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?

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

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

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

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.

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

$$T_n = \left\{-2^{n-1}, -2^{n-1}+1, \ldots, 2^{n-1}-1\right\} \quad \textit{two's comp. rep. range}$$

$$B_n = \{0, \ldots, 2^n - 1\} \quad \textit{binary rep. range}$$

Same string different numbers

Claim

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

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

Example

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

$$\langle A[3:0] \rangle = 6\,, \qquad [A[3:0]] = 6 \qquad \left[\vec{A}\right] = \langle \vec{A} \rangle$$

$$\langle B[3:0] \rangle = 9\,, \qquad [B[3:0]] = -7 \qquad \left[\vec{B}\right] = \langle \vec{B} \rangle - 2^4$$

Alternative definition

Claim

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

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

Corollary

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

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

Computing a two's complement representation

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 $(0 \circ bin_{n-1}(x))$.
3. If $x < 0$ return $(bin_n(x + 2^n))$.

proof:

Let $\vec{A}$ denote output of algorithm. If $x \geq 0$, then $\left[\vec{A}\right] = \langle\vec{A}\rangle = x$. If $x < 0$, then $\left[\vec{A}\right] = \langle\vec{A}\rangle - 2^n = (x + 2^n) - 2^n = x$.

Computing a two's complement representation

Example

$$T_4 = \left\{-2^3, -2^3+1, \ldots, 2^3-1\right\}.$$

Hence,

$$\begin{aligned}
\text{two-comp}(8,4) &= \text{fail}\ ,\\
\text{two-comp}(5,4) &= (0 \circ bin_3(5)) = 0101\ ,\\
\text{two-comp}(-6,4) &= (bin_4(-6+2^4)) = 1010\ ,\\
\text{two-comp}(-1,4) &= (bin_4(-1+2^4)) = 1111\ .
\end{aligned}$$

Negation in two's complement representation

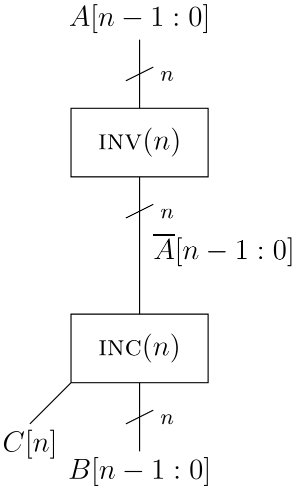
The following claim deals with negating a value represented in two's complement representation.

Claim

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

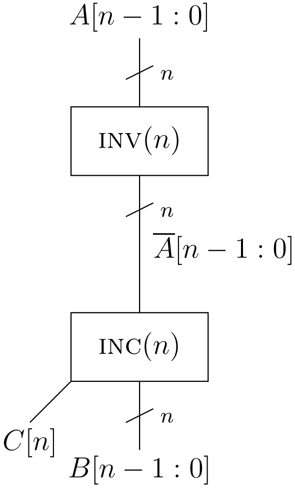
Examples: $\vec{A} = 0110$ and $\vec{A} = 1001$.

Negation based on $-[A[n-1:0]] = [\textsc{inv}(A[n-1:0])] + 1$



Negation based on $-\left[A[n-1:0]\right] = \left[\textsc{inv}(A[n-1:0])\right] + 1$

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



Negation

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

Sign bit

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

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

Do not be misled by the term sign-bit. Computing the absolute value of $\left[\vec{A}\right]$ requires negation...
Example: $A = 1111$ and $A = 0111$.

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

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

Example:

$$[11111111111111111110] = [10] = -2$$
$$[11111111111111111111] = [1] = -1$$

Reduction: two's complement addition to binary addition

Goal: two's complement addition

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

Suppose:

$$\begin{aligned} A[n-1:0], B[n-1:0], S[n-1:0] &\in \{0,1\}^n \\ C[0], C[n] &\in \{0,1\} \end{aligned}$$

satisfy

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

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

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

- 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,

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

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

$$z \in T_n \iff C[n] = C[n-1] \tag{4}$$

$$z \in T_n \implies z = [S[n-1:0]]. \tag{5}$$

Example

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

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 $\textsc{ovf}$ is defined as follows:

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

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

$$\textsc{ovf} = \textsc{xor}(C[n-1], C[n]).$$

Determining the sign of the sum

Definition

The signal NEG is defined as follows:

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

brute force method:

$$\textsc{neg} = \begin{cases} S[n-1] & \text{if no overflow} \\ 1 & \text{if } C[n] - C[n-1] = 1 \\ 0 & \text{if } C[n-1] - C[n] = 1. \end{cases} \tag{6}$$

Claim

$$\textsc{neg} = \textsc{xor}_3(A[n-1], B[n-1], C[n]).$$

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:

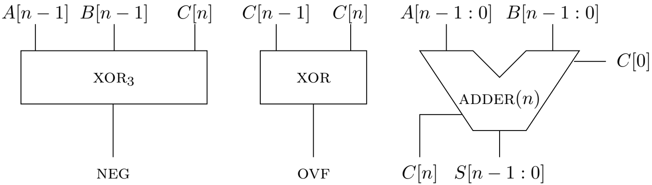
$$z \triangleq [A[n-1:0]] + [B[n-1:0]] + C[0].$$

The functionality is defined as follows:

$$\begin{aligned} z \in T_n &\implies [S[n-1:0]] = z \\ z \in T_n &\iff \text{OVF} = 0 \\ z < 0 &\iff \text{NEG} = 1. \end{aligned}$$

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 $\text{NEG}, \text{OVF} \in \{0,1\}$.

Functionality: Define z as follows:

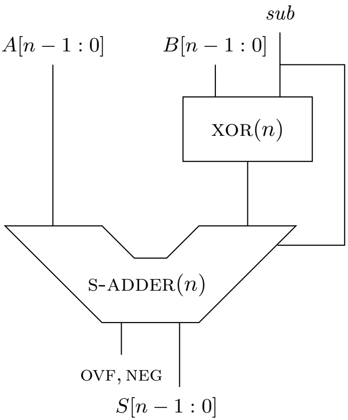
$$z \triangleq [A[n-1:0]] + (-1)^{sub} \cdot [B[n-1:0]] .$$

The functionality is defined as follows:

$$\begin{aligned} z \in T_n &\implies [S[n-1:0]] = z \\ z \in T_n &\iff \text{OVF} = 0 \\ z < 0 &\iff \text{NEG} = 1. \end{aligned}$$

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

An implementation of an ADD-SUB(n)



Claim

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

Summary

- 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).