CMPE 212
Principles of Digital Design

Lecture 3

Signed Number Representation

February 1, 2016

www.csee.umbc.edu/~younis/CMPE212/CMPE212.htm

Lecture's Overview

- ***Previous Lecture:***
  - ➔ Different representations of numbers
    (Radix, Weighted Position Code, Binary, octal and hexadecimal)
  - ➔ Arithmetic in different radix
  - ➔ Converting numbers between bases
    (multiplication and remainder conversion method, bit grouping)
  - ➔ Converting integer and fractions
- ***This Lecture:***
  - ➔ Representation of signed fixed numbers
    - ➢ Signed Magnitude
    - ➢ Diminished radix Complement
    - ➢ Excess (Biased)
    - ➢ Radix Complement

Signed Fixed Point Numbers

- For an 8-bit number, there are $2^8 = 256$ possible bit patterns. These bit patterns can represent negative numbers if we choose to assign bit patterns to numbers in this way. We can assign half of the bit patterns to negative numbers and half of the bit patterns to positive numbers.

- We cover four signed representations of numbers:
  - Signed Magnitude
  - Radix complement (e.g., Two's Complement for binary)
  - Diminished radix complement (e.g., One's complement for binary)
  - Excess-M (Biased)

* Slide is (partially) courtesy of M. Murdocca and V. Heuring

Signed Magnitude

- Also known as "sign and magnitude," the leftmost bit is the sign (0 = positive, 1 = negative) and the remaining bits are the magnitude.

- <u>Example:</u>

  $+25_{10} = 00011001_2$, $-25_{10} = 10011001_2$

- Two representations for zero:

  $+0 = 00000000_2$, $-0 = 10000000_2$

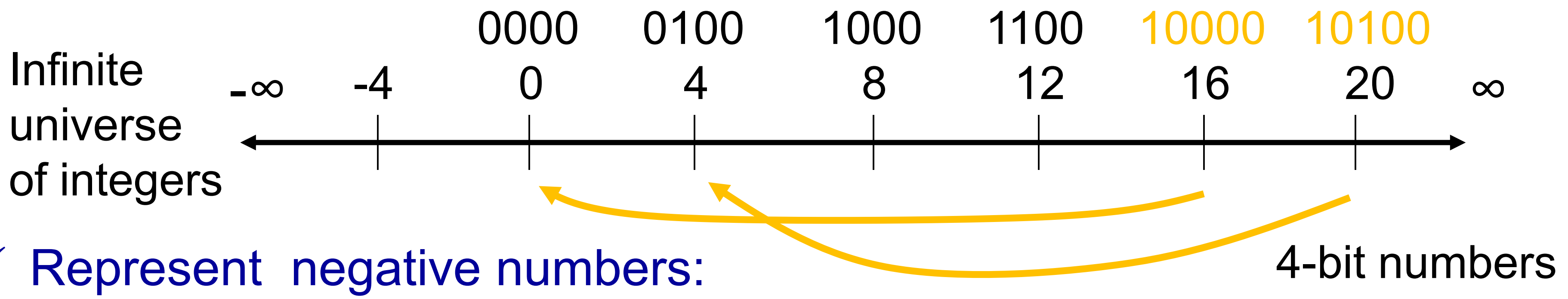
- Using an 8-bit representation:
  - Largest number is $+127_{10}$, smallest number is -127_{10}

- Using an N-bit representation:
  - Largest number is $+(2^{N-1}-1)_{10}$, smallest number is $-(2^{N-1}-1)_{10}$

* Slide is courtesy of M. Murdocca and V. Heuring

Problems with Finite Math

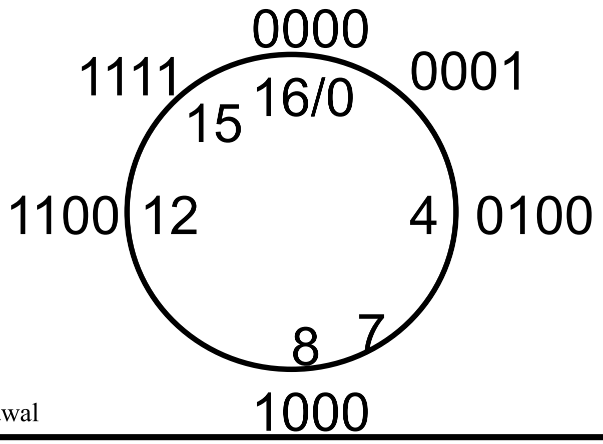
- Finite size of representation:
  - Digital circuits cannot be arbitrarily large
  - Overflow detection – easy to determine when the number becomes too large.

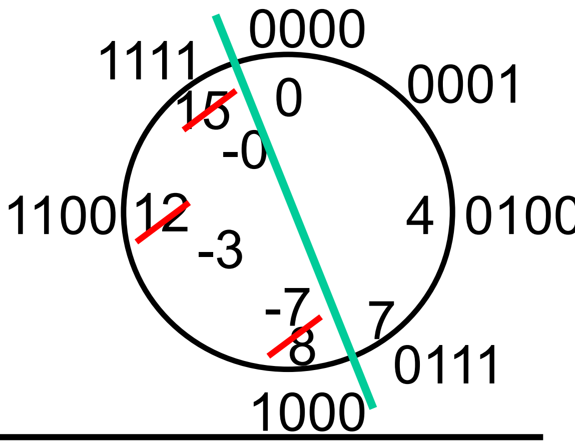


✓ Represent negative numbers:
  - Unique representation of 0

Modulo-16 (4-bit) universe

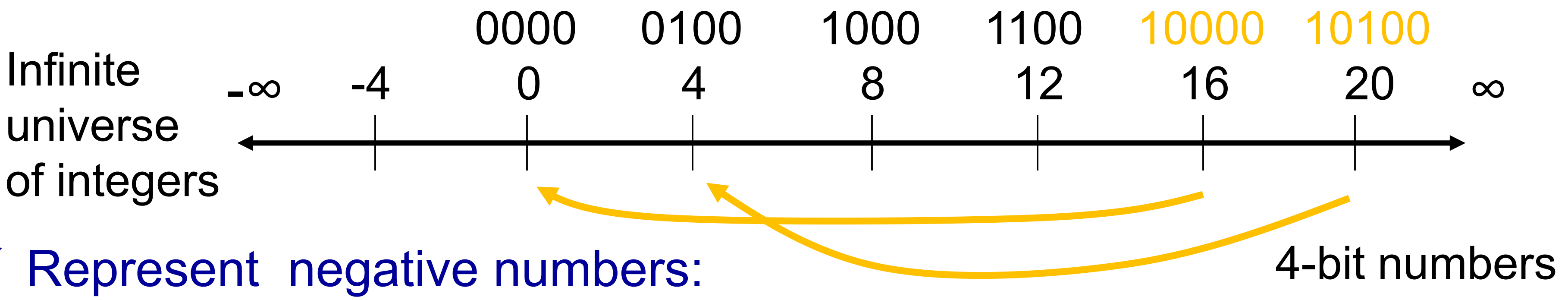
Only 16 integers:
0 through 15, or
– 7 through 7





Problems with Finite Math

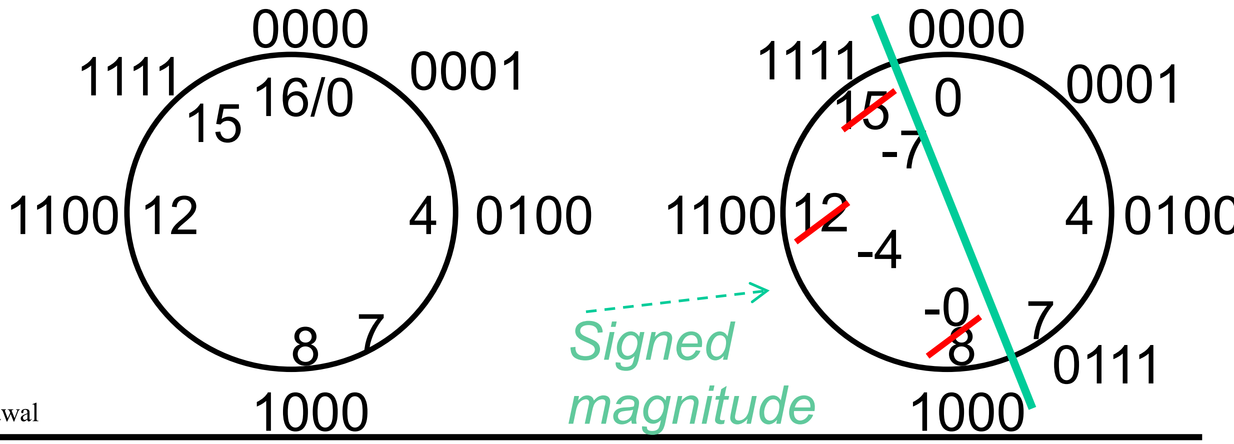
- Finite size of representation:
  - Digital circuits cannot be arbitrarily large
  - Overflow detection – easy to determine when the number becomes too large.



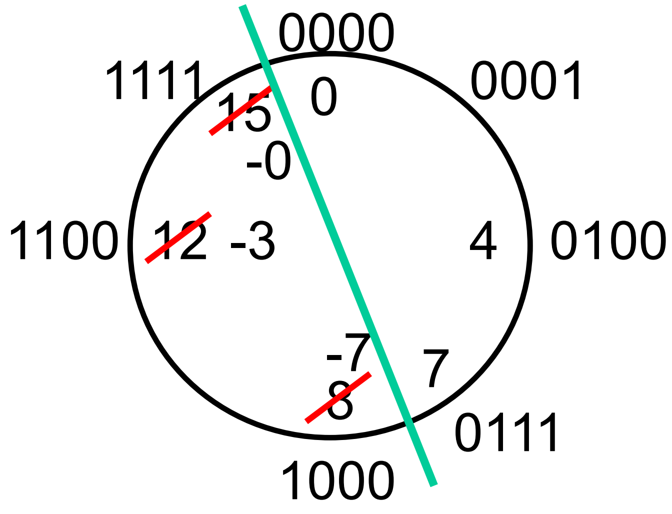
✓ Represent negative numbers:
  - Unique representation of 0

Modulo-16 (4-bit) universe

Only 16 integers: 0 through 15, or – 7 through 7



One Way to Divide Universe 1's Complement Numbers



Decimal magnitude	Binary number	
	Positive	Negative
0	0000	1111
1	0001	1110
2	0010	1101
3	0011	1100
4	0100	1011
5	0101	1010
6	0110	1001
7	0111	1000

Negation rule: invert bits.

Problem: $0 \neq -0$

One's Complement

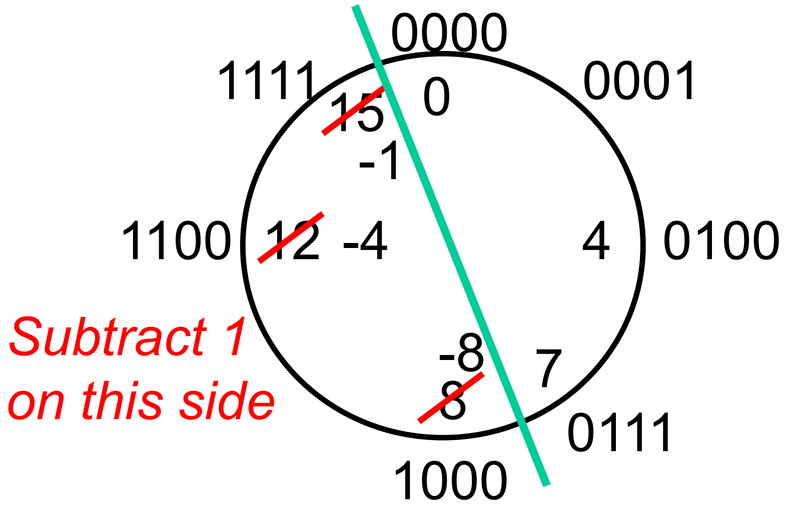
- The leftmost bit is the sign (0 = positive, 1 = negative)
- Negative of a number is obtained by subtracting each bit from 2 (essentially, complementing each bit from 0 to 1 or from 1 to 0). This goes both ways: converting positive numbers to negative numbers, and converting negative numbers to positive numbers
- Example:

  $+25_{10} = 00011001_2$, $-25_{10} = 11100110_2$
- Two representations for zero:

  $+0 = 00000000_2$, $-0 = 11111111_2$
- Using an 8-bit representation:
  - Largest number is $+127_{10}$, smallest number is -127_{10}
- Using an N-bit representation:
  - Largest number is $+(2^{N-1}-1)_{10}$, smallest number is $-(2^{N-1}-1)_{10}$

* Slide is courtesy of M. Murdocca and V. Heuring

Another Way to Divide Universe 2's Complement Numbers



Negation rule: invert bits and add 1

Decimal magnitude	Binary number	
	Positive	Negative
0	0000	
1	0001	1111
2	0010	1110
3	0011	1101
4	0100	1100
5	0101	1011
6	0110	1010
7	0111	1001
8		1000

Two's Complement

- The leftmost bit is the sign (0 = positive, 1 = negative). Negative of a number is obtained by adding 1 to the one's complement negative. This goes both ways, converting between positive and negative numbers.
- Example (recall that -25_{10} in one's complement is 11100110_2):

  $+25_{10} = 00011001_2$, $-25_{10} = 11100111_2$
- Only one representations for zero:

  $+0 = 00000000_2$, $-0 = 00000000_2$
- Using an 8-bit representation:
  - Largest number is $+127_{10}$, smallest number is -128_{10}
- Using an N-bit representation:
  - Largest number is $+(2^{N-1}-1)_{10}$, smallest number is $-(2^{N-1})_{10}$

* Slide is courtesy of M. Murdocca and V. Heuring

Complement Representation

Complement means "something that completes":

e.g., X + complement (X) = "Whole".

The radix complement of a number x is:

$$R^n - x = (R^n - 1) - x + 1$$

Meanwhile, the diminished radix complement of x is:

$$(R^n - 1) - x$$

Where *n* is the digit count in *x* and *R* is the base (radix)

- Examples:

10's Comp of $(56421)_{10}$	= 99999 – 56421+ 1	= $(43579)_{10}$
8's Comp of $(56421)_8$	= 77777 – 56421+ 1	= $(21357)_8$
2's Comp $(1011)_2$	= 1111 – 1011 + 1	= $(0101)_2$
9's Comp of $(56421)_{10}$	= 99999 – 56421	= $(43578)_{10}$
7's Comp of $(56421)_8$	= 77777 – 56421	= $(21356)_8$
1's Comp $(1011)_2$	= 1111 – 1011	= $(0100)_2$

32-bit wide Numbers

$(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 = (0)_{10}$

$(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001)_2 = (1)_{10}$

$(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010)_2 = (2)_{10}$

....

$(0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101)_2 = (2,147,483,645)_{10}$

$(0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110)_2 = (2,147,483,646)_{10}$

$(0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111)_2 = (2,147,483,647)_{10}$

$(1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 = (-2,147,483,648)_{10}$

$(1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001)_2 = (-2,147,483,647)_{10}$

$(1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010)_2 = (-2,147,483,646)_{10}$

....

$(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101)_2 = (-3)_{10}$

$(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110)_2 = (-2)_{10}$

$(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111)_2 = (-1)_{10}$

- Two's complement does have one negative number that has no corresponding positive number
- The most positive and the least negative number are different in all bits

Quick negation for Two's Complement

Method 1:

- Convert every 1→0 and every 0→1 and then add 1 to the rest

Method 2:

1. Move from right to left leave every leading 0's until reaching the first 1
2. Convert every 0→1 and 1→0 afterward until reaching the right end

Example: Negate $(2)_{10}$

$(2)_{10} = (0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010)_2$

Method 1:

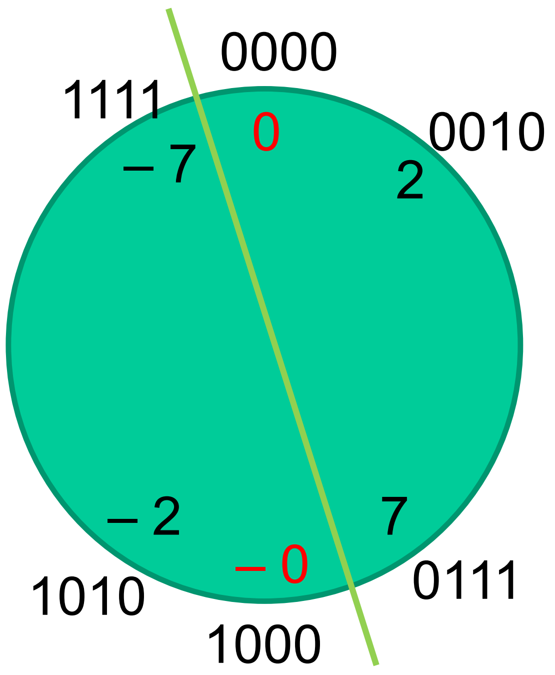
```
   1111 1111 1111 1111 1111 1111 1111 1101
 +                                        1
   ----------------------------------------
   1111 1111 1111 1111 1111 1111 1111 1110
```

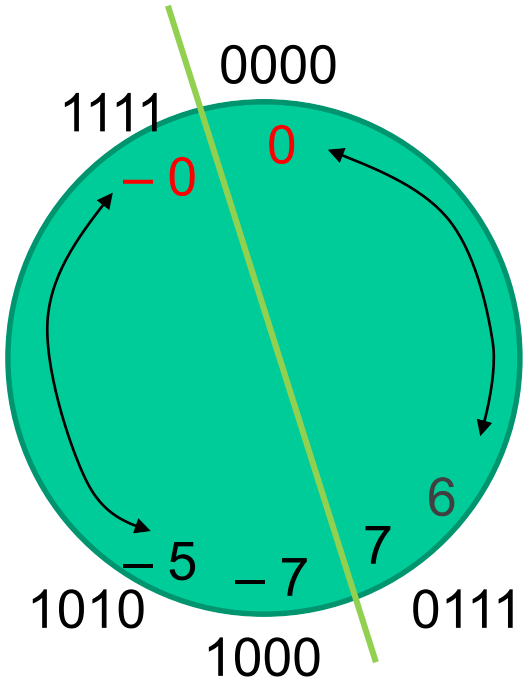
Method 2: $\underbrace{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 00}10$

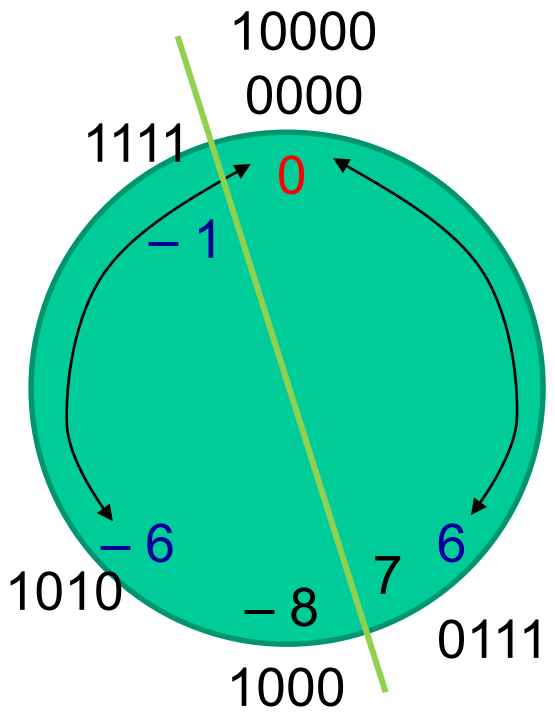
1111 1111 1111 1111 1111 1111 1111 1110

General Method: Signed Binary Integers

- For (n) binary digit numbers there are 2^n unique combinations
- Partition 2^n integers into two sets: positive and negative







1010 = – 2

1010 = – 5

1010 = – 6

Signed magnitude

1’s complement integers

2’s complement integers

Slide contents are courtesy of Vishwani D. Agrawal

Excess (Biased)

- The leftmost bit is the sign (usually 1 = positive, 0 = negative). Representations of a number are obtained by adding a bias to the two's complement representation. This goes both ways, converting between positive and negative numbers.
- The effect is that numerically smaller numbers have smaller bit patterns, simplifying comparisons for floating point exponents.
- Example (excess 128 "adds" 128 to the two's complement version, ignoring any carry out of the most significant bit):

  $+12_{10} = 10001100_2$, $-12_{10} = 01110100_2$

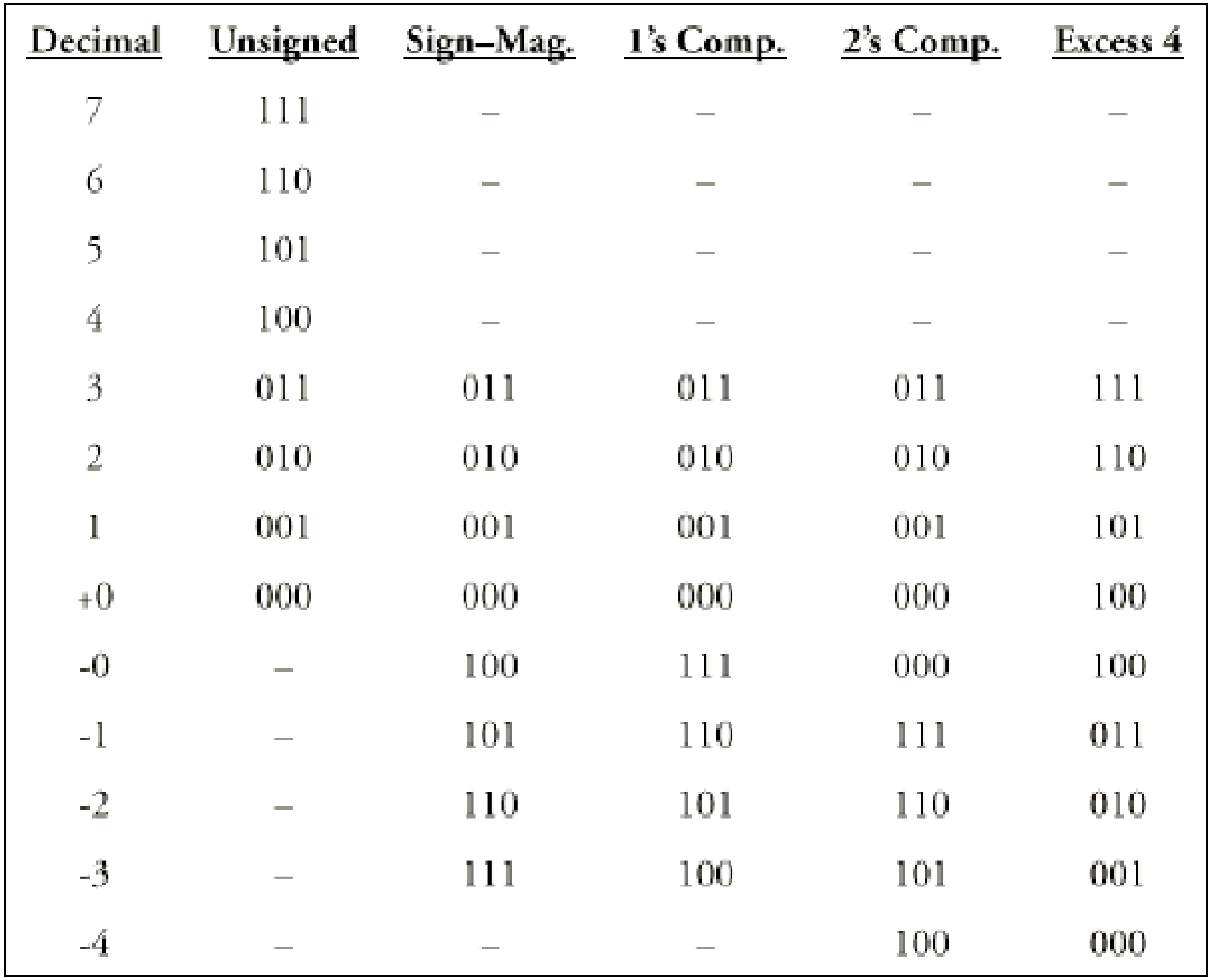
- Only one representations for zero:

  $+0 = 10000000_2$, $-0 = 10000000_2$

- Range for an 8-bit representation is $[+127_{10}, -128_{10}]$
- Range for an N-bit representation is $[+(2^{N-1}-1)_{10}, -(2^{N-1})_{10}]$

* Slide is courtesy of M. Murdocca and V. Heuring

3-Bit Signed Integer Representations



Decimal	Unsigned	Sign–Mag.	1's Comp.	2's Comp.	Excess 4
7	111	–	–	–	–
6	110	–	–	–	–
5	101	–	–	–	–
4	100	–	–	–	–
3	011	011	011	011	111
2	010	010	010	010	110
1	001	001	001	001	101
+0	000	000	000	000	100
-0	–	100	111	000	100
-1	–	101	110	111	011
-2	–	110	101	110	010
-3	–	111	100	101	001
-4	–	–	–	100	000

* Slide is courtesy of M. Murdocca and V. Heuring

Fixed Point Numbers

- Using only two digits of precision for signed base 10 numbers, the range (interval between lowest and highest) is [-99, +99] and the precision (distance between successive numbers) is 1.
- The maximum error, which is the difference between the value of a real number and the closest representable number, is 1/2 the precision. For this case, the error is $1/2 \times 1 = 0.5$.
- If we choose a = 70, b = 40, and c = -30, then a + (b + c) = 80 (which is correct) but (a + b) + c = -20 which is incorrect. The problem is that (a + b) is +110 for this example, which exceeds the range of +99, and so only the rightmost two digits (+10) are retained in the intermediate result. This is a problem that we need to keep in mind when representing real numbers in a finite representation.

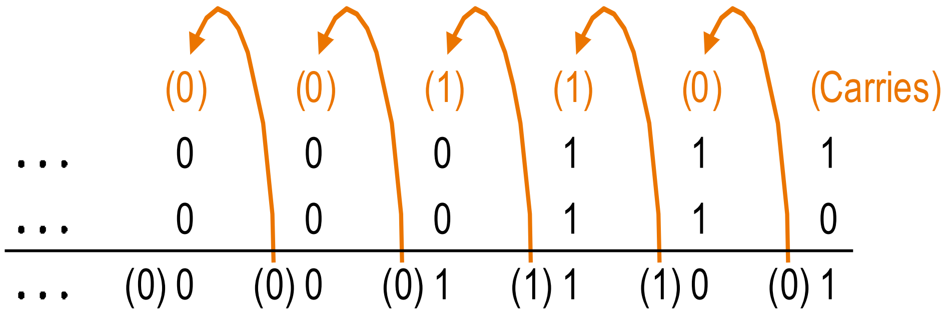
* Slide is courtesy of M. Murdocca and V. Heuring

2's Complement Addition & Subtraction

- Digits are added bit by bit from right to left, with carries passed to the next digit to the left

- *Example:*

```
   0000 0000 0000 0000 0000 0111 =  7
+  0000 0000 0000 0000 0000 0110 =  6
   ---------------------------------------------
   0000 0000 0000 0000 0000 1101 = 13
```



- Subtraction uses addition: the appropriate operand is simply negated

- *Example:*

```
      0000 0000 0000 0000 0000 0000 0000 0111 =  7
-     0000 0000 0000 0000 0000 0000 0000 0110 =  6
     ----------------------------------------------------------------
      0000 0000 0000 0000 0000 0000 0000 0001 =  1
```

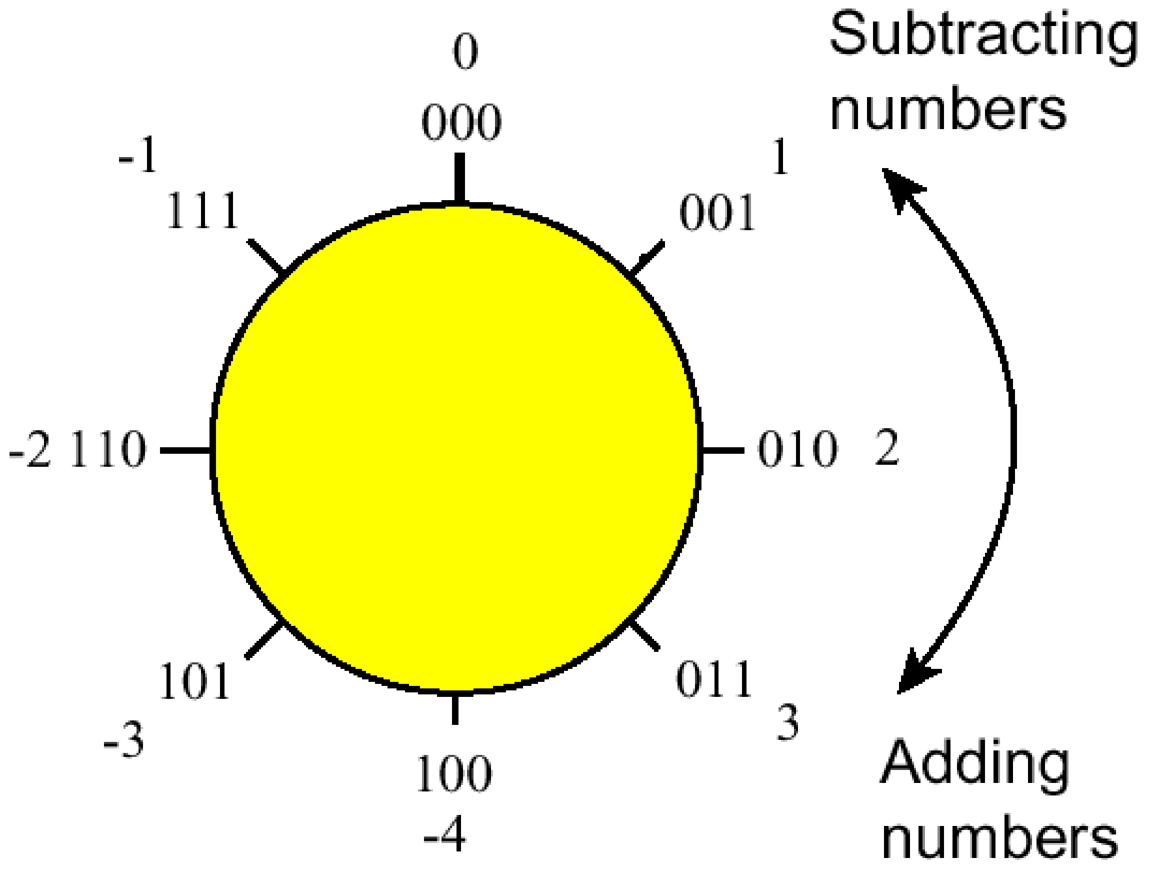
Or using two's complement arithmetic

```
      0000 0000 0000 0000 0000 0000 0000 0111 =   7
+     1111   1111 1111   1111   1111   1111 1111 1010 = - 6
     ----------------------------------------------------------------
      0000 0000 0000 0000 0000 0000 0000 0001 =   1
```

Number Circle for 3-Bit Two's Complement Numbers

• Numbers can be added or subtracted by traversing the number circle clockwise for addition and counterclockwise for subtraction.

• *Overflow* occurs when a transition is made from +3 to -4 while proceeding around the number circle when adding, or from -4 to +3 while subtracting (also called *Underflow*)

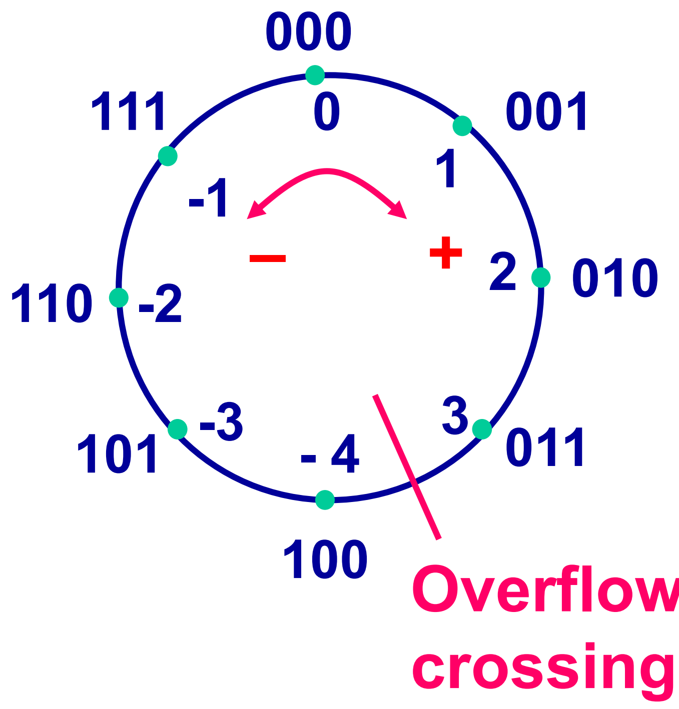


* Slide is courtesy of M. Murdocca and V. Heuring

Overflow: An Error

- Examples: Addition of 3-bit integers (range - 4 to +3)

  - -2-3 = -5
    ```
      110   = -2
    + 101   = -3
    = 1011  =  3 (error)
    ```
  - 3+2 = 5
    ```
      011  =  3
      010  =  2
    = 101  = -3 (error)
    ```



- **Overflow rule: If two numbers with the same sign bit (both positive or both negative) are added, the overflow occurs if and only if the result has the opposite sign.**

Arithmetic Overflow

- Overflow occurs when adding two positive numbers produces a negative result, or when adding two negative numbers produces a positive result. Adding operands of unlike signs never produces an overflow.

- Notice that discarding the carry out of the most significant bit during two's complement addition is a normal occurrence, and does not by itself indicate overflow.

Example: consider adding $(80 + 80 = 160)_{10}$, which produces a result of -96_{10} in an 8-bit two's complement format:

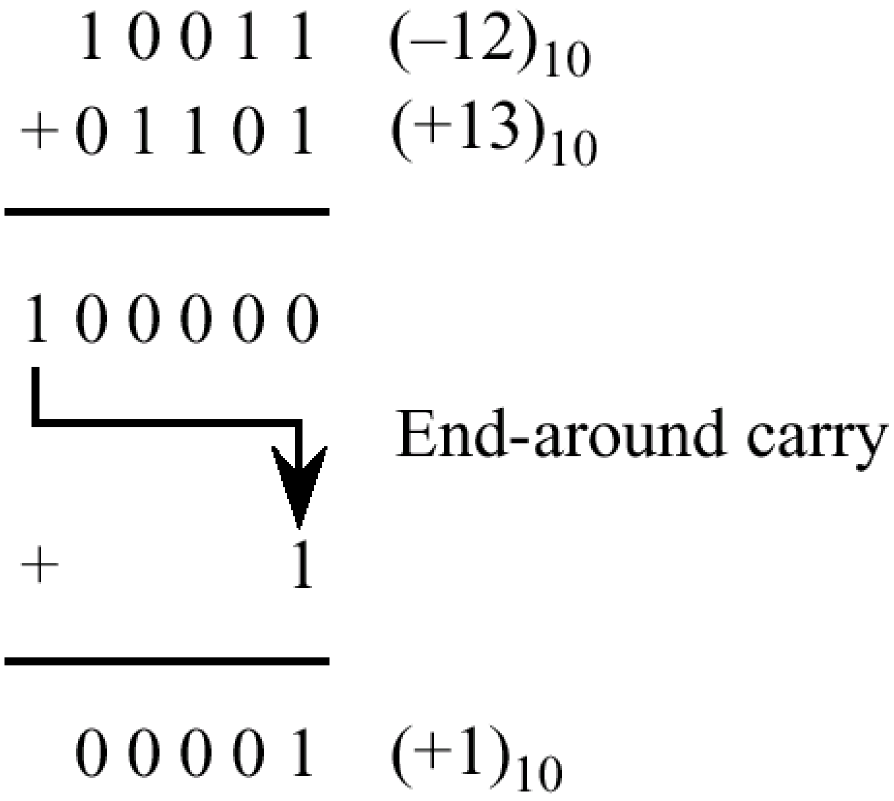
```
  01010000    = 80

+ 01010000    = 80
  --------------
  10100000     = -96 ( not 160 because the sign bit is 1.)
```

* Slide is courtesy of M. Murdocca and V. Heuring

One’s Complement Addition

- An example of one’s complement integer addition with an end-around carry:

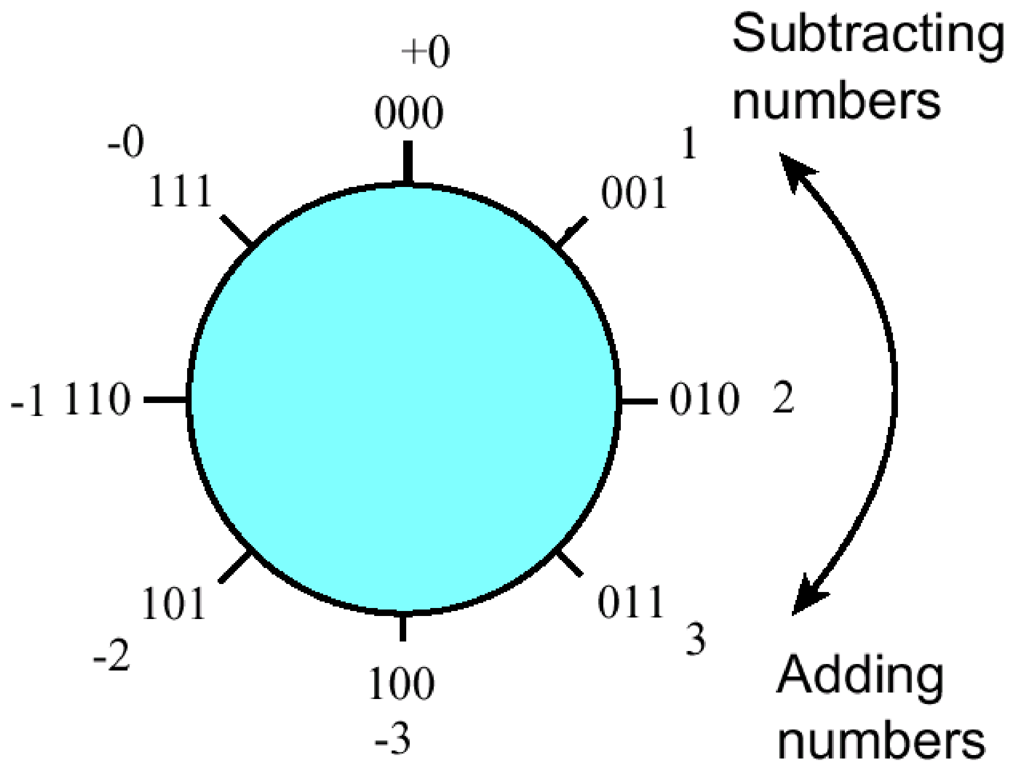


- The end-around carry is needed because there are two representations for 0 in one’s complement. Both representations for 0 are visited when one or both operands are negative.

* Slide is courtesy of M. Murdocca and V. Heuring

Number Circle (1's Complement)

- Number circle for a three-bit signed one's complement representation. Notice the two representations for 0.
- The end-around carry complicates one's complement addition for non-integers, and is generally not used for this situation.



* Slide is courtesy of M. Murdocca and V. Heuring

Dealing with Fractions

- ❑ Simply ignore the binary point and perform addition as if the number has no fraction

$$\xcancel{\begin{array}{rrl} & 0101.1 & (+5.5)_{10} \\ + & 1110.0 & (-1.0)_{10} \\ \hline & 10011.1 & \\ + & \llcorner\!\longrightarrow 1.0 & \\ \hline & 0100.1 & (+4.5)_{10} \end{array}}$$

$$\begin{array}{rrl} & 0101.1 & (+5.5)_{10} \\ + & 1110.1 & (-1.0)_{10} \\ \hline & 10100.0 & \\ + & \llcorner\!\longrightarrow 1 & \\ \hline & 0100.1 & (+4.5)_{10} \end{array}$$

Multiplication of Signed Integers

- ❑ Sign extension to the target size is needed for the negative operand(s).
- ❑ A target size of 8 bits is used here for two 4-bit signed operands, but only a 7-bit target size is needed for the result

$$
\begin{array}{cccccccc l}
 & & & & 1 & 1 & 1 & 1 & (-1)_{10} \\
 & & & \times & 0 & 0 & 0 & 1 & (+1)_{10} \\
\cline{4-8}
 & & & & 1 & 1 & 1 & 1 & \\
 & & & 0 & 0 & 0 & 0 & & \\
 & & 0 & 0 & 0 & 0 & & & \\
 & 0 & 0 & 0 & 0 & & & & \\
\cline{1-8}
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & (+15)_{10}
\end{array}
$$

(Incorrect; result should be –1)

$$
\begin{array}{ccccccccc l}
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & (-1)_{10} \\
\times & & & & & 0 & 0 & 0 & 1 & (+1)_{10} \\
\hline
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \\
 & 0 & 0 & 0 & 0 & 0 & 0 & & & \\
 & 0 & 0 & 0 & 0 & 0 & & & & \\
\hline
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & (-1)_{10}
\end{array}
$$

* Slide is courtesy of M. Murdocca and V. Heuring

Conclusion

- ***Summary***

  ➔ Representation of signed fixed numbers
  (signed magnitude, 1’s complement, 2’s complement, excess)

  ➔ Addition and subtraction of signed binary fixed numbers
  (sign handling, overflow and underflow, effect of the representation)

  ➔ Fixed number representation
  (arithmetic overflow)

- ***Next Lecture***

  ➔ Boolean Algebra

  ➔ Simplification of Boolean expressions

  ➔ Boolean algebra and switching circuits

Reading assignment: section 1.4 in the textbook