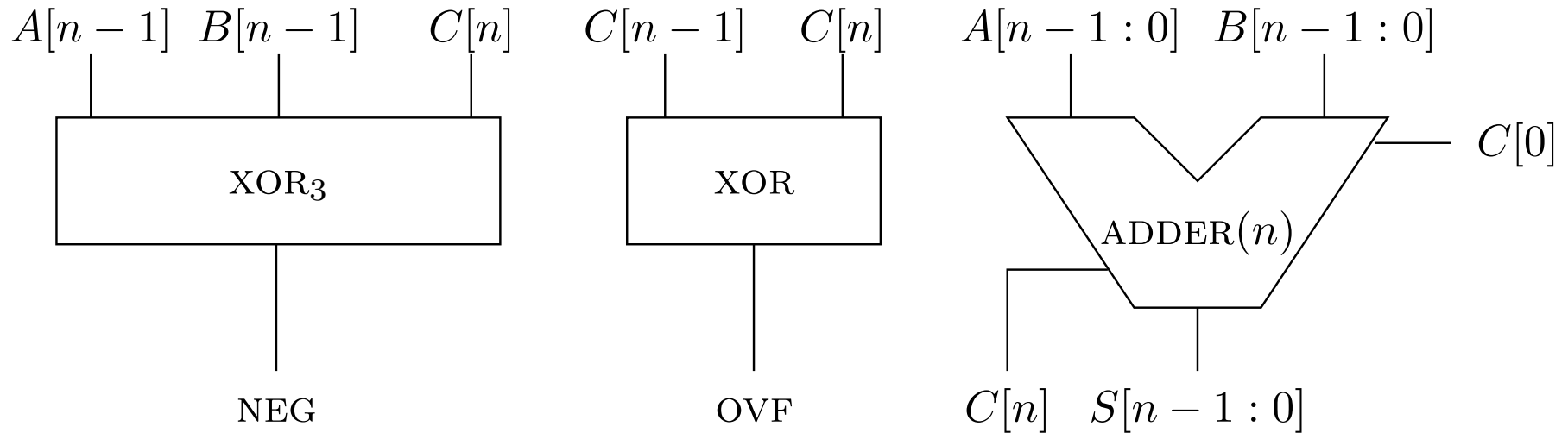


S-ADDER(n) - implementation



- a two's complement adder is identical to a binary adder except for the circuitry that computes the flags OVF and NEG .
- in an arithmetic logic unit (ALU), the same circuit is used for signed addition and unsigned addition.

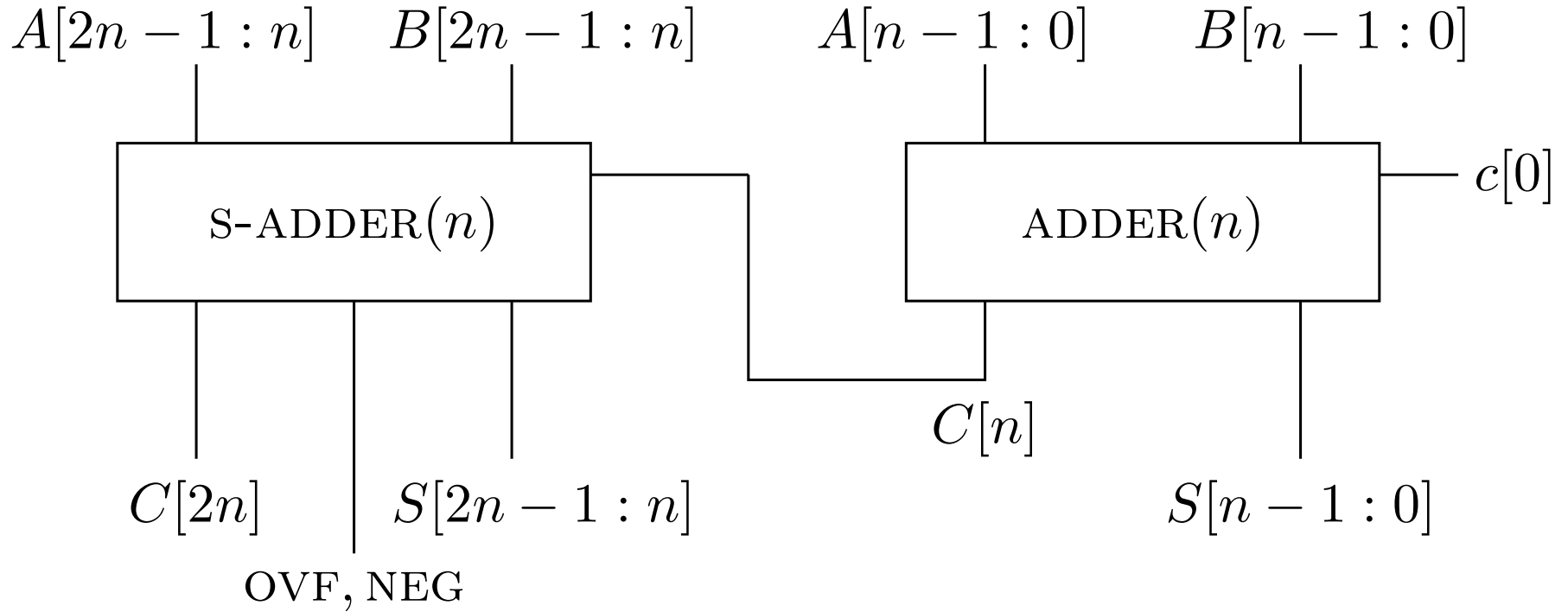
S-ADDER(n) - correctness



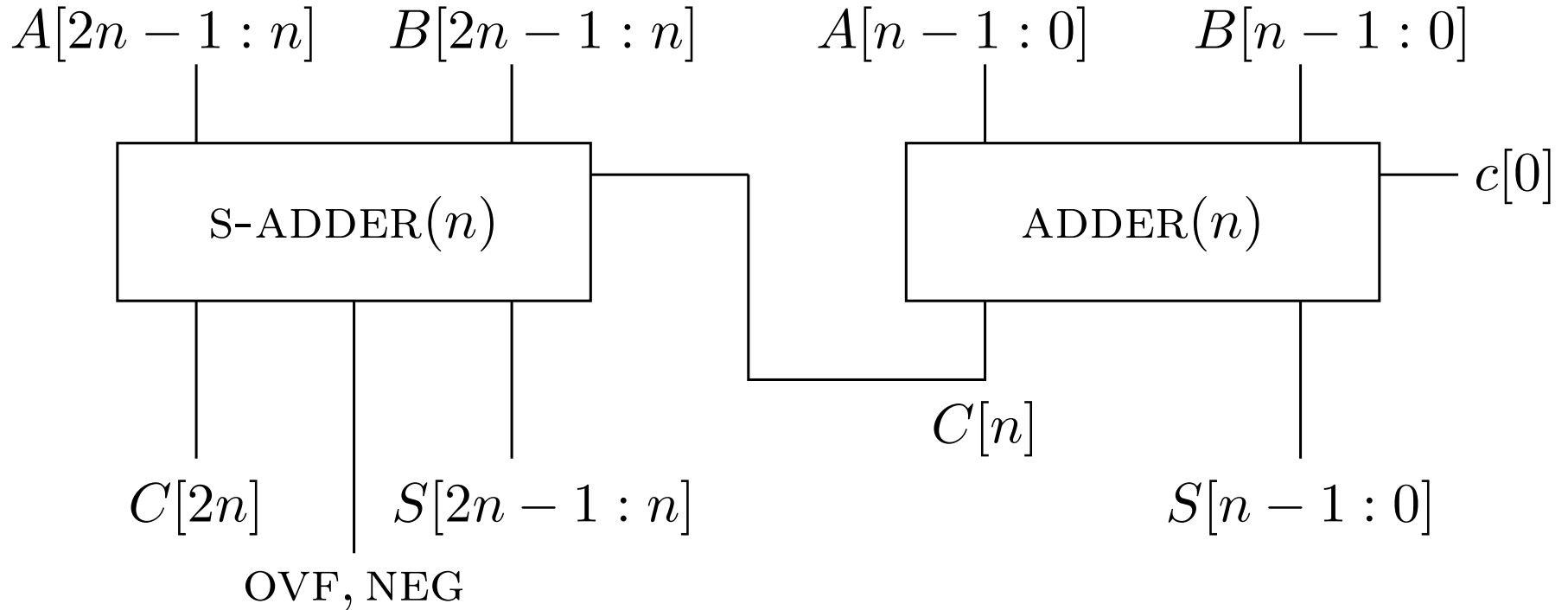
Question: Prove that this design is correct.

Concatenating adders

Concatenating adders

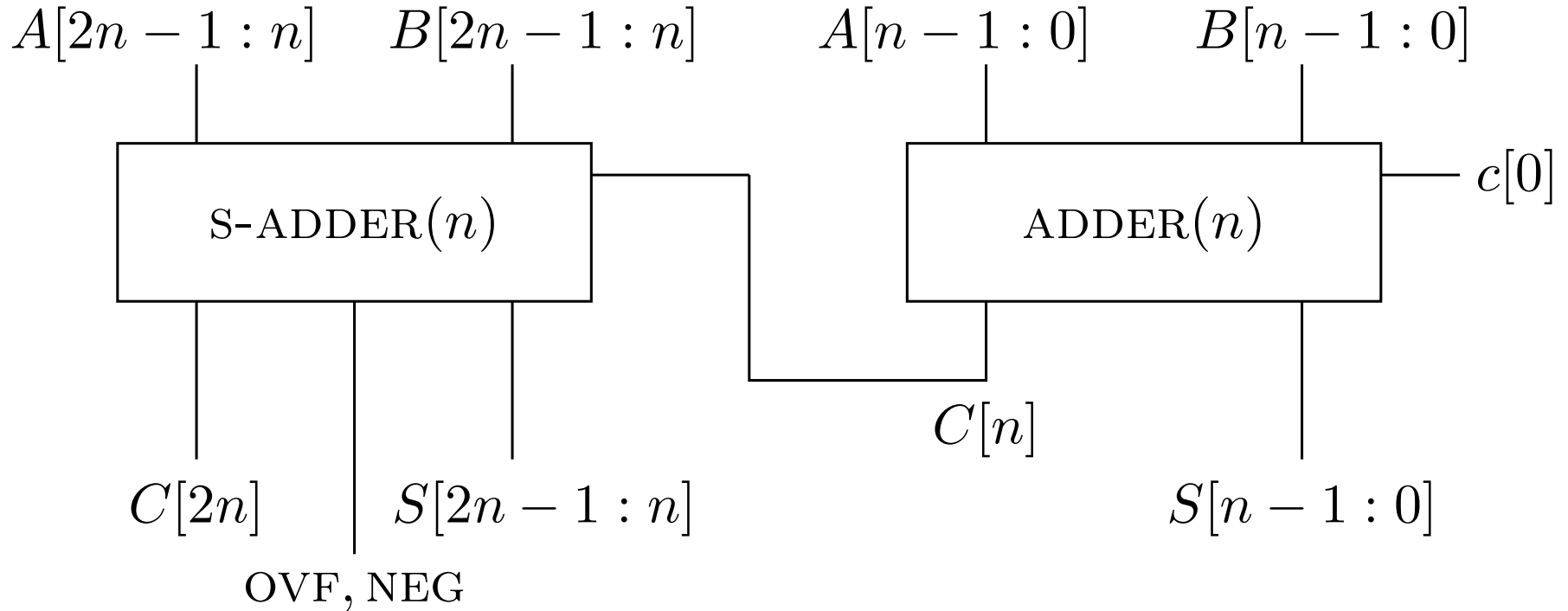


Concatenating adders



Question: Is this a correct s-ADDER($2n$)?

Concatenating adders



Question: Is this a correct **S-ADDER($2n$)**?

Question: How about a partition k and $2n - k$?

two's-complement adder/subtractor - ADD-SUB(n)

two's-complement adder/subtractor - ADD-SUB(n)

DEF:

Input: $A[n - 1 : 0], B[n - 1 : 0] \in \{0, 1\}^n$, and $sub \in \{0, 1\}$.

Output: $S[n - 1 : 0] \in \{0, 1\}^n$ and $NEG, OVF \in \{0, 1\}$.

Functionality: Define z as follows:

$$z \triangleq [A[n - 1 : 0]] + (-1)^{sub} \cdot [B[n - 1 : 0]].$$

The functionality is defined as follows:

$$z \in T_n \implies [S[n - 1 : 0]] = z$$

$$z \in T_n \iff OVF = 0$$

$$z < 0 \iff NEG = 1.$$

two's-complement adder/subtractor - ADD-SUB(n)

DEF:

Input: $A[n - 1 : 0], B[n - 1 : 0] \in \{0, 1\}^n$, and $sub \in \{0, 1\}$.

Output: $S[n - 1 : 0] \in \{0, 1\}^n$ and $NEG, OVF \in \{0, 1\}$.

Functionality: Define z as follows:

$$z \triangleq [A[n - 1 : 0]] + (-1)^{sub} \cdot [B[n - 1 : 0]].$$

The functionality is defined as follows:

$$z \in T_n \implies [S[n - 1 : 0]] = z$$

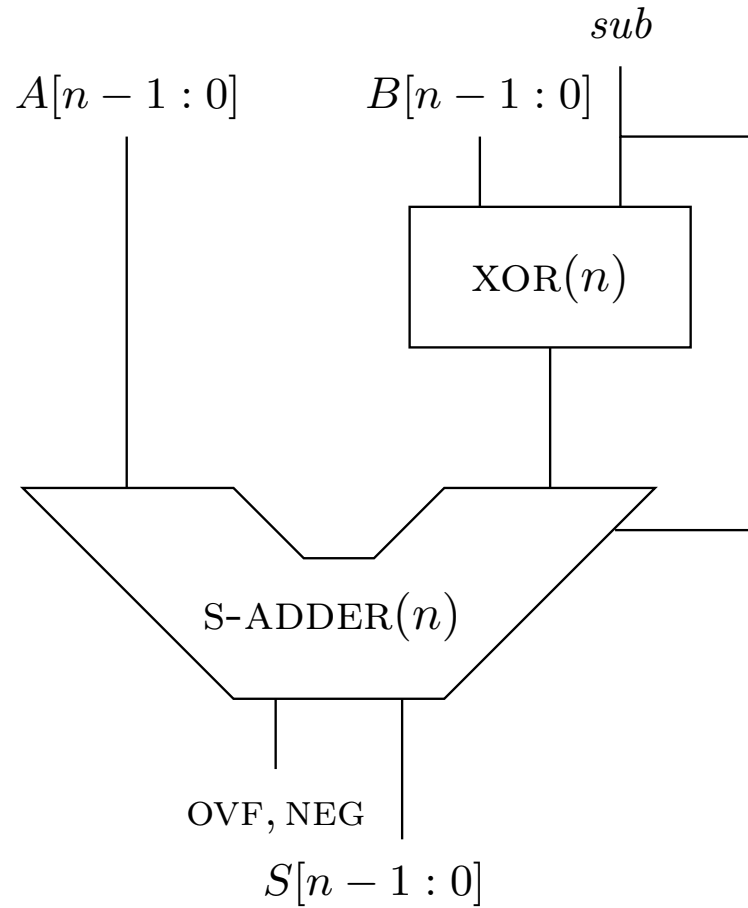
$$z \in T_n \iff OVF = 0$$

$$z < 0 \iff NEG = 1.$$

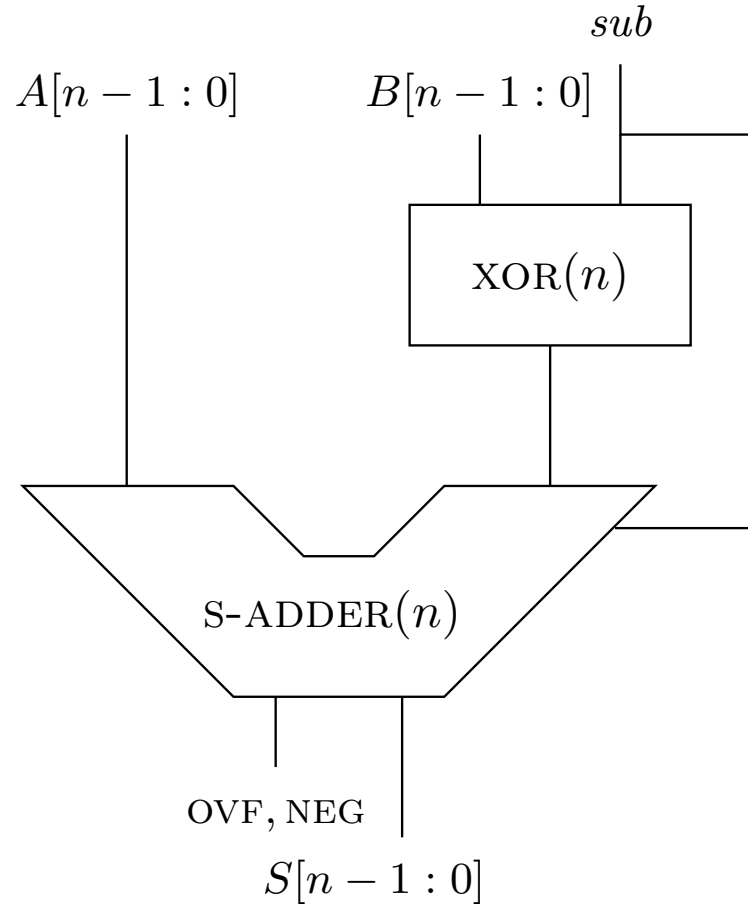
- sub - indicates if the operation is addition or subtraction.
- no carry-in bit $C[0]$ is input & no carry-out $C[n]$ is output.

ADD-SUB(n) - **implementation**

ADD-SUB(n) - implementation

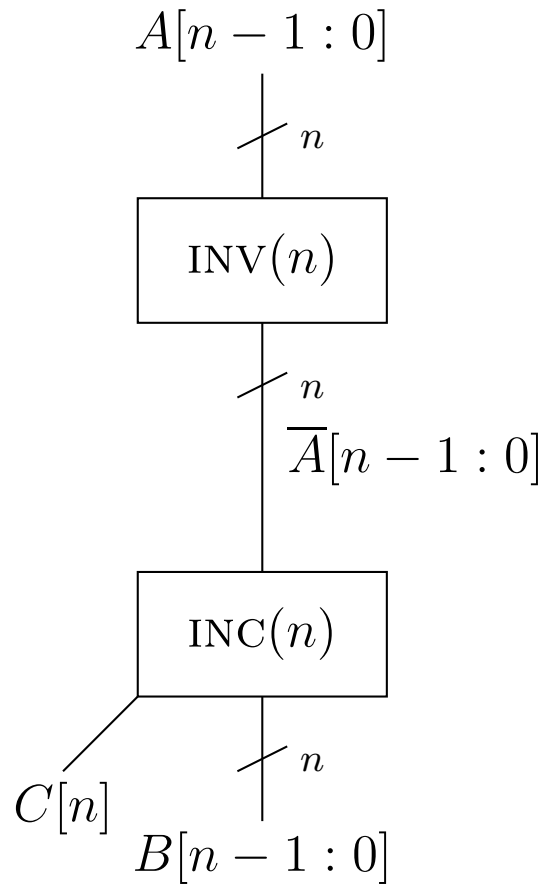


ADD-SUB(n) - implementation



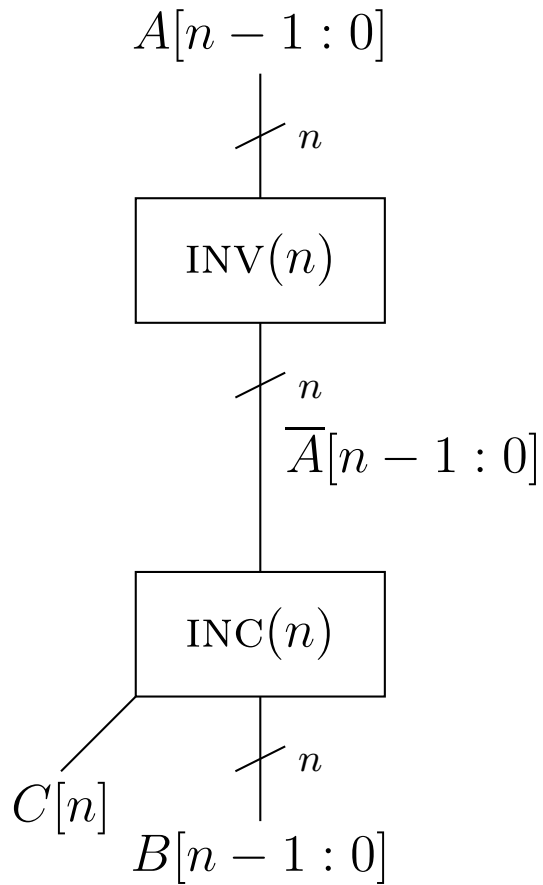
Question: Is this implementation correct?

back to the negation circuit



back to the negation circuit

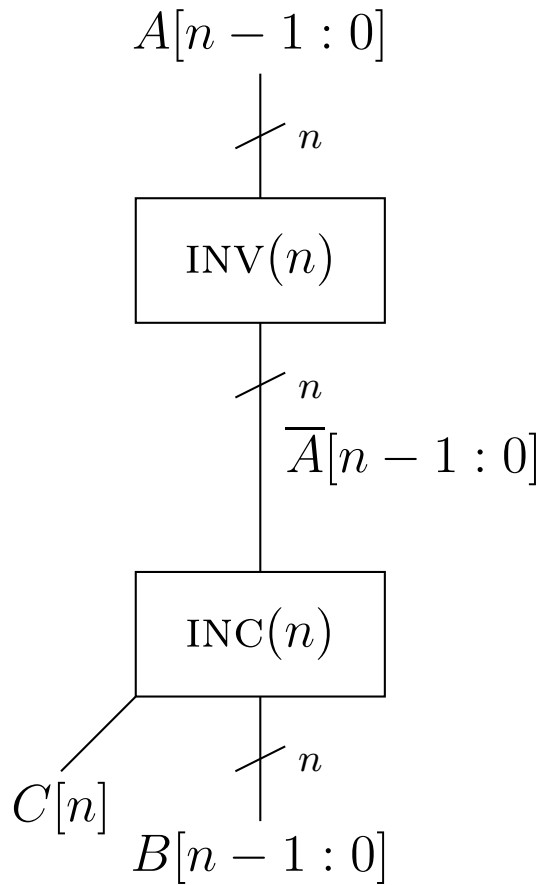
Question:



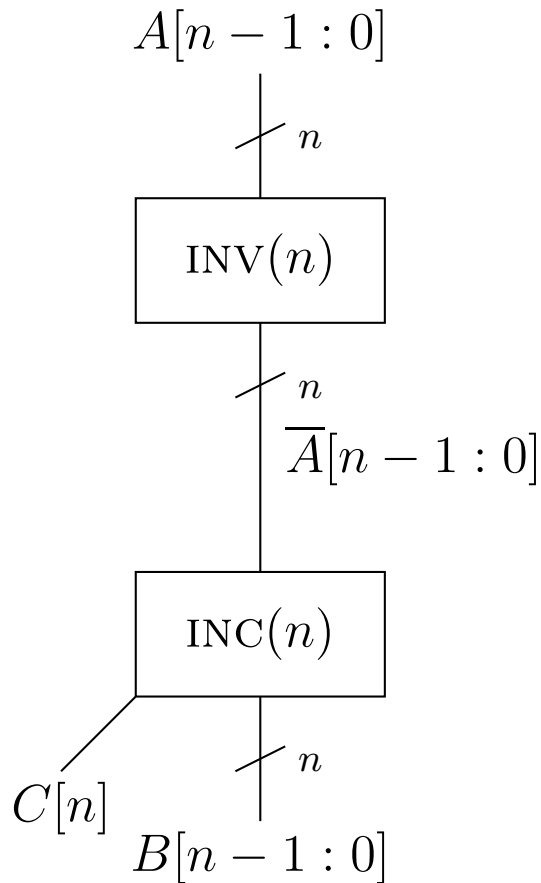
back to the negation circuit

Question:

1. When is the circuit correct?



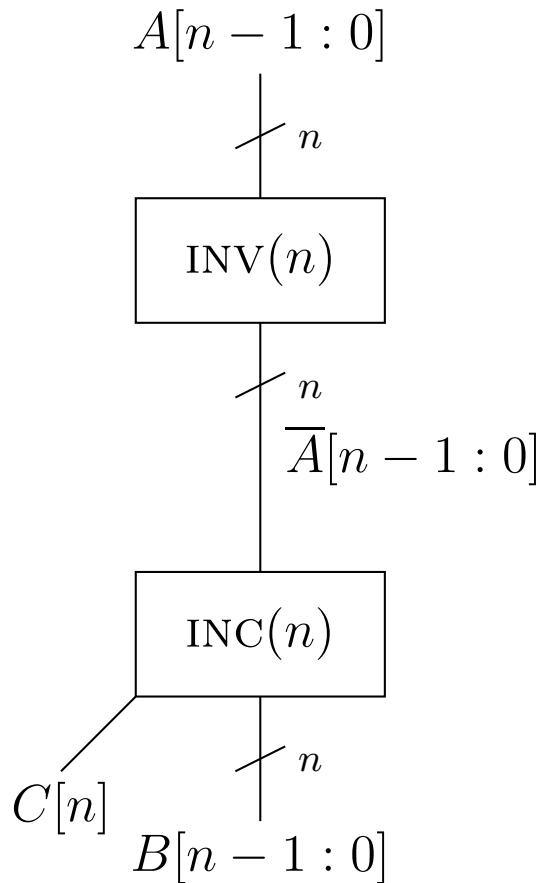
back to the negation circuit



Question:

1. When is the circuit correct?
2. Suppose we wish to add a signal that indicates whether the circuit satisfies $[\vec{B}] = -[\vec{A}]$. How should we compute this signal?

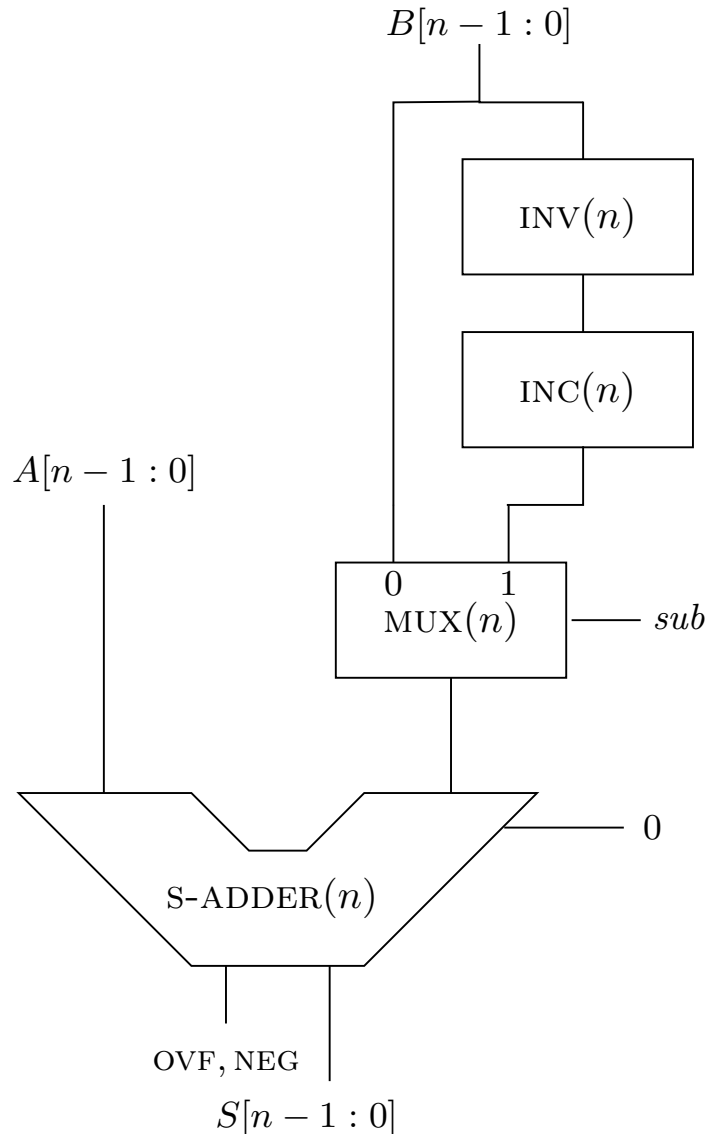
back to the negation circuit



Question:

1. When is the circuit correct?
2. Suppose we wish to add a signal that indicates whether the circuit satisfies $[\vec{B}] = -[\vec{A}]$. How should we compute this signal?
3. Does $C[n]$ indicate whether $[\vec{B}] \neq -[\vec{A}]$?

wrong implementation of $\text{ADD-SUB}(n)$



Question: Why is this design wrong?

OVF and NEG flags in high level programming

Question: High level programming languages such as C and Java do not enable one to see the value of the OVF and NEG signals (although these signals are computed by adders in all microprocessors).

OVF and NEG flags in high level programming

Question: High level programming languages such as C and Java do not enable one to see the value of the OVF and NEG signals (although these signals are computed by adders in all microprocessors).

1. Write a short program that deduces the values of these flags. Count how many instructions are needed to recover these lost flags.

OVF and NEG flags in high level programming

Question: High level programming languages such as C and Java do not enable one to see the value of the OVF and NEG signals (although these signals are computed by adders in all microprocessors).

1. Write a short program that deduces the values of these flags. Count how many instructions are needed to recover these lost flags.
2. Short segments in a low level language (Assembly) can be integrated in C programs. Do you know how to see the values of the OVF and NEG flags using a low level language?

Summary

- representation of signed numbers: sign-magnitude, one's complement, two's complement.

Summary

- representation of signed numbers: sign-magnitude, one's complement, two's complement.
- negation of two's complement numbers.

Summary

- representation of signed numbers: sign-magnitude, one's complement, two's complement.
- negation of two's complement numbers.
- reduction: two's complement addition \mapsto binary addition.

Summary

- representation of signed numbers: sign-magnitude, one's complement, two's complement.
- negation of two's complement numbers.
- reduction: two's complement addition \mapsto binary addition.
- Computation of `OVF` and `NEG` flags.

Summary

- representation of signed numbers: sign-magnitude, one's complement, two's complement.
- negation of two's complement numbers.
- reduction: two's complement addition \mapsto binary addition.
- Computation of `OVF` and `NEG` flags.
- two's complement adder and adder/subtractor.

Summary

- representation of signed numbers: sign-magnitude, one's complement, two's complement.
- negation of two's complement numbers.
- reduction: two's complement addition \mapsto binary addition.
- Computation of `OVF` and `NEG` flags.
- two's complement adder and adder/subtractor.
- all these issues are important in: designing an ALU, DSP programming, and even regular programming (signed vs. unsigned int).