

Digital Logic Design: a rigorous approach ©

Chapter 16: Signed Addition

Guy Even Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

May 24, 2020

Book Homepage:

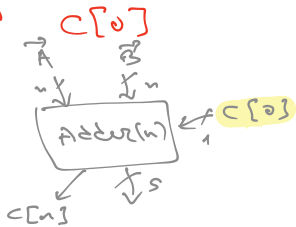
<http://www.eng.tau.ac.il/~guy/Even-Medina>

Preliminary questions

- 1 How are signed integers represented in a computer?
- 2 How are signed integers added and subtracted in a computer?
- 3 Can we use the same circuitry for adding unsigned and signed integers?

4) what is the purpose of $C[0]$ in $ADDER(n)$?

$$2^n \cdot C[n] + \langle \vec{S} \rangle = \langle \vec{A} \rangle + \langle \vec{B} \rangle + C[0]$$



Representation of negative integers

Definition

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

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

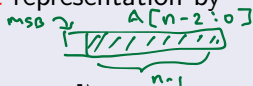
$$(-1)^0 = 1$$

$$(-1)^1 = -1$$

Definition

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

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



Definition

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

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

Comparison between representation methods

$$-2^2 \cdot A[2] + \langle A[1:0] \rangle \quad - (2^2 - 1) \cdot A[2] + \langle A[1:0] \rangle$$

binary string \vec{X}	$\langle \vec{X} \rangle$	2's comp	1's comp	sign-mag
000	0	0	0	+0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-4	-3	-0
101	5	-3	-2	-1
110	6	-2	-1	-2
111	7	-1	0	-3

- symmetric range: one's complement and sign-magnitude.
- two representations for zero: one's complement and sign-magnitude.

* why is unique repr. important?

if $A == B$ then ...

Range of representable integers

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

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

We denote the set of integers that are representable in two's complement representation using n -bit binary strings by T_n . We denote the set of integers that are representable in binary representation using n -bit binary strings by B_n .

Claim

$$T_n = \{-2^{n-1}, -2^{n-1} + 1, \dots, 2^{n-1} - 1\} \quad \text{two's comp. rep. range}$$
$$B_n = \{0, \dots, 2^n - 1\} \quad \text{binary rep. range}$$

$$B_3 = [0, 7] \quad T_3 = [-4, 3]$$

Same string different numbers

Claim

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

2's
comp

$$[\vec{A}] = \begin{cases} \langle \vec{A} \rangle & \text{if } A[n-1] = 0 \\ \langle \vec{A} \rangle - 2^n & \text{if } A[n-1] = 1 \end{cases}$$

bin. rep.

MSB

Example

Let $n = 4$ and let $A[3:0] = 0110$, $B[3:0] = 1001$, then:

$$\begin{array}{lll} \langle A[3:0] \rangle = 6, & [A[3:0]] = 6 & [\vec{A}] = \langle \vec{A} \rangle \\ \langle B[3:0] \rangle = 9, & [B[3:0]] = -7 & [\vec{B}] = \langle \vec{B} \rangle - 2^4 \end{array}$$

$$-8 + 1 = -7 \quad -7 = 9 - 16$$

claim: $[\vec{A}] = \begin{cases} \langle \vec{A} \rangle & \text{if } A[n-1] = 0 \\ \langle \vec{A} \rangle - 2^n & \text{if } A[n-1] = 1 \end{cases}$

proof: $[\vec{A}] \stackrel{\circ}{=} -2^{n-1} \cdot A[n-1] + \langle A[n-2:0] \rangle$

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

$$[\vec{A}] = \langle A[n-2:0] \rangle = \langle 0 \cdot A[n-2:0] \rangle = \langle A[n-1:0] \rangle$$

if $A[n-1] = 1$:

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

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



Claim

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

$$[\vec{A}] = \begin{cases} \langle \vec{A} \rangle & \text{if } A[n-1] = 0 \\ \langle \vec{A} \rangle - 2^n & \text{if } A[n-1] = 1 \end{cases}$$

Corollary

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

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

}
bin. rep.

}
2's comp

Computing a two's complement representation

given x find $A[n-1:0]$ s.t. $[A[n-1:0]] = x$:

Algorithm 1 two-comp(x, n) - An algorithm for computing the two's complement representation of x using n bits.

- 1 If $x \notin T_n$ return (fail).
 - 2 If $x \geq 0$ return $(0 \circ \text{bin}_{n-1}(x))$.
 - 3 If $x < 0$ return $(\text{bin}_n(x + 2^n))$.
-

proof:

Let \vec{A} denote output of algorithm. If $x \geq 0$, then $[\vec{A}] = \langle \vec{A} \rangle = x$.

If $x < 0$, then $[\vec{A}] = \langle \vec{A} \rangle - 2^n = (x + 2^n) - 2^n = x$. ↖ $A[n-1] = 0$

↖ $x + 2^n \geq 2^{n-1} \Rightarrow A[n-1] = 1$

Example

$$T_4 = \{-2^3, -2^3 + 1, \dots, 2^3 - 1\}.$$

Hence,

$$\text{two-comp}(8, 4) = \text{fail},$$

$$\text{two-comp}(5, 4) = (0 \circ \text{bin}_3(5)) = 0101,$$

$$\text{two-comp}(-6, 4) = (\text{bin}_4(-6 + 2^4)) = 1010,$$

$$\text{two-comp}(-1, 4) = (\text{bin}_4(-1 + 2^4)) = 1111.$$

Negation in two's complement representation

$$-\vec{A} = ?$$

The following claim deals with negating a value represented in two's complement representation.

Claim

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

Examples: $\vec{A} = 0110$ and $\vec{A} = 1001$.

$$-[0110] = -6 \quad \& \quad [1001] + 1 = -8 + 1 + 1 = -6$$

$$-[1001] = -(-8 + 1) = 7 \quad \& \quad [0110] + 1 = 6 + 1 = 7$$

$$\text{claim: } - [A[n-1:0]] = [\overline{A[n-1:0]}] + 1$$

proof:

$$[\overline{A[n-1:0]}] \stackrel{\Delta}{=} -2^{n-1} \cdot \overline{A[n-1]} + \langle \overline{A[n-2:0]} \rangle$$

Recall that $\text{not}(x) = 1 - x$

$$= -2^{n-1} \cdot (1 - A[n-1]) + \sum_{i=0}^{n-2} (1 - A[i]) \cdot 2^i$$

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

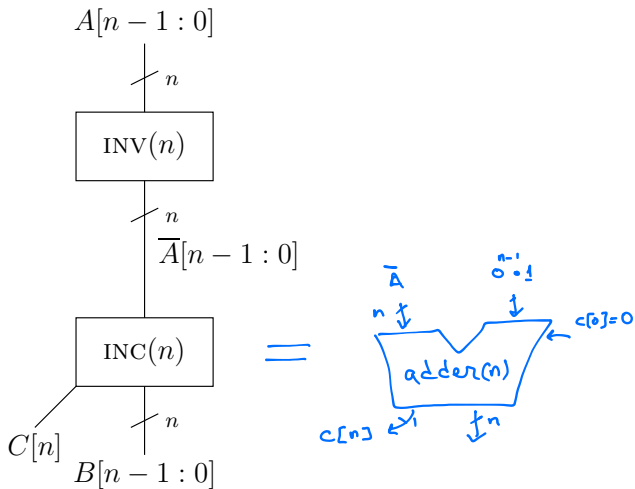
$$- 2^{n-1} + \sum_{i=0}^{n-2} 2^i$$

$$= - [A[n-1:0]] - \cancel{2^{n-1}} + (\cancel{2^{n-1}} - 1)$$



Negation based on

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

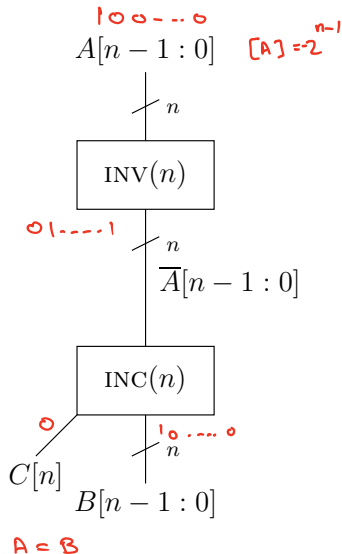


Q : $[\vec{B}] \stackrel{?}{=} - [\vec{A}]$

Negation based on

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

- We compute $\langle \overline{A[n - 1 : 0]} \rangle + 1$.
- But need $\langle \overline{A[n - 1 : 0]} \rangle + 1$.
- So $\langle C[n] \cdot B[n - 1 : 0] \rangle = \langle \overline{A[n - 1 : 0]} \rangle + 1$.
- Does $[\vec{B}] = -[\vec{A}]$?
- Suspect $C[n] = 1$! Assume $C[n] = 0$...
- So (if $C[n] = 0$)
 $\langle B[n - 1 : 0] \rangle = \langle \overline{A[n - 1 : 0]} \rangle + 1$.
- $[B[n - 1 : 0]] = \langle \overline{A[n - 1 : 0]} \rangle + 1$?



computing $-[\vec{A}]$

- Very easy in sign-magnitude representation.
- Easy in one's-complement representation.
- In two's complement representation: need to check that $-[\vec{A}] \in T_n \dots$
- Still, we need a proof and a way to tell when we fail.

(flip sign bit!)
(exercise!)

The most significant bit $A[n - 1]$ of a string $A[n - 1 : 0]$ that represents a two's complement integer is often called the **sign-bit** of \vec{A} . The following claim justifies this term.

Claim

(exercise)

$$[A[n - 1 : 0]] < 0 \iff A[n - 1] = 1.$$

Do not be misled by the term sign-bit. Computing the absolute value of $[\vec{A}]$ requires negation...

Example: $A = 1111$ and $A = 0111$.

$$[A] = -1$$

$$[A] = 7$$

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.

Claim

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

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

Corollary

(by ind. on #rep. of $A[n-1]$)

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

Example:

$$\dots = [1 \cdot 110] = [110] = [10]$$

$$[11111111111111111110] = [10] = -2$$

$$[11111111111111111111] = [1] = -1$$

claim: $A[n] = A[n-1] \Rightarrow [A[n:0]] = [A[n-1:0]]$

proof: $[A[n:0]] \stackrel{\Delta}{=} -2^n \cdot A[n] + 2^{n-1} \cdot A[n-1] + \langle A[n-2:0] \rangle$

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

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

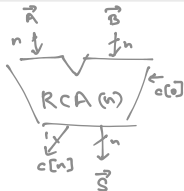
$$\stackrel{\Delta}{=} [A[n-1:0]]$$



Reduction: two's complement addition to binary addition

Goal: two's complement addition

$$[\vec{A}] + [\vec{B}] + C[0].$$



Suppose:

$$A[n-1:0], B[n-1:0], S[n-1:0] \in \{0,1\}^n$$

$$C[0], C[n] \in \{0,1\}$$

satisfy

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

- When does the output $S[n-1:0]$ satisfy:

$$[\vec{S}] = [A[n-1:0]] + [B[n-1:0]] + C[0]? \quad (1)$$

- How can we know that Equation (1) holds?

Reduction of two's comp addition to binary addition

Theorem

Let $C[n - 1]$ denote the carry-bit in position $[n - 1]$ associated with the binary addition described in ~~Equation 16~~ and let

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

Then,

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

$$C[n - 1] - C[n] = 1 \quad \implies \quad z > 2^{n-1} - 1 \quad (3)$$

$$z \in T_n \quad \iff \quad C[n] = C[n - 1] \quad (4)$$

$$z \in T_n \quad \implies \quad z = [S[n - 1 : 0]]. \quad (5)$$

Example

$[A[3 : 0]]$	-3	-4	-6	7
$[B[3 : 0]]$	-5	-5	5	1
$C[0]$	1	0	0	1
<hr/>				
$C[n]$	1	1	1	0
$C[n - 1]$	1	0	1	1
$[S[n - 1 : 0]]$	-7	7	-1	-7
z	-7	-9	-1	9

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

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

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

↙
adder
RCRA(n-1)

$$= -2^{n-1} (A[n-1] + B[n-1] - C[n-1]) + \langle S[n-2:0] \rangle$$

Now: $\underbrace{FA_{n-1}}_{\text{of } RCRA(n)} \quad A[n-1] + B[n-1] + C[n-1] = 2 \cdot c[n] + S[n-1]$

$$\Rightarrow A[n-1] + B[n-1] - C[n-1] = 2(c[n] - c[n-1]) + S[n-1]$$

We conclude that

$$z = -2^n (c[n] - c[n-1]) + [S[n-1:0]]$$

proof (cont.):

$$z = -2^n (c[n] - c[n-1]) + [S[n-1:0]]$$

cases:

$$1) c[n] = c[n-1] \Rightarrow z = [S[n-1:0]]$$

$$2) c[n] - c[n-1] = 1$$

$$\Rightarrow z = -2^n + [S[n-1:0]] \leq -2^n + 2^{n-1} = -2^{n-1}$$

$$\Rightarrow z \notin T_n \quad (z \text{ too small})$$

$$3) c[n] - c[n-1] = -1$$

$$\Rightarrow z = 2^n + [S[n-1:0]] \geq 2^n - 2^{n-1} = 2^{n-1}$$

$$\Rightarrow z \notin T_n \quad (z \text{ too big})$$

□

Detecting overflow

Overflow - the sum of signed integers is not in T_n .

Definition

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

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

the term “out-of-range” is more appropriate than “overflow” (which suggests that the sum is too big). Favor tradition...
By the theorem

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

Determining the sign of the sum

Definition

The signal `NEG` is defined as follows:

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

brute force method:

$$\text{NEG} = \begin{cases} S[n-1] & \text{if no overflow} \\ 1 & \text{if } C[n] - C[n-1] = 1 \\ 0 & \text{if } C[n-1] - C[n] = 1. \end{cases} \quad (6)$$

Claim

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

claim: $NEG = XOR_3(A[n-1], B[n-1], C[n])$

proof: sign extend \vec{A} & \vec{B} by one position.

$$\begin{array}{ccccccc} & & & & & & C_0 \\ C_n & & & & & & \\ A_{n-1} & A_{n-1} & A_{n-2} & \dots & & & A_0 \\ B_{n-1} & B_{n-1} & B_{n-2} & \dots & & & B_0 \end{array}$$

$$\tilde{C}_{n+1} \left| \begin{array}{cccc} \tilde{S}_n & S_{n-1} & S_{n-2} & \dots & S_0 \end{array} \right.$$

exercise: $z \in T_{n+1}$ (hence $z = [\tilde{S}_n \circ S[n-1:0]]$)

hence: $NEG = \tilde{S}_n = XOR_3(A_{n-1}, B_{n-1}, C_n)$



A two's-complement adder

Definition

A **two's-complement adder** with input length n is a combinational circuit specified as follows.

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:

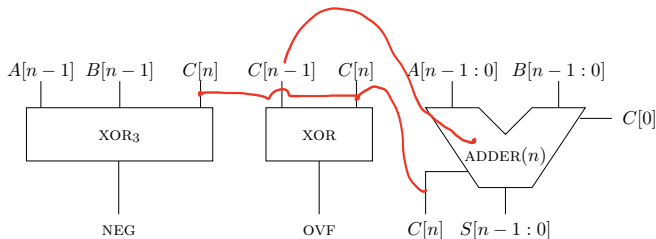
$$z \in T_n \implies [S[n-1:0]] = z$$

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

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

We denote a two's-complement adder by $\text{S-ADDER}(n)$.

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



In an arithmetic logic unit (ALU), one may share the same ADDER(n) for signed addition and unsigned addition.

A two's complement adder/subtractor

Definition

A **two's-complement adder/subtractor** with input length n is a combinational circuit specified as follows.

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:

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

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

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

We denote a two's-complement adder/subtractor by $ADD-SUB(n)$.

An implementation of an ADD-SUB(n)

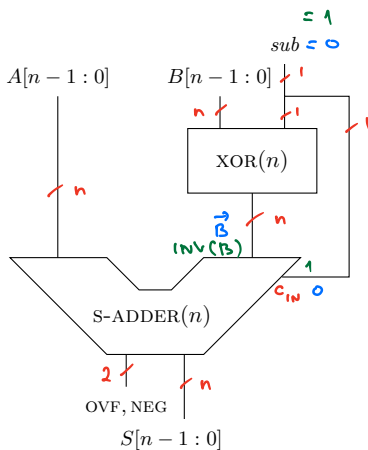
sub = 0
correctness holds
because
S-ADDER(n)
is correct

sub = 1

$$[\vec{A}] + [\text{INV}(\vec{B})] + 1$$

$= -[B]$

$$= [\vec{A}] - [\vec{B}]$$



Claim

The implementation of ADD-SUB(n) is correct.

- three ways for representing negative integers: sign-magnitude, one's-complement, and two's complement. We then focused on two's complement representation.
- Negating.
- Properties of two's complement representation: (i) modulo 2^n congruent to binary rep. (ii) sign bit. (iii) sign-extension.
- Reduce the task of two's complement addition to binary addition, and: (i) overflow detection (ii) sign of the sum even if an overflow occurs.
- Implementation of a circuit of adder/subtractor (basic part in ALU).