جامعة الملك سعود
King Saud University

CSC 220: Computer Organization

Unit 4

Signed Number Representation

Prepared by:
Md Saiful Islam, PhD

Updated by:
Isra Al-Turaiki, PhD

Department of Computer Science
College of Computer and Information Sciences

Overview

- Unsigned Representation
- Representation of signed numbers
  - Signed Magnitude Representation
  - One's Complement Notation
  - Two's Complement Notation
- Signed Overflow
- Sign Extension

Unsigned Representation

- Positive integers and the number zero can be represented as *unsigned numbers*.
- Ex: 8 bit unsigned numbers:

Position	7	6	5	4	3	2	1	0
Contribution	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
157	1	0	0	1	1	1	0	1
1	0	0	0	0	0	0	0	1
10	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0

Binary Addition

Addition is simple:

- Add individual bits
- Propagate carries

The addition table for binary numbers is

0+0 =	0
0+1 =	1
1+0 =	1
1+1 =	1 0

result

carry 1 to the next column

Binary Addition

Example: Add 13_{10} and 11_{10} in binary.

$$
\begin{array}{rl}
 & 1111 \longleftarrow \text{carries} \\
13_{10} = & 1101 \\
11_{10} = & \underline{1011} \\
 & 11000 = 24_{10}
\end{array}
$$

Binary Subtraction

Minuend →	0	1	1
Subtrahend →	- 0	- 0	- 1
	0	1	0

A "borrow" is made from the next highest bit position

```
1 0
- 1
---
  1
```

Example:

```
  0 1       0
  1 0 0 1 1 1 0 1 1
- 0 0 1 0 1 0 1 0 1
-------------------
  0 1 1 1 0 0 1 1 0
```

Unsigned Representation

Advantages:

- Simple addition
- One representation of zero

Disadvantages

- Negative numbers can not be represented.

Representation of signed numbers

- Is a representation of negative numbers possible?
  - you can not just stick a negative sign in front of a binary number. (it does not work like that)

- There are three methods used to represent negative numbers.
  - Signed magnitude representation
  - One's complement representation
  - Two's complement representation

Signed Magnitude Representation

For an n-bit word,

– the first bit is the sign (0 for +ve, 1 for –ve).

– the remaining n-1 bits represent the magnitude of the number.

Example: for n=4

$+3_{10}$	0011
-3_{10}	1011

Signed Magnitude Representation

Exercise: Find the signed magnitude in 8 bits?

	Positive	Negative
37_{10}		
24_{10}		
63_{10}		

Signed Magnitude Representation

Exercise: Find the signed magnitude in 8 bits?

	Positive	Negative
37_{10}	0010 0101	1010 0101
24_{10}	0001 1000	1001 1000
63_{10}	0011 1111	1011 1111

Disadvantage of Signed Magnitude

- Addition and subtractions are difficult:
  - Signs and magnitude are processed separately
  - Subtraction requires comparison of signs and magnitudes.
- There are two representations of 0 : 00000000 and 10000000

  To test if a number is 0 or not, the CPU will need to see whether it is 00000000 or 10000000.

  ➢0 is always performed in programs. Therefore, having two representations of 0 is inconvenient.

One's Complement

- Given a number N in binary having n digits, the 1's complement of N is defined as $(2^n - 1) - N$.
- $2^n - 1$ is a binary number represented by n 1's.

- **Example:** find the 1's complement of $(+2_{10})$
- Solution: remember $(+2_{10}) = (0010)_2$ and n=4

$(2^n - 1) - N = (1111)_2 - (0010)_2 = 1101$

One's Complement

- **1's complement shortcut:**

  1's complement of a binary number is formed by changing all 1's to 0's and all 0's to 1's.
- The sign bit is complemented along with rest of bits.
- The sign bit is 0 for +ve and 1 for –ve numbers.
- Complementing a bit is equivalent to subtracting it from 1.

X	X'	1-X
0	1	1
1	0	0

One's Complement

- **Example:** find the 1's complement of the following:

Number	1's complement
1011001	
0001111	

One's Complement

- **Example:** find the 1's complement of the following:

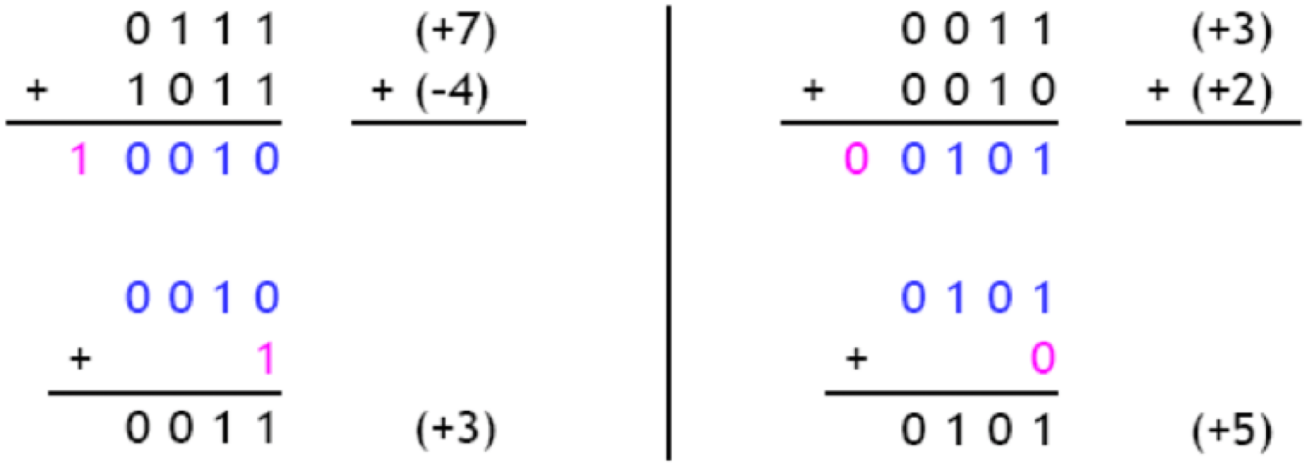
Number	1's complement
1011001	0100110
0001111	1110000

Addition in One's Complement

There are two steps in adding 1's complement numbers:

1. Do unsigned addition, including the sign bit
2. Take the carry out and add it to the sum

Example:



Problem: Two representation of zero (0000 = +0, 1111 = -0)

Two's Complement Representation

- Given a number N in binary having n digits, the 2's complement of N is defined as $2^n - N$.
- 2^n is represented by a binary number that consists of a 1 followed by n 0s.

- **Example:** find the 2's complement of $+2_{10}$
- Solution: $+2_{10} = (0010)_2$ and n=4

  $2^n - N = (10000)_2 - (0010)_2 = 1110$

Two's Complement Representation

The sign bit is 0 for +ve and 1 for –ve numbers.

2's complement shortcuts:

- Methods 1: Complement each bit and then add 1.
- Method 2: Complement all of the bits to the left of the rightmost one.

Example: find the 2's complement of 0101100

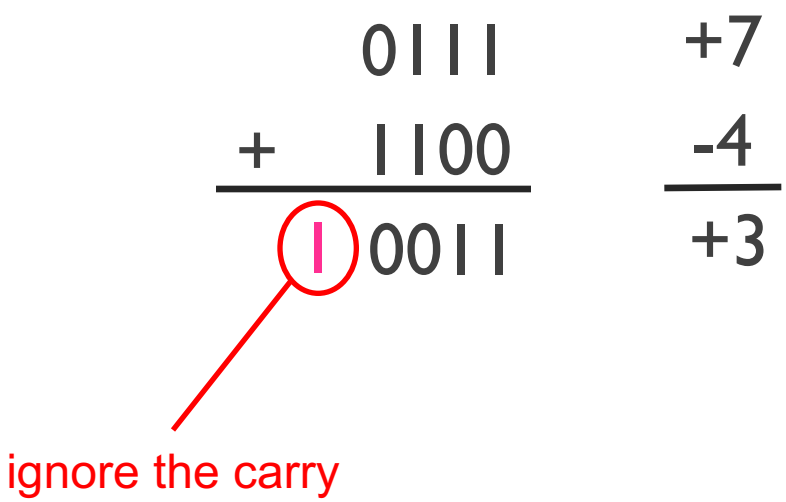
Answer: 1010100

Addition in Two's Complement

Adding 2's complement numbers is done as follows:

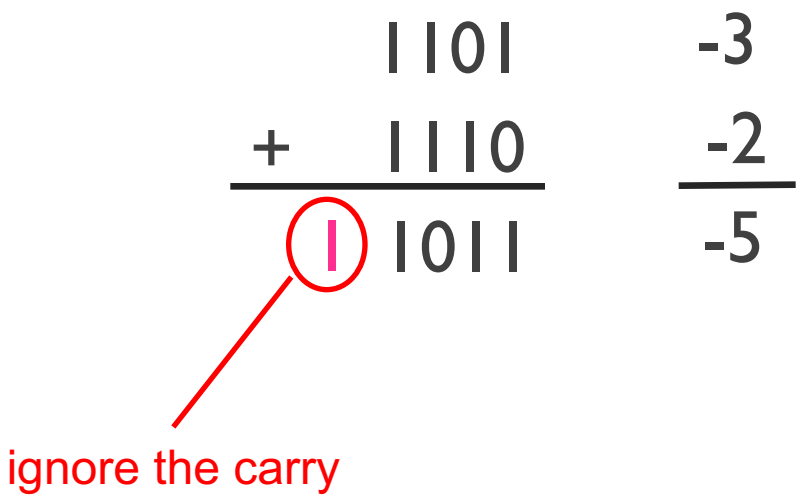
1. Do unsigned addition, including the sign bit.
2. Any carry from the sign position is *ignored*.

Example: Add 0111 to 1100



Addition in Two's Complement

Example: Add 1101 to 1110



An Algebraic Explanation

- For *n*-bit numbers, the negation of B in two's complement is 2^n - B. (This was one of the alternate ways of negating a two's complement number.)

$$\begin{aligned} A - B &= A + (-B) \\ &= A + (2^n - B) \\ &= (A - B) + 2^n \end{aligned}$$

- If $A \geq B$, then (A - B) has to be positive, and the 2^n represents a carry out of 1. Discarding this carry out leaves us with the desired result, (A - B).
- If $A < B$, then (A - B) must be negative, and 2^n - (A - B) corresponds to the correct result -(A - B) in two's complement form.

Advantages of Two's Complement Notation

- One representation of zero as 0000 using 4-bit binary sequence.
- It is easy to add two numbers.
- Subtraction can be easily performed.
- Multiplication is just a repeated addition.
- Division is just a repeated subtraction
- Two's complement is widely used in ALU

Comparing Signed Number Systems

- Positive numbers are the same in all representation.
- There are two representations of zero in signed magnitude and 1's complement→ complicate things.
- In 2's complement one more negative number -8 (but no +8)
- 2's complement is preferred because it has one representation of zero and a simple addition algorithm.

Decimal	SM	1C	2C
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000	0000
-0	1000	1111	–
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	–	–	1000

Ranges of Signed Number Systems

How many negative and positive numbers can be represented in each of the 4-bit systems?

	Unsigned	SM	1C	2C
Smallest	0000 (0)	1111 (-7)	1000 (-7)	1000 (-8)
Largest	1111 (15)	0111 (+7)	0111 (+7)	0111 (+7)

The ranges for n-bit numbers (including the sign bit) are as follows:

	Unsigned	SM	1C	2C
Smallest	0	$-(2^{n-1}-1)$	$-(2^{n-1}-1)$	-2^{n-1}
Largest	2^n-1	$+(2^{n-1}-1)$	$+(2^{n-1}-1)$	$+(2^{n-1}-1)$

Overflow

- With 4-bit 2's complement, the largest representable decimal value is +7 and the smallest is -8.
- But, what if we compute 4+5 or (-4)+(-5)?

```
  0 1 0 0     (+4)        1 1 0 0     (-4)
+ 0 1 0 1   + (+5)      + 1 0 1 1   + (-5)
-----------  -------    -----------  -------
  0 1 0 0 1   (-7)        1 0 1 1 1   (+7)
```

- if the correct representation of the sum (including sign) requires more than n bits, we say that an *overflow* occurs.

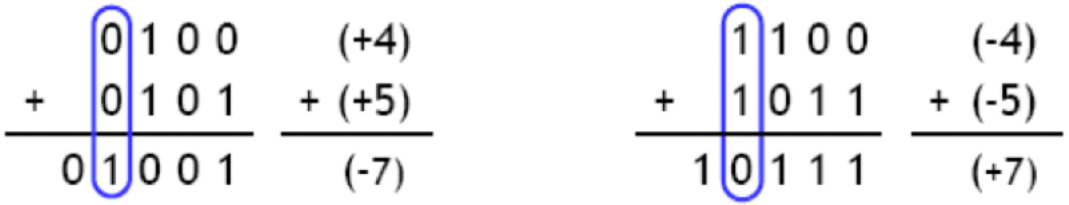
Overflow

Signed overflow is different from unsigned:

- Unsigned:
  - an overflow is detected in addition from the end carry out of the most significant position.
- Signed:
  - the most significant bit always represents the sign.
  - When two signed numbers are added, the sign bit is treated as a part of the number, and an end carry of 1 does not necessarily indicate an overflow.

Detecting Overflow

- The easiest way to detect signed overflow is to look at the sign bit.



- An overflow may occur if the two numbers added are both positive or both negative.
  - ❑ If you add two positive numbers and get a negative result.
  - ❑ If you add two negative numbers and get a positive result.

No Overflow Examples

Example 1

+3	0011
+4	0100
+7	0111

Correct answer

Example 2

+5	0101
−6	1010
−1	1111

Correct answer

No Overflow Examples

Example 3

$$\begin{array}{rr} -5 & 1011 \\ +6 & 0110 \\ \hline +1 & (1)0001 \end{array}$$

Correct answer

Example 4

$$\begin{array}{rr} -3 & 1101 \\ -4 & 1100 \\ \hline -7 & (1)1001 \end{array}$$

Correct answer

correct answer when the carry from the sign bit is ignored (this is *not* an overflow)

Overflow Examples

Example 5

$$\begin{array}{rr} +5 & 0101 \\ +6 & 0110 \\ \hline & 1011 \end{array}$$

wrong answer because of overflow (+11 requires 5 bits including sign)

Example 6

$$\begin{array}{rr} -5 & 1011 \\ -6 & 1010 \\ \hline & (1)0101 \end{array}$$

wrong answer because of overflow (-11 requires 5 bits including sign)

Sign Extension

- Decimal numbers are assumed to have an infinite number of 0s in front of them, which helps in "lining up" values for arithmetic operations.

$$\begin{array}{r} 2\ 2\ 5 \\ +\ 0\ 0\ 6 \\ \hline 2\ 3\ 1 \end{array}$$

- You need to be careful in extending signed binary numbers, because the leftmost bit is the *sign* and not part of the magnitude.
- To extend a signed binary number, you have to replicate the sign bit. If you just add 0s in front, you might accidentally change a negative number into a positive one!
- For example, consider going from 4-bit to 8-bit numbers.

(+5) 0101 ⟶ 0000 0101 (+5)

(-4) 1100 ⟶ 1111 1100 (-4)