Logic:Expression

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

CS-230: Digital Logic Design & Computer Architecture

Lecture 3 (10 January 2022)

CADSL

Signed Number System

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

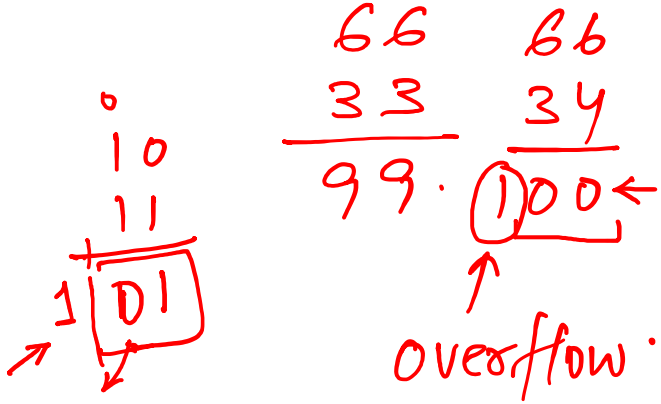
  $-18_{ten} = 10010010_{two}$

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:

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

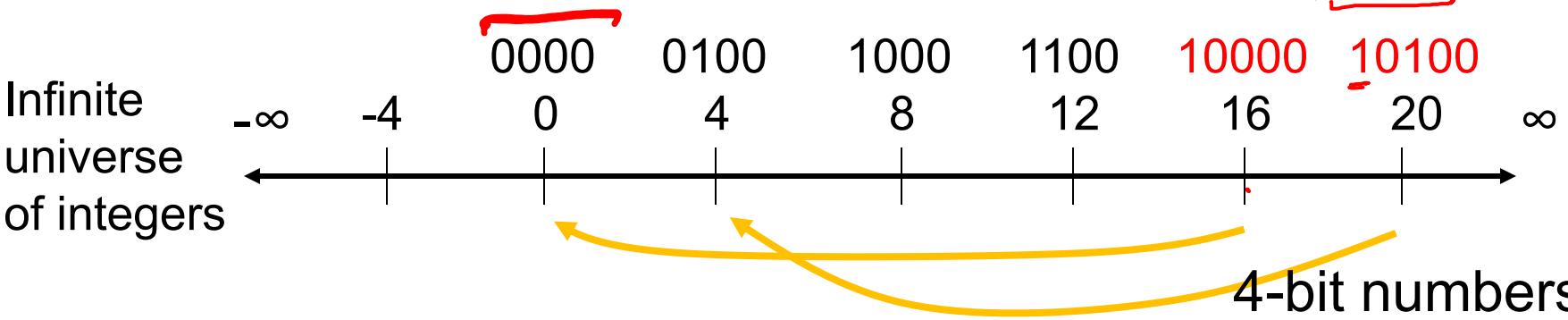
$$- 0_{ten} = 10000000_{two}$$



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

Problems with Finite Math

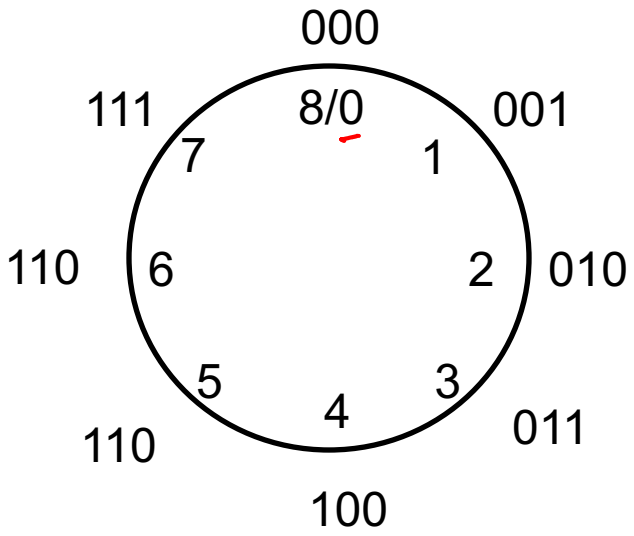
- Finite size of representation:
  - Digital circuit cannot be arbitrarily large.
  - Overflow detection – easy to determine when the number becomes too large.

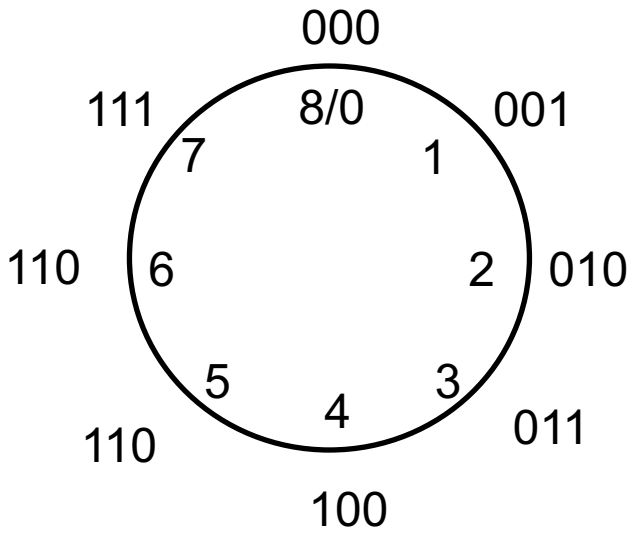


- Represent negative numbers:
  - Unique representation of 0.

3-bit Universe

Modulo-8 (3-bit) universe

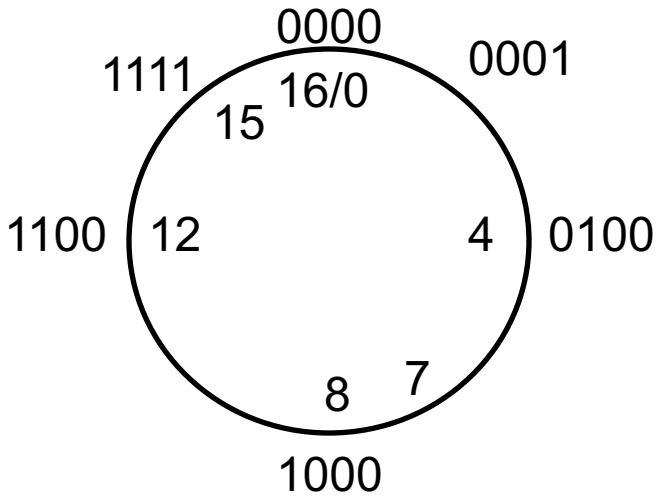


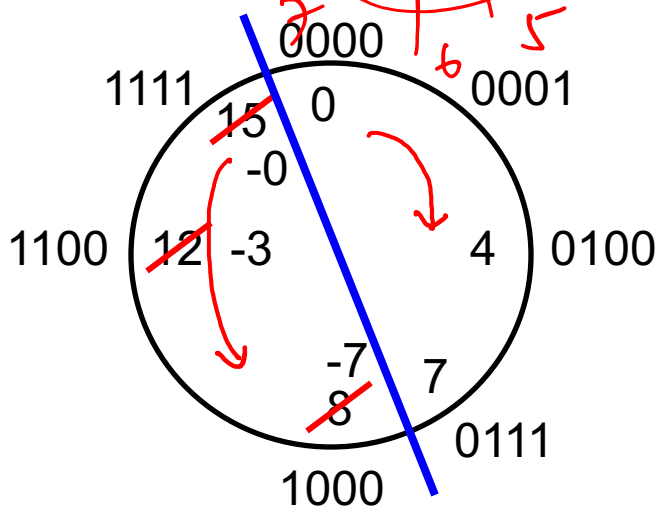


Only 8 integers: 0 through 7, or – 3 through 3

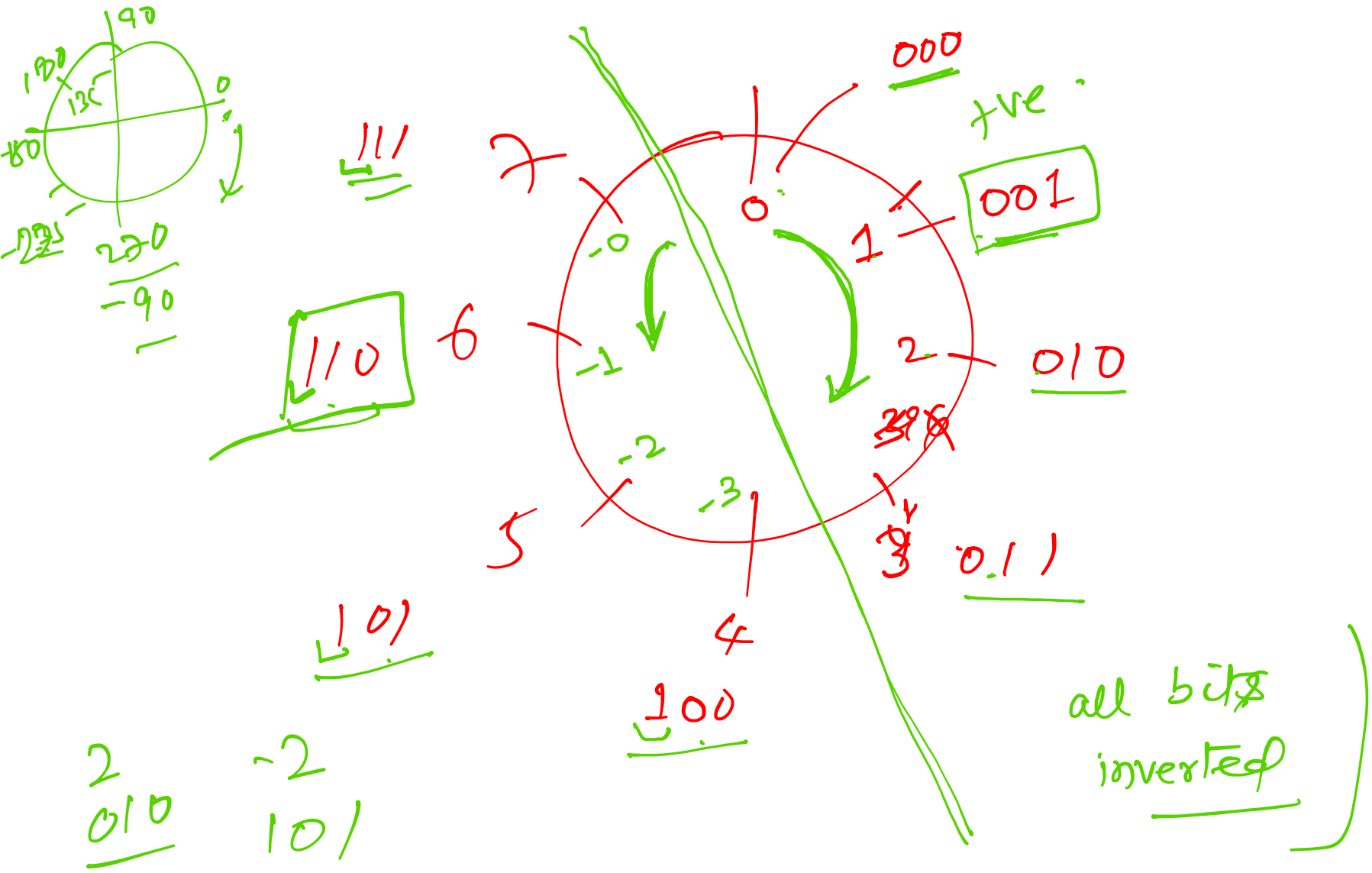
4-bit Universe

Modulo-16 (4-bit) universe





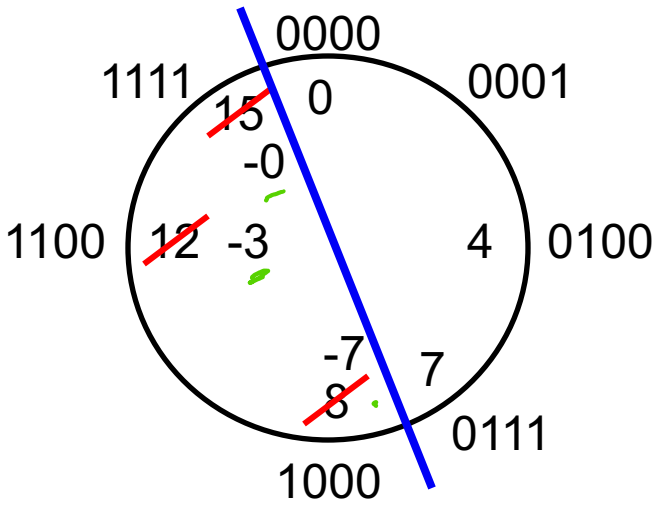
Only 16 integers: 0 through 15, or – 7 through 7


90
180
135
0
-45
-135
-90
111
7
000
+ve
001
0
1
-0
110
6
-1
2
010
-2
5
-3
3
011
101
4
100
all bits inverted
2
010
-2
101

$0 \Rightarrow 1$

$1 \Rightarrow 0$

One Way to Divide Universe
1's Complement Numbers

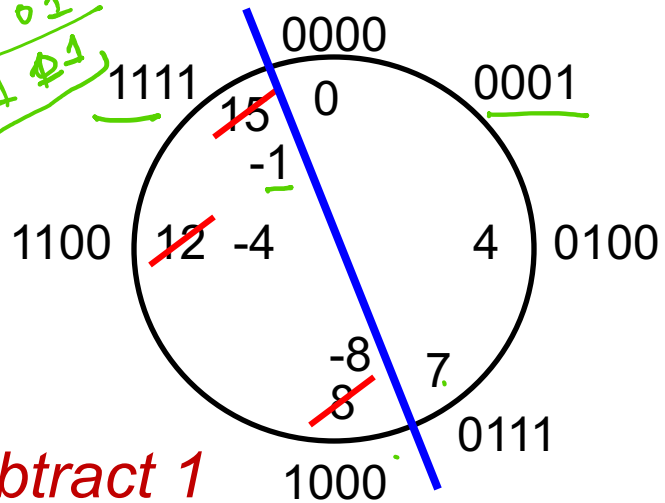


Negation rule: invert bits.

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.

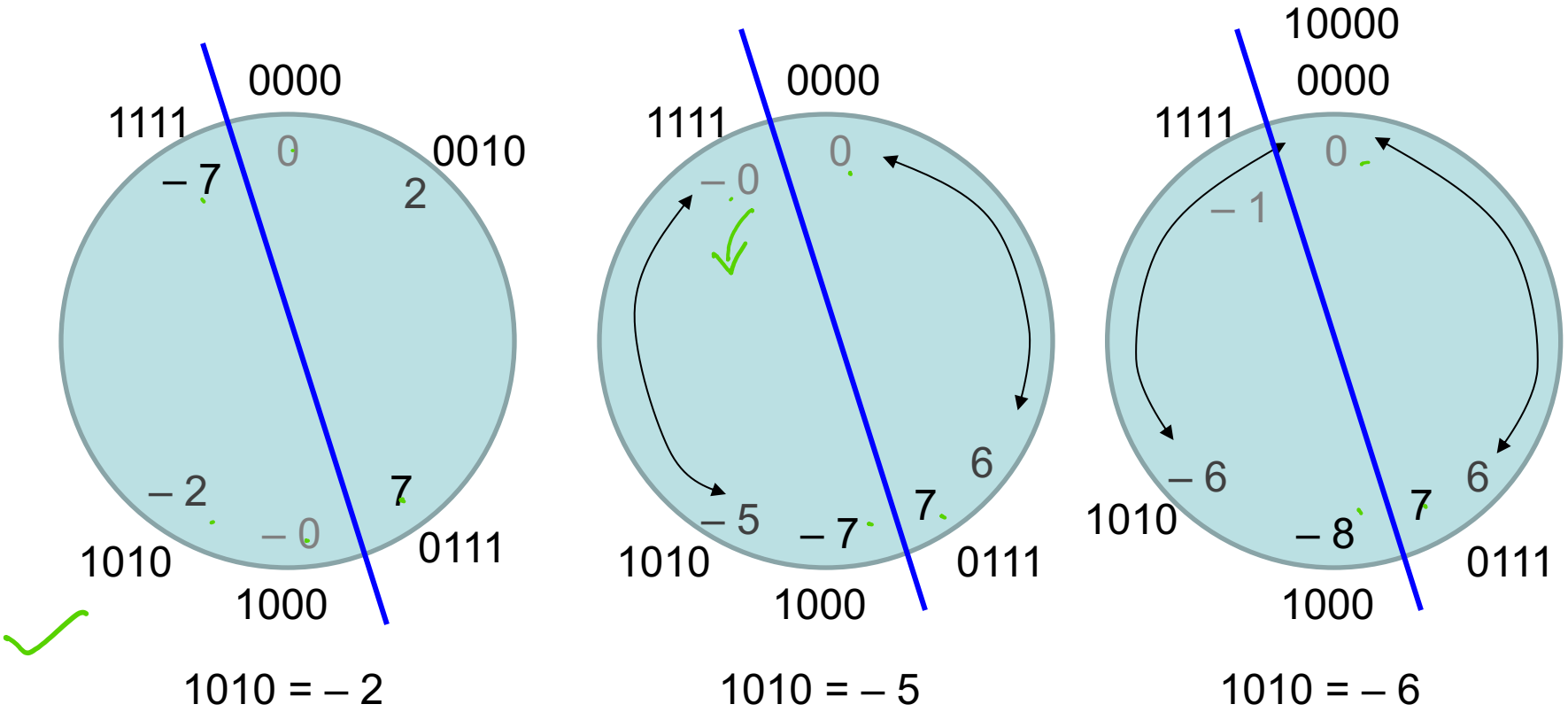
2's-Complement Integers

- 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)

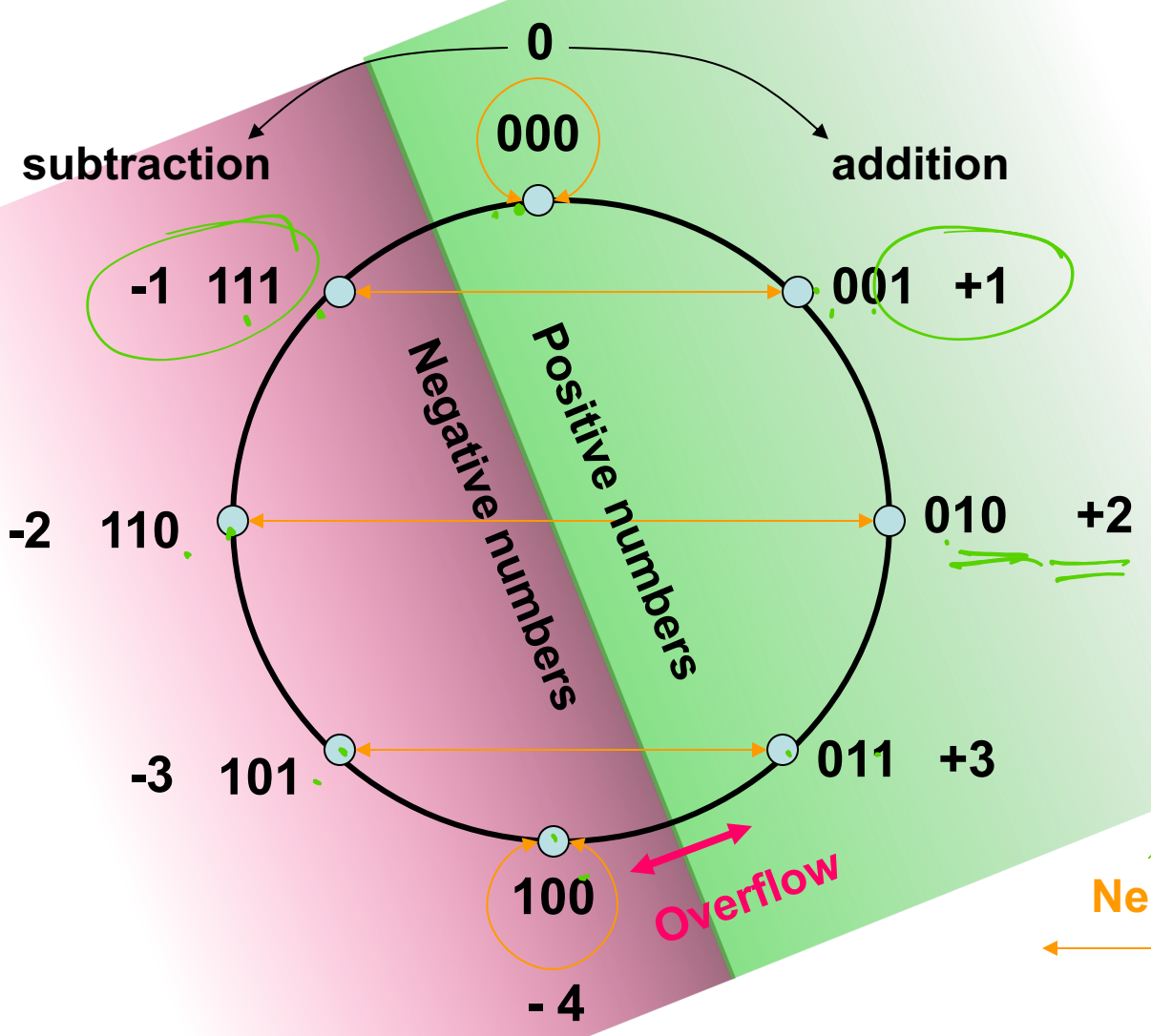


Signed magnitude | 1's complement integers | 2's complement integers

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)

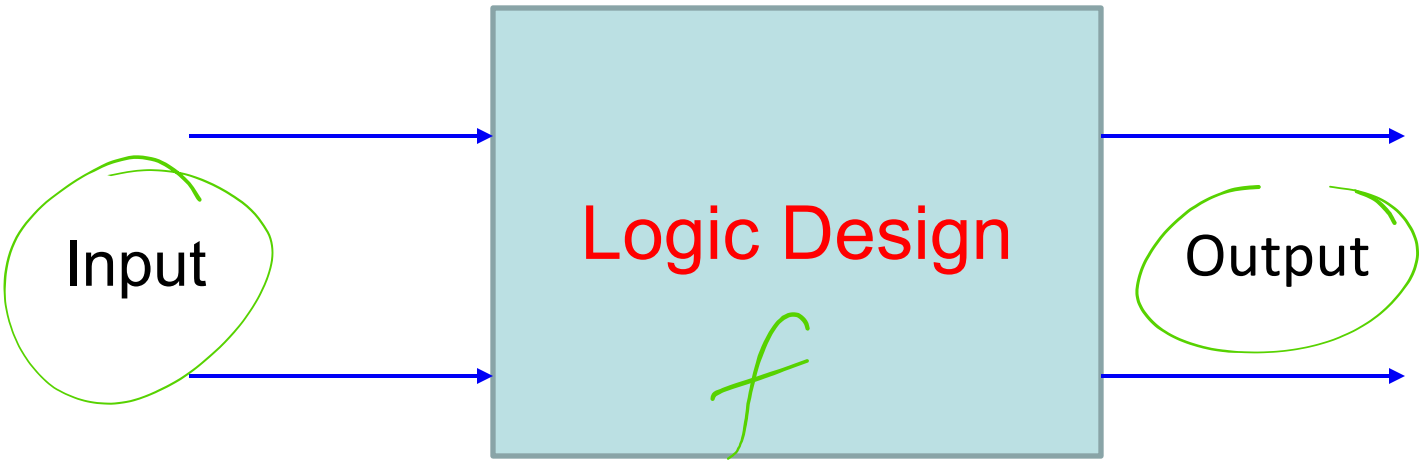
2's Complement Numbers (n = 3)



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.

Digital System

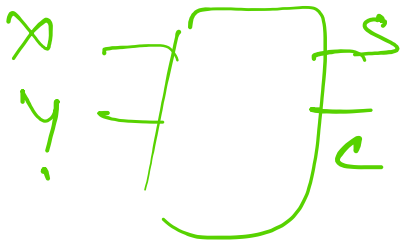


output = f(input) {0,1}

I/O Behaviour: Automobile Ignition

- Engine turns on when
- Ignition key is applied AND
  - either Car is in parking gear OR
  - Brake pedal is on
- AND
  - either Seat belt is fastened OR
  - Car is in parking gear

Truth Table: Half Adder



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

CARRY

SUM

Truth Table: Full Adder

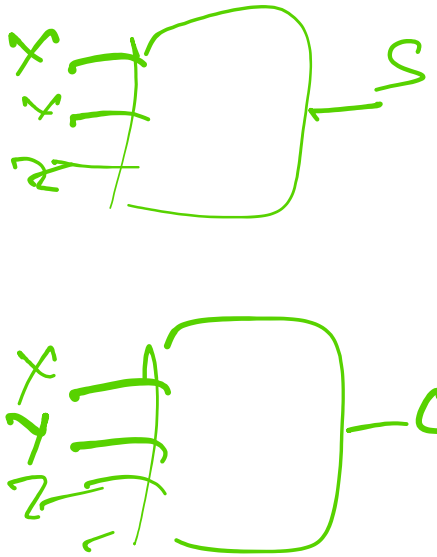
Z	Y	X	Binary value (C)(S)
0	0	0	0 0
0	0	1	0 1
0	1	0	0 1
0	1	1	1 0
1	0	0	0 1
1	0	1	1 0
1	1	0	1 0
1	1	1	1 1

CARRY

SUM

Truth Table: Full Adder

Z	Y	X	Carry C	Sum 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



Truth Tables of logical functions

- 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	$p.q$	$P+q$	$p \oplus q$	$p \Rightarrow q$	$p \Leftrightarrow q$
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

Logic Expressions

Truth Table

X Y Z	F
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

Logic Expression

$$F = \bar{X}.\bar{Y}.Z + X.\bar{Y}.\bar{Z} + X.\bar{Y}.Z$$

$$X.Y.\bar{Z} + X.Y.Z$$

- Logic expressions, truth tables describe the same function!
- Truth tables are unique; expressions are not. This gives flexibility in implementing functions.

Thank You