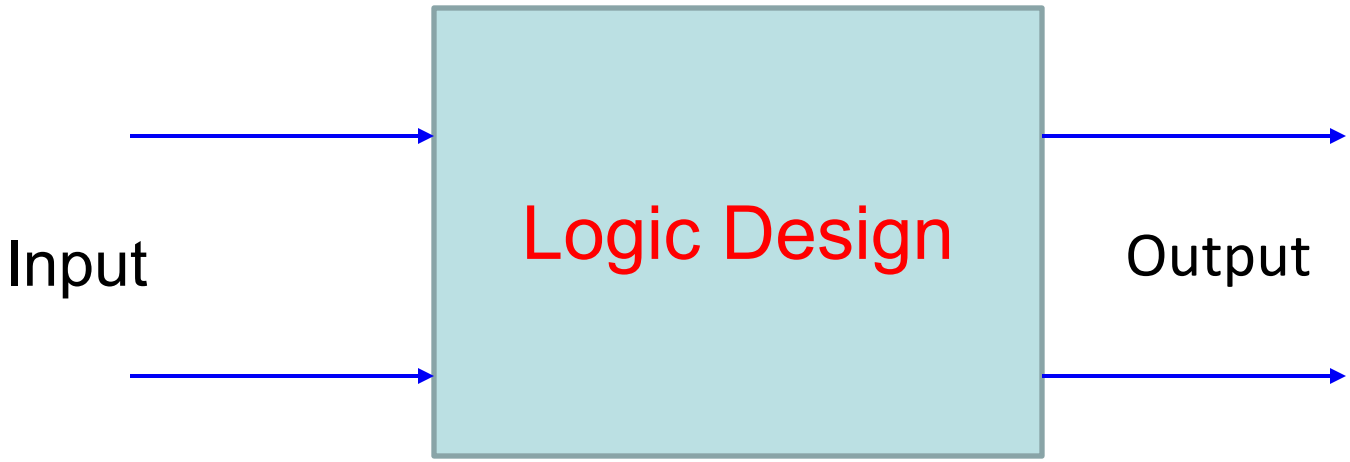
CS-226: Digital Logic Design

Binary Arithmetic

Virendra Singh
Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

Digital System



Why Binary Arithmetic?

3 + 5

= 8

0011 + 0101

= 1000

Binary Arithmetic

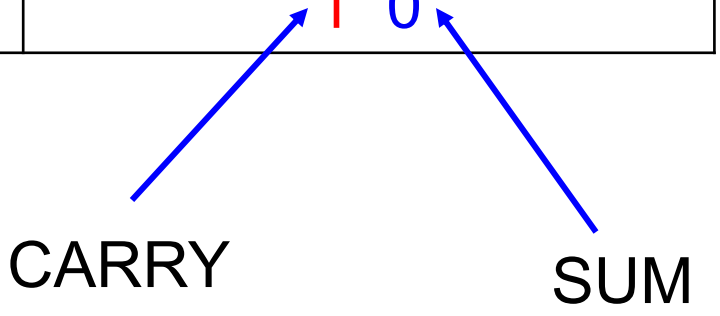
Single Bit Binary Addition

Given two binary digits (X,Y), we get the following sum (S) and carry (C):

X	**0**	**0**	**1**	**1**
+ Y	**+ 0**	**+ 1**	**+ 0**	**+ 1**
C S	**0**	**1**	**1**	**1 0**

Truth Table: Two Bit Adder

X	Y	Binary Sum (C)(S)
0	0	0 0
0	1	0 1
1	0	0 1
1	1	1 0



Truth Tables of Logical Operations

- Truth tables are used to show/define the relationships between the truth values of
  - the individual propositions and
  - the compound propositions based on them

p	q	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \Rightarrow q$	$p \Leftrightarrow q$
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	0
1	1	1	1	0	1	1

Single Bit Binary Addition with Carry

Given two binary digits (X,Y), a carry in (Z) we get the following sum (S) and carry (C):

Carry in (Z) of 0:

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

Carry in (Z) of 1:

Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

Full Adder: Include Carry Input

Z	Y	X	S = X + Y + Z	
			Decimal value	Binary value
0	0	0	0	0 0
0	0	1	1	0 1
0	1	0	1	0 1
0	1	1	2	1 0
1	0	0	1	0 1
1	0	1	2	1 0
1	1	0	2	1 0
1	1	1	3	1 1

CARRY

SUM

Truth Table: Full Adder

Z	Y	X	Binary value (C)(S)
0	0	0	0 0
0	0	1	0 1
0	1	0	0 1
0	1	1	1 0
1	0	0	0 1
1	0	1	1 0
1	1	0	1 0
1	1	1	1 1

Carry

$XY + YZ + ZX$

Sum

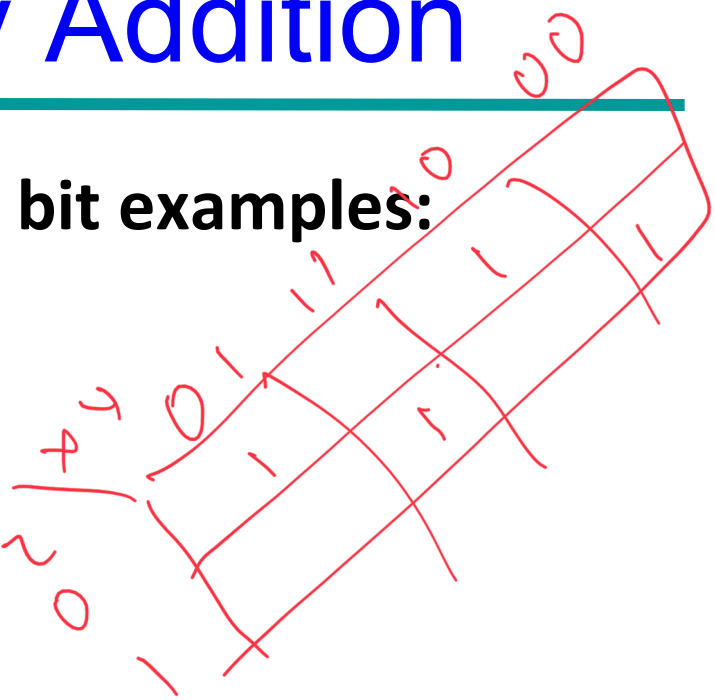
$X \oplus Y \oplus Z$

CARRY

SUM

Multiple Bit Binary Addition

- **Extending this to a multiple bit examples:**



Carries	00000
Augend	01100
Addend	10001
Sum	11101

Single Bit Binary Subtraction

- **Given two binary digits (X,Y), we get the following difference (S) and borrow (B)**

X	0	0	1	1
- Y	- 0	-1	-0	-1
BS	0 0	1 1	0 1	0 0

Truth Table: Two Bit Subtractor

X	Y	Binary Difference (B)(D)
0	0	0 0
0	1	1 1
1	0	0 1
1	1	0 0

BORROW

DIFFERENCE

Single Bit Binary Subtraction with Borrow

- **Given two binary digits (X,Y), a borrow in (Z) we get the following difference (S) and borrow (B):**
- **Borrow in (Z) of 0:**

Z	0	0	0	0
X	0	0	1	1
- Y	-0	-1	-0	-1
BS	0 0	1 1	0 1	0 0

- **Borrow in (Z) of 1:**

Z	1	1	1	1
X	0	0	1	1
- Y	-0	-1	-0	-1
BS	11	1 0	0 0	1 1

Multiple Bit Binary Subtraction

- **Extending this to a multiple bit example:**
- **Notes:**
  - **The 0 is a Borrow-In to the least significant bit.**
  - **If the Subtrahend > the Minuend, interchange and append a – to the result.**

Borrows	00000
Minuend	10110
Subtrahend	10010
Difference	00100

Multiple Bit Binary Subtraction

- Extending this to a multiple bit examples:
- Notes: The 0 is a Borrow-In to the least significant bit. If the Subtrahend > the Minuend, interchange and append a – to the result.

Borrows	00110
Minuend	10110
Subtrahend	10011
Difference	00011

Binary Multiplication

The binary multiplication table is simple:

0 * 0 = 0 | 1 * 0 = 0 | 0 * 1 = 0 | 1 * 1 = 1

Extending multiplication to multiple digits:

Multiplicand	**1011**
Multiplier	**x 101**
Partial Products	**1011**
	0000 -
	1011 - -
Product	**110111**

Signed Numbers

Signed Magnitude?

- Use fixed length binary representation
- Use left-most bit (called *most significant bit* or MSB) for sign:

  0 for positive

  1 for negative

- Example: $+18_{ten} = 00010010_{two}$

  $-18_{ten} = 10010010_{two}$

Difficulties with Signed Magnitude

- Sign and magnitude bits should be differently treated in arithmetic operations.
- Addition and subtraction require different logic circuits.
- Overflow is difficult to detect.
- "Zero" has two representations:

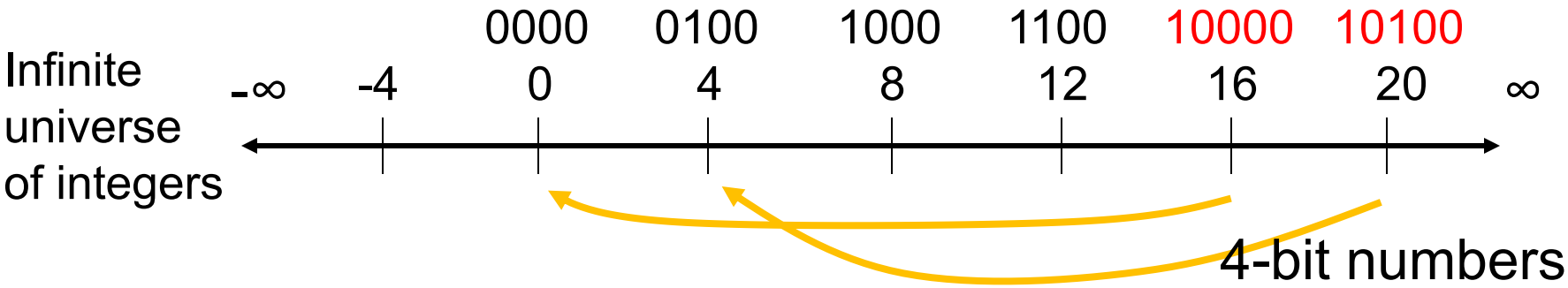
$$+ 0_{ten} = 00000000_{two}$$

$$- 0_{ten} = 10000000_{two}$$

- *Signed-integers are not used in modern computers.*

Problems with Finite Math

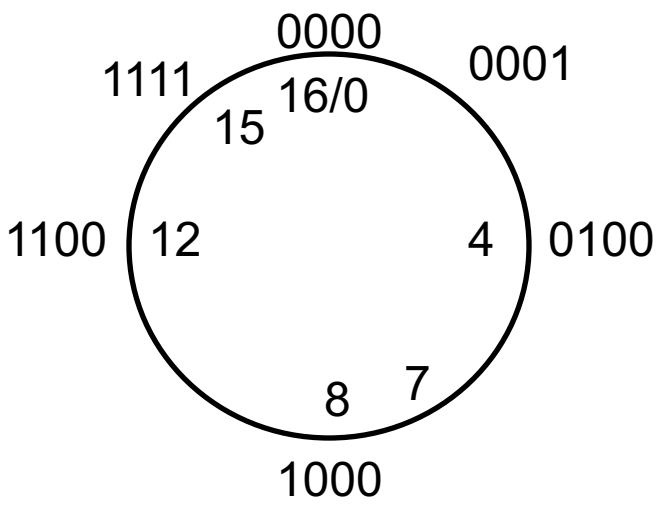
- Finite size of representation:
  - Digital circuit cannot be arbitrarily large.
  - Overflow detection – easy to determine when the number becomes too large.

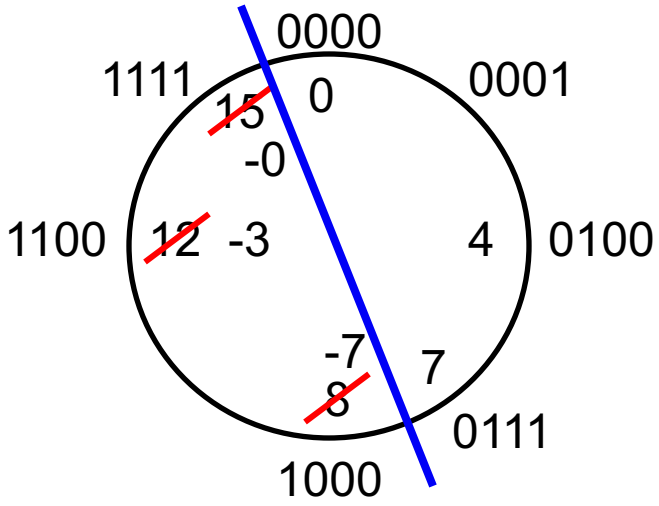


- Represent negative numbers:
  - Unique representation of 0.

4-bit Universe

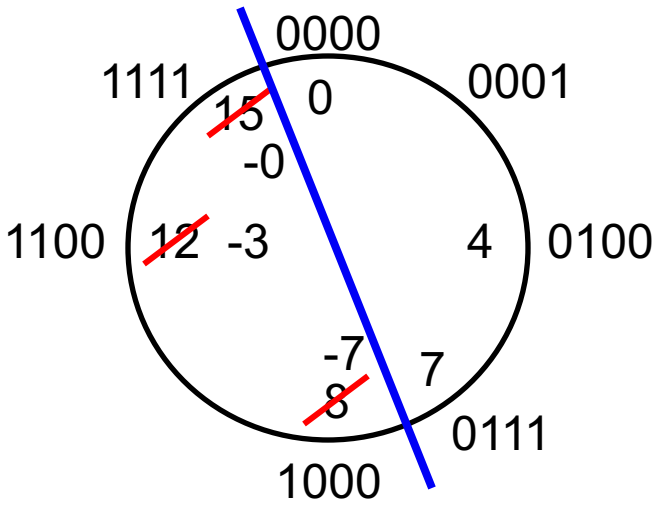
Modulo-16 (4-bit) universe





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

One Way to Divide Universe
1's Complement Numbers

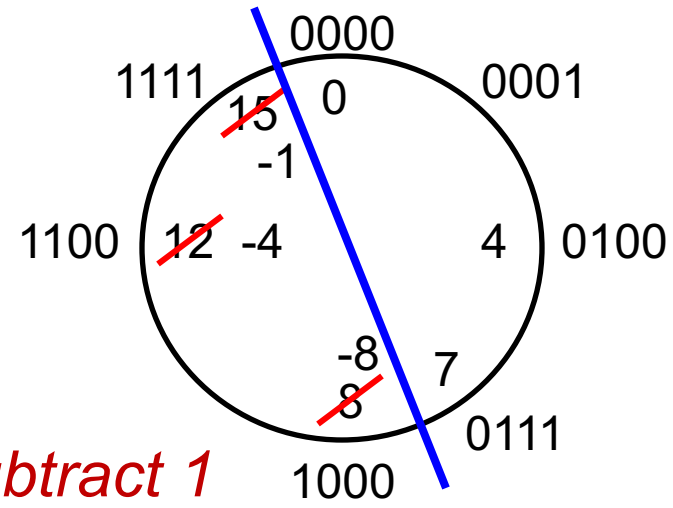


Negation rule: invert bits.

Problem: 0 ≠ – 0

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

Another Way to Divide Universe 2's Complement Numbers



Subtract 1 on this side

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

Integers With Sign – Two Ways

- Use fixed-length representation, but no explicit sign bit:
  - 1's complement: To form a negative number, complement each bit in the given number.
  - 2's complement: To form a negative number, start with the given number, subtract one, and then complement each bit, *or*

    first complement each bit, and then add 1.
- 2's complement is the preferred representation.

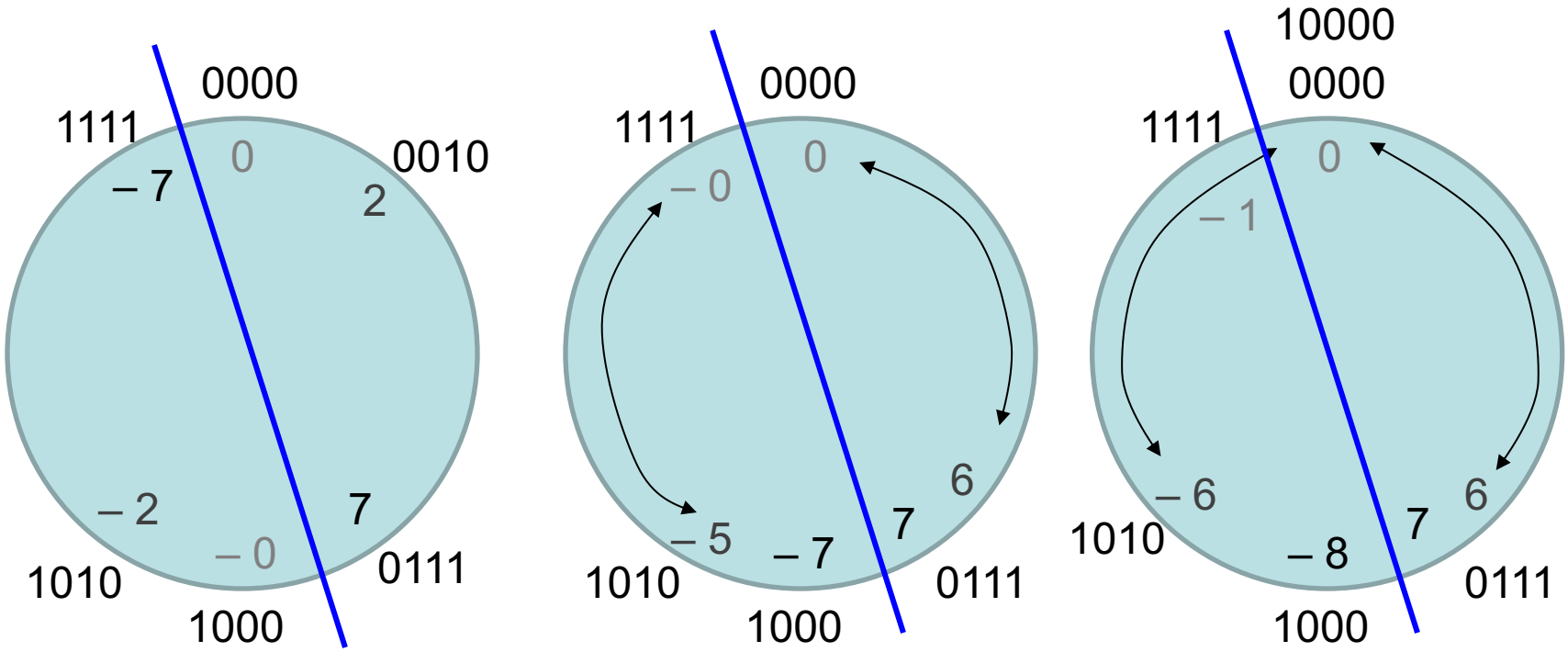
2's-Complement Integers

- Why not 1's-complement? *Don't like two zeros.*
- Negation rule:
  - Subtract 1 and then invert bits, or
  - Invert bits and add 1
- Some properties:
  - Only one representation for 0
  - Exactly as many positive numbers as negative numbers
  - Slight asymmetry – there is one negative number with no positive counterpart

General Method for Binary Integers with Sign

- Select number (n) of bits in representation.
- Partition 2^n integers into two sets:
  - 00…0 through 01…1 are $2^n/2$ positive integers.
  - 10…0 through 11…1 are $2^n/2$ negative integers.
- Negation rule transforms negative to positive, and vice-versa:
  - Signed magnitude: invert MSB (most significant bit)
  - 1's complement: Subtract from $2^n - 1$ or 1…1 (same as "inverting all bits")
  - 2's complement: Subtract from 2^n or 10…0 (same as 1's complement + 1)

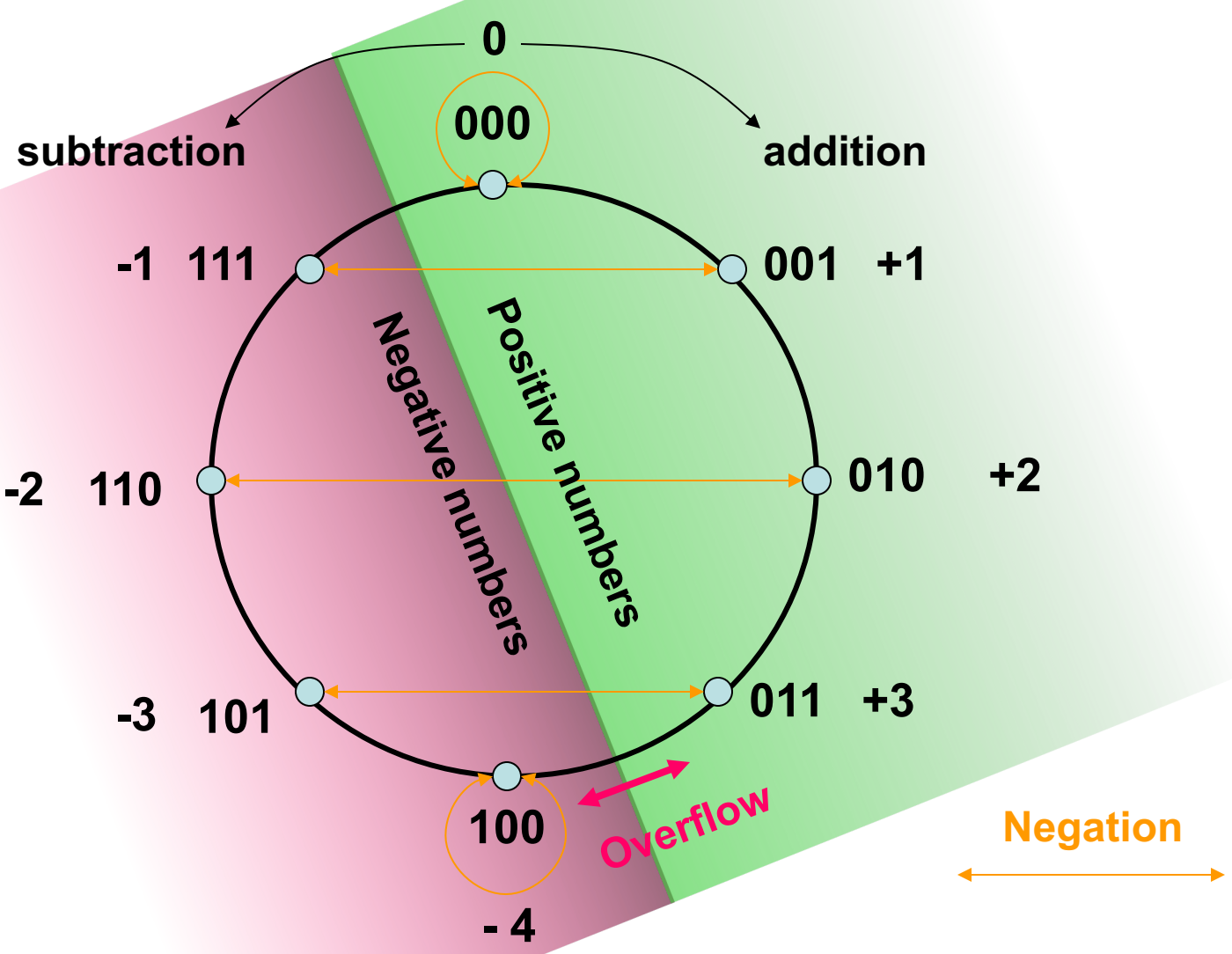
Three Systems (n = 4)



Three Representations

Sign-magnitude	1's complement	2's complement
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = - 0	100 = - 3	100 = - 4
101 = - 1	101 = - 2	101 = - 3
110 = - 2	110 = - 1	110 = - 2
111 = - 3	111 = - 0	111 = - 1
		(Preferred)

2's Complement Numbers (n = 3)



Summary

- For a given number (n) of digits we have a finite set of integers. For example, there are 10^3 = 1,000 decimal integers and 2^3 = 8 binary integers in 3-digit representations.
- We divide the finite set of integers [0, r^n – 1], where radix r = 10 or 2, into two equal parts representing positive and negative numbers.
- Positive and negative numbers of equal magnitudes are complements of each other: x + complement (x) = 0.

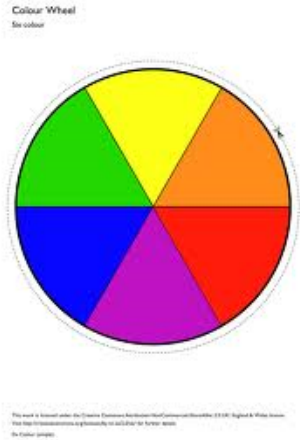
Summary: Defining Complement

- Decimal integers:
  - 10's complement: $-x = \text{Complement}(x) = 10^n - x$
  - 9's complement: $-x = \text{Complement}(x) = 10^n - 1 - x$
  - For 9's complement, subtract each digit from 9
  - For 10's complement, add 1 to 9's complement
- Binary integers:
  - 2's complement: $-x = \text{Complement}(x) = 2^n - x$
  - 1's complement: $-x = \text{Complement}(x) = 2^n - 1 - x$
  - For 1's complement, subtract each digit from 1
  - For 2's complement, add 1 to 1's complement

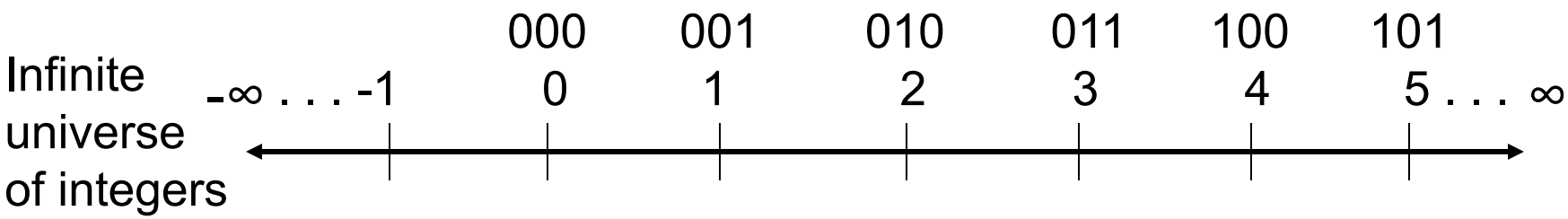
Understanding Complement

- Complement means “something that completes”:

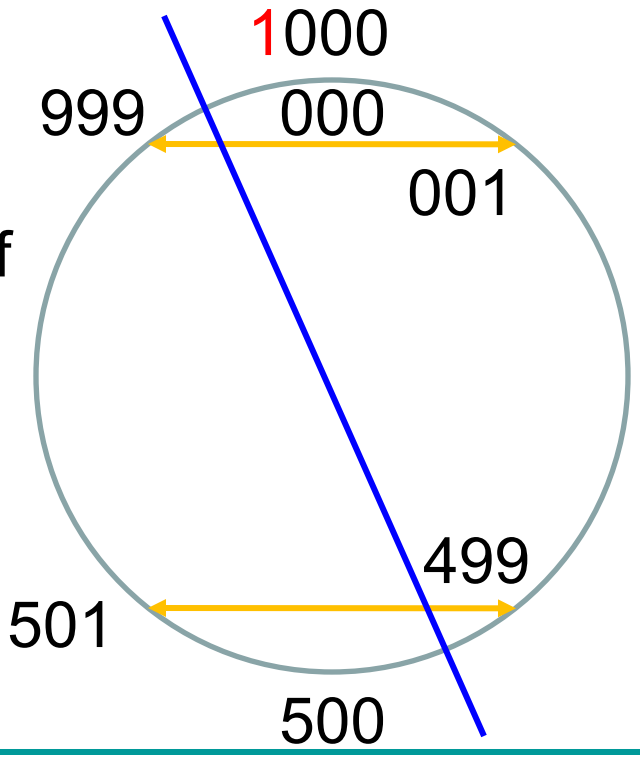
  e.g., X + complement (X) = “Whole”.
- Complement also means “opposite”, e.g., complementary colors are placed opposite in the primary color chart.
- Complementary numbers are like electric charges. Positive and negative charges of equal magnitudes annihilate each other.

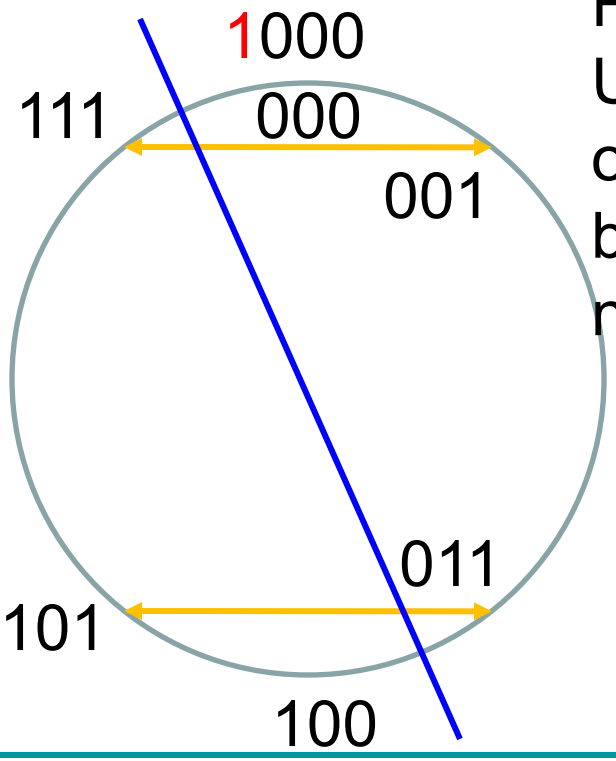


2's-Complement Numbers



Finite Universe of 3-digit Decimal numbers





Finite Universe of 3-bit binary numbers

2's-Complement to Decimal Conversion

$$b_{n-1}\, b_{n-2} \ldots b_1\, b_0 = -2^{n-1} b_{n-1} + \sum_{i=0}^{n-2} 2^i\, b_i$$

8-bit conversion box

-128	64	32	16	8	4	2	1

Example

-128	64	32	16	8	4	2	1
1	1	1	1	1	1	0	1

– 128 + 64 + 32 + 16 + 8 + 4 + 1 = – 128 + 125 = – 3

Thank You