CS 223: Computer Architecture & Organization

Binary Number System and Information Representation

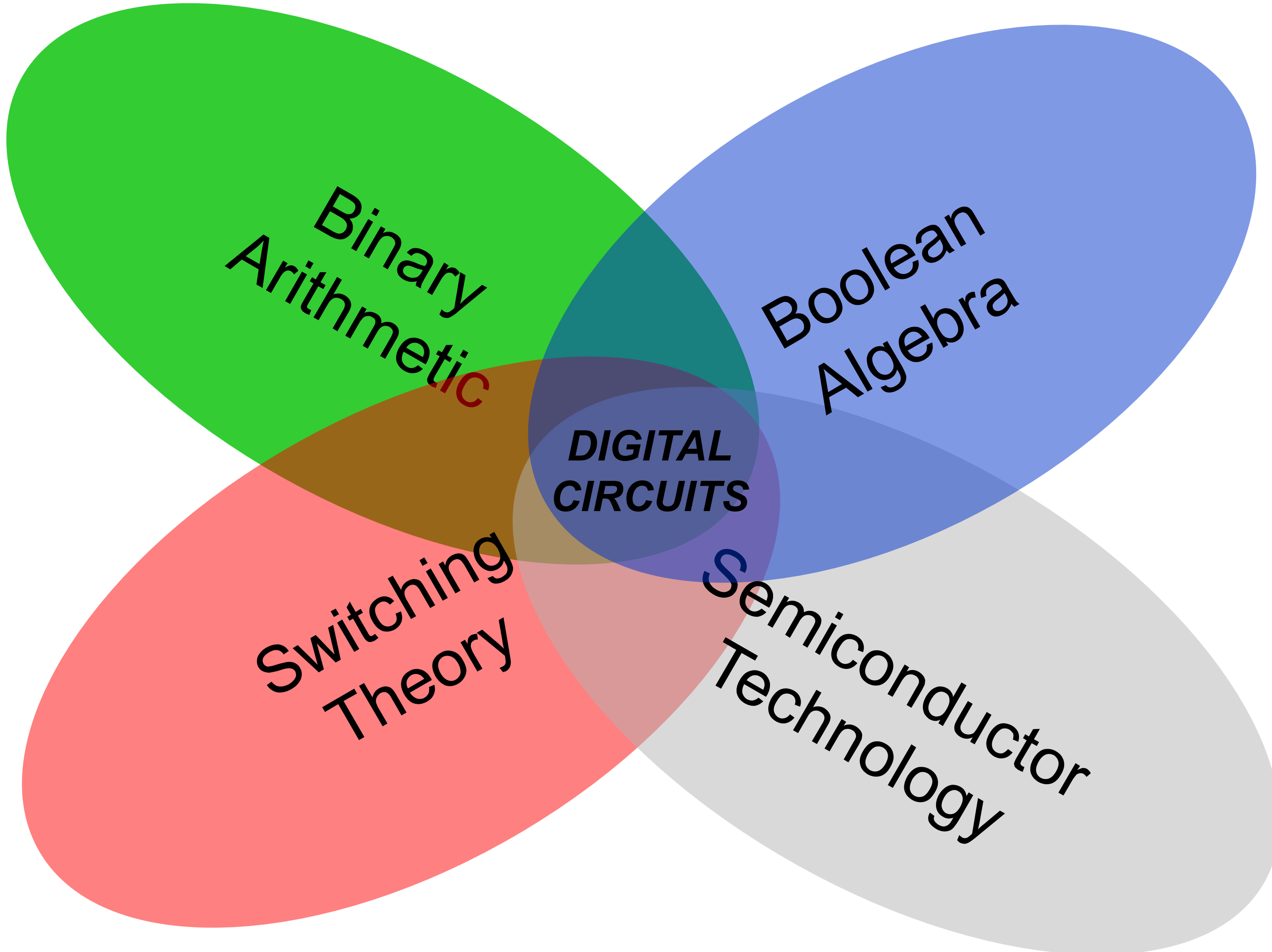
J. K. Deka

Professor

Department of Computer Science & Engineering

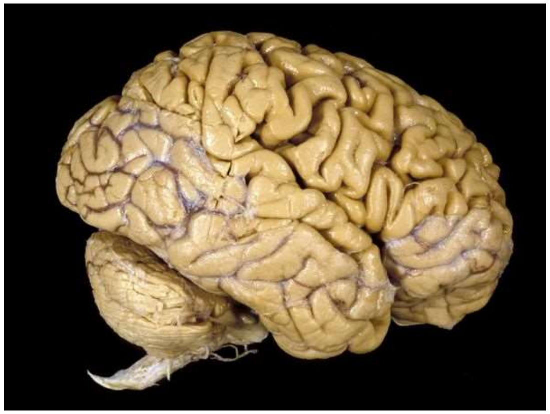
Indian Institute of Technology Guwahati, Assam.

Digital Systems



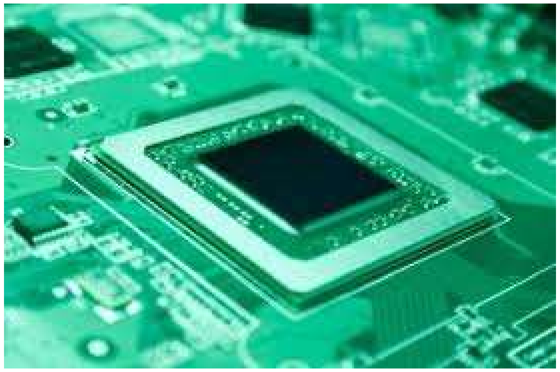
Why Binary Arithmetic?

3 + 5



= 8

0011 + 0101



= 1000

Why Binary Arithmetic?

- Hardware can only deal with binary digits, 0 and 1.
- Must represent all numbers, integers or floating point, positive or negative, by **bi**nary digi**ts**, called bits.
- Can devise electronic circuits to perform arithmetic operations: add, subtract, multiply and divide, on binary numbers.

Positive Integers

- Decimal system: made of 10 digits, {0,1,2, . . . , 9}

  $41 = 4\times10^1 + 1\times10^0$

  $255 = 2\times10^2 + 5\times10^1 + 5\times10^0$

- Binary system: made of two digits, {0,1}

  $00101001 = 0\times2^7 + 0\times2^6 + 1\times2^5 + 0\times2^4$

  $+1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$

  $= 32 + 8 + 1 = 41$

- 11111111= 255, largest number with 8 binary digits, 2^8-1
- LSB and MSB

Base or Radix

- For decimal system, 10 is called the base or radix.
- Decimal 41 is also written as 41_{10} or 41_{ten}
- Base (radix) for binary system is 2.
  - $41_{ten} = 101001_2$ or 101001_{two}

    $111_{ten} = 1101111_{two}$

    $111_{two} = 7_{ten}$

Base or Radix

- For Hexadecimal system, 16 is the base or radix.
  - Needs 16 symbols: 0, 1, …..,9, A, B, C, D, E, F

$$111_{ten} = 01101111_{two} = 6F_{16} = 6*16^1 + 15*16^0$$

Number Systems

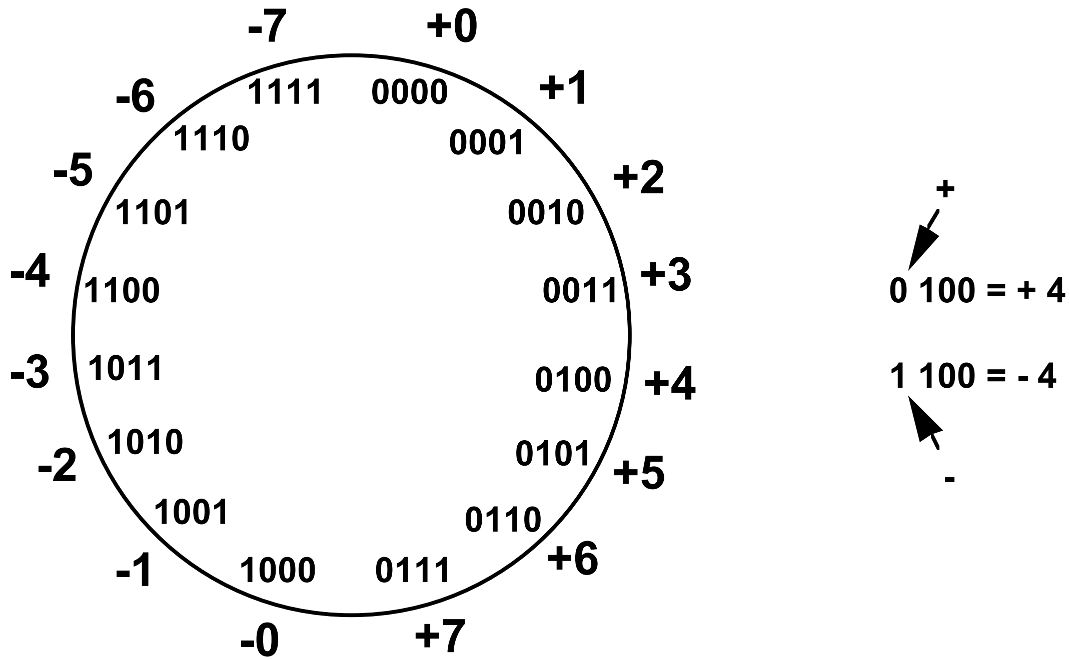
- Representation of positive numbers same in most systems
  - What about negative numbers?
- Major differences are in how negative numbers are represented
- Three major schemes:
  - sign and magnitude
  - ones complement
  - twos complement

Sign and Magnitude Representation

- ❖ Use fixed length binary representation
- ❖ Use left-most bit (called most significant bit or MSB) for sign:
  - ❖ 0 for positive
  - ❖ 1 for negative
- ❖ Example: $+18_{ten} = \textcolor{red}{0}0010010_{two}$

  $-18_{ten} = \textcolor{red}{1}0010010_{two}$

Sign and Magnitude Representation



❖High order bit is sign: 0 = positive (or zero), 1 = negative

❖Three low order bits is the magnitude: 0 (000) thru 7 (111)

❖Number range for n bits = +/-2^{n-1}-1

❖Two representations for 0

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.

Addition and Subtraction of Numbers

Sign and Magnitude Form

result sign bit is the same as the operands' sign

4	0100	-4	1100
+ 3	0011	+ (-3)	1011
7	0111	-7	1111

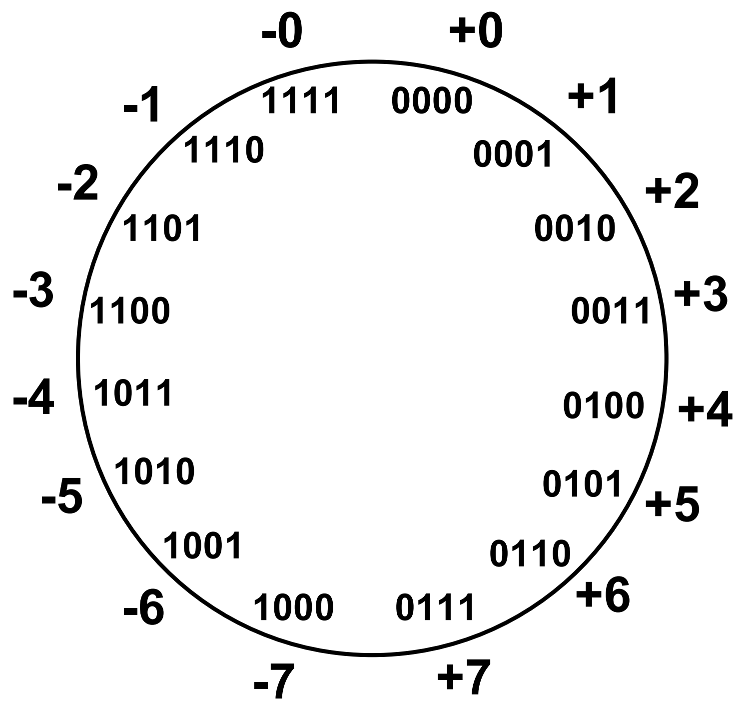
when signs differ, operation is subtract, sign of result depends on sign of number with the larger magnitude

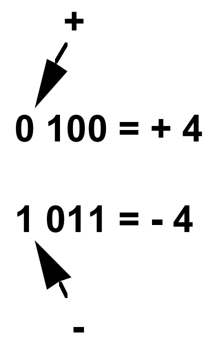
4	0100	-4	1100
- 3	1011	+ 3	0011
1	0001	-1	1001

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.

Ones Complement

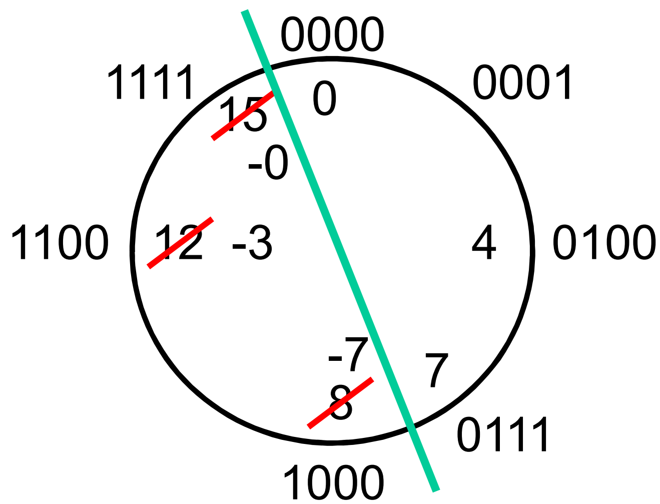




❖Subtraction implemented by addition & 1's complement

❖Still two representations of 0! This causes some problems

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

Addition and Subtraction of Numbers

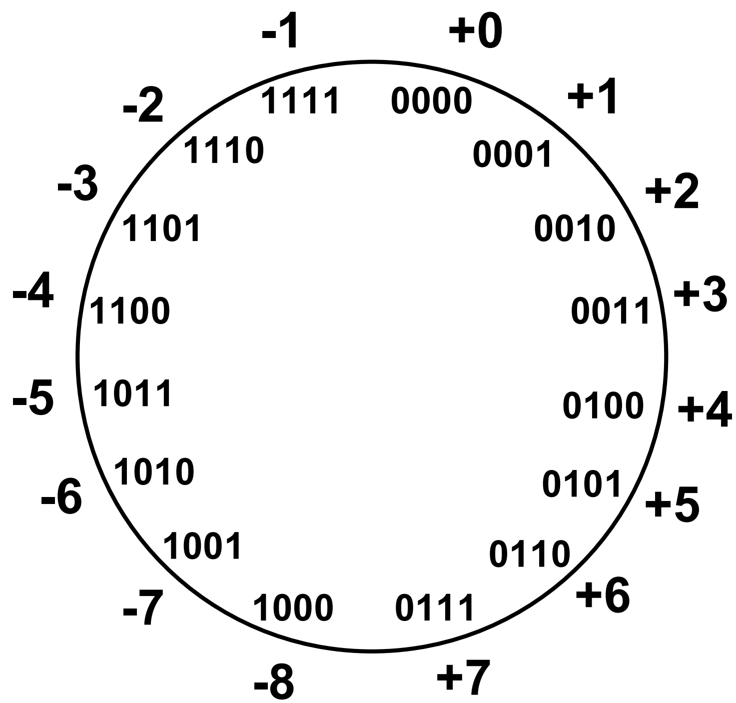
Ones Complement Calculations

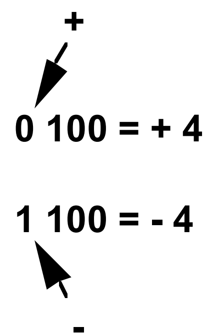
4	0100	-4	1011
+ 3	0011	+ (-3)	1100
7	0111	-7	10111
		End around carry	1
			1000

	4	0100	-4	1011
	- 3	1100	+ 3	0011
	1	10000	-1	1110
End around carry		1		
		0001		

Twos Complement

like 1's comp except shifted one position clockwise

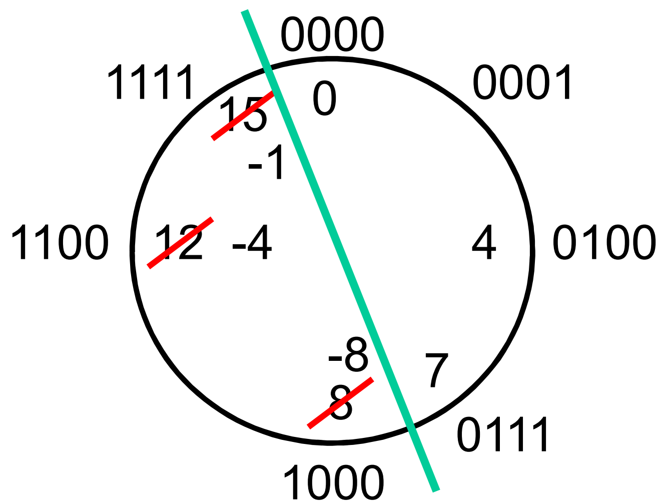




❖Only one representation for 0

❖One more negative number than positive number

2's Complement Numbers



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

Negation rule: invert bits and add 1

2's Complement Numbers

$N^* = 2^n - N$

Example: Twos complement of 7

$$\begin{array}{rl} 2^4 = & 10000 \\ \text{sub } 7 = & \underline{\ 0111} \\ & \ 1001 \ = \text{repr. of } -7 \end{array}$$

Example: Twos complement of -7

$$\begin{array}{rl} 2^4 = & 10000 \\ \text{sub } -7 = & \underline{\ 1001} \\ & \ 0111 \ = \text{repr. of } 7 \end{array}$$

Shortcut method:

Twos complement = bitwise complement + 1

0111 -> 1000 + 1 -> 1001 (representation of -7)

1001 -> 0110 + 1 -> 0111 (representation of 7)

Addition and Subtraction of Numbers

Twos Complement Calculations

If carry-in to sign = carry-out then ignore carry

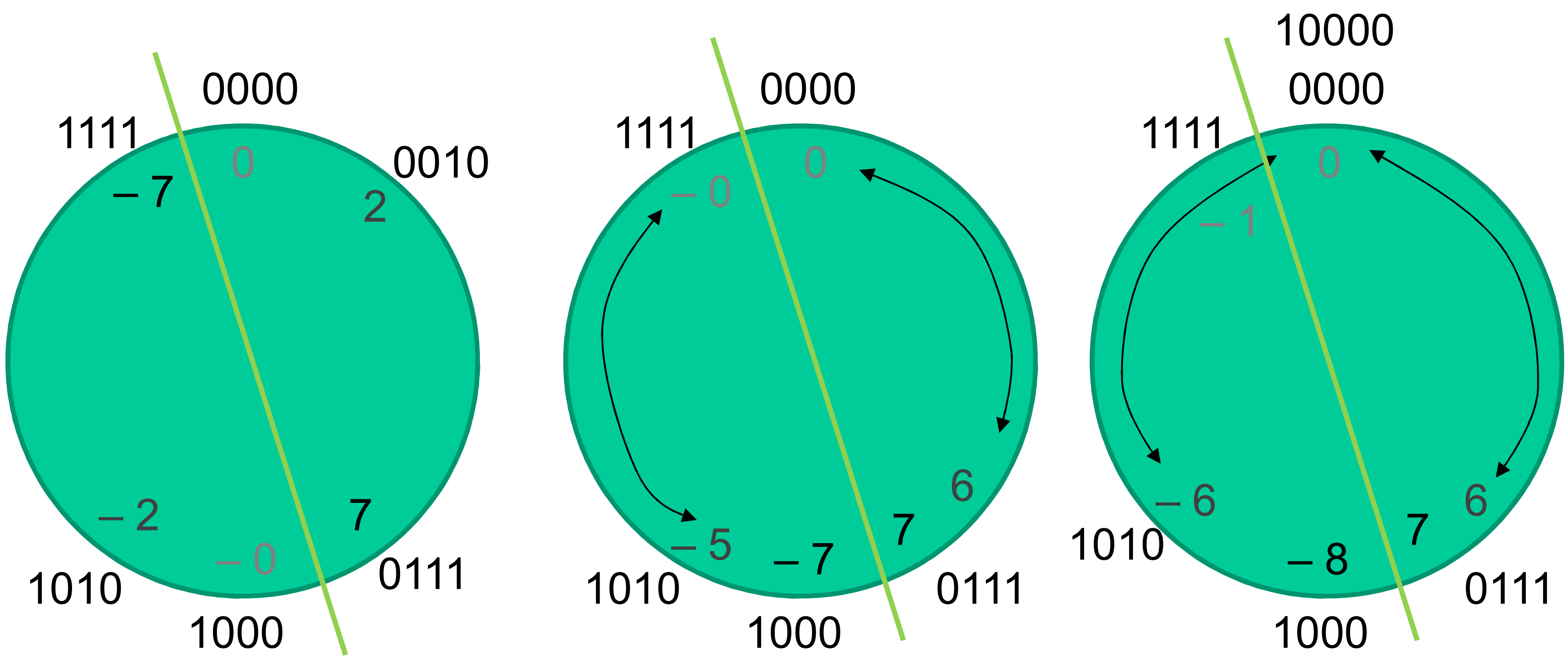
4	0100	-4	1100
+ 3	0011	+ (-3)	1101
7	0111	-7	11001

if carry-in differs from carry-out then overflow

4	0100	-4	1100
- 3	1101	+ 3	0011
1	10001	-1	1111

Simpler addition scheme makes twos complement the most common choice for integer number systems within digital systems

Three Systems (n = 4)



1010 = – 2 1010 = – 5 1010 = – 6

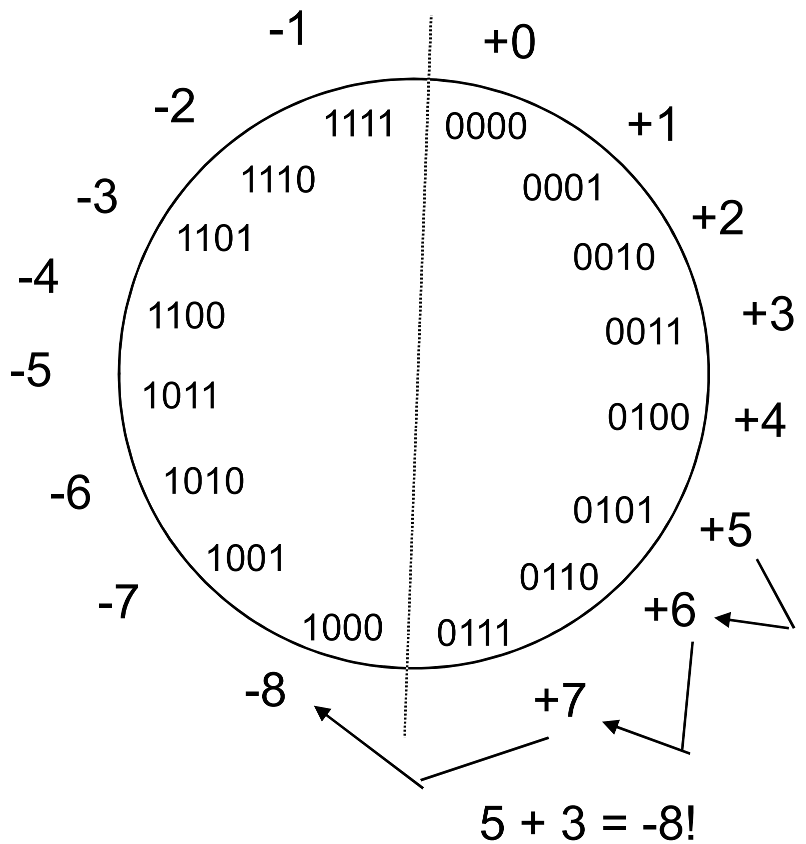
Signed magnitude 1's complement integers 2's complement integers

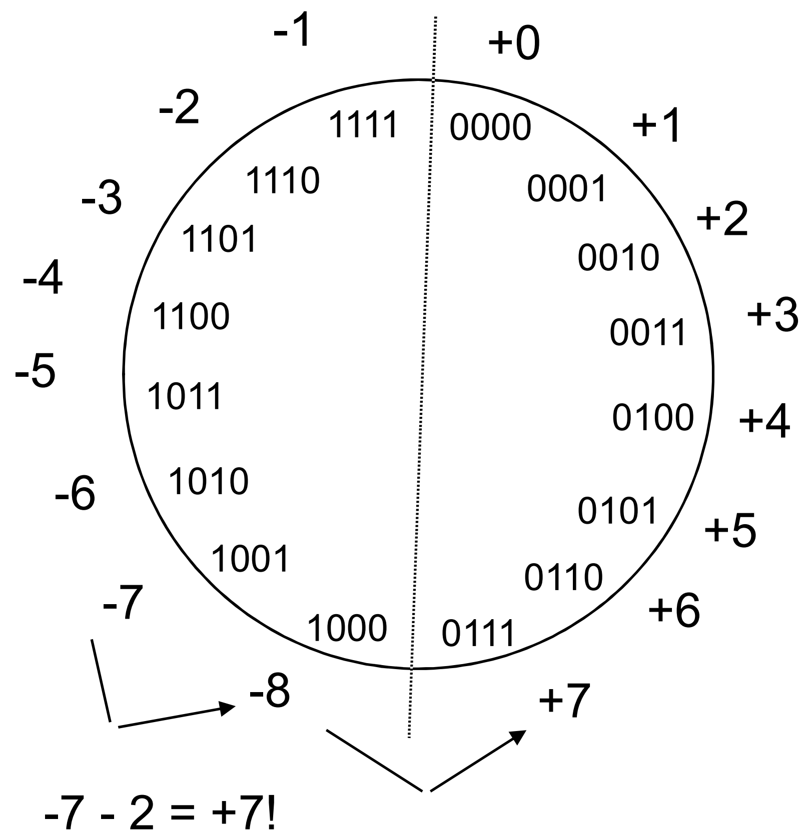
Three Representations

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)

Overflow Conditions

- ❖Add two positive numbers to get a negative number
- ❖or two negative numbers to get a positive number





Overflow: An Error

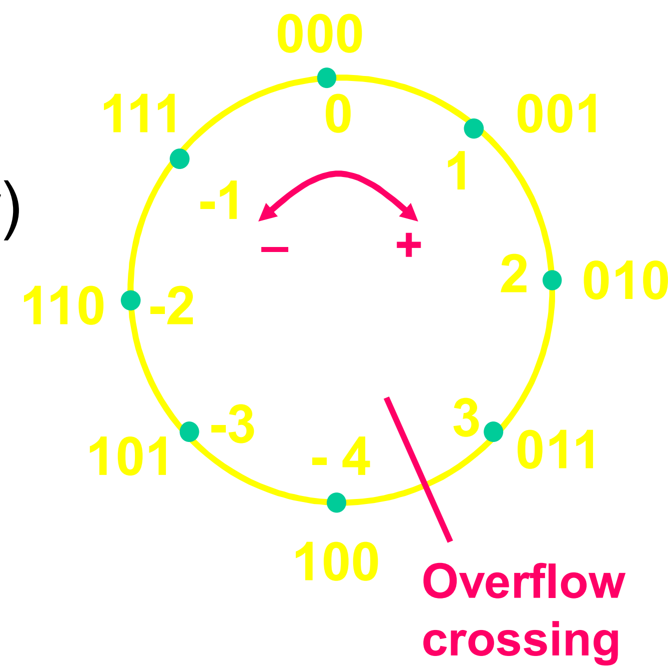
❖ Examples: Addition of 3-bit integers (range - 4 to +3)

- -2-3 = -5

  110 = -2
  + 101 = -3
  = 1011 = 3 (error)

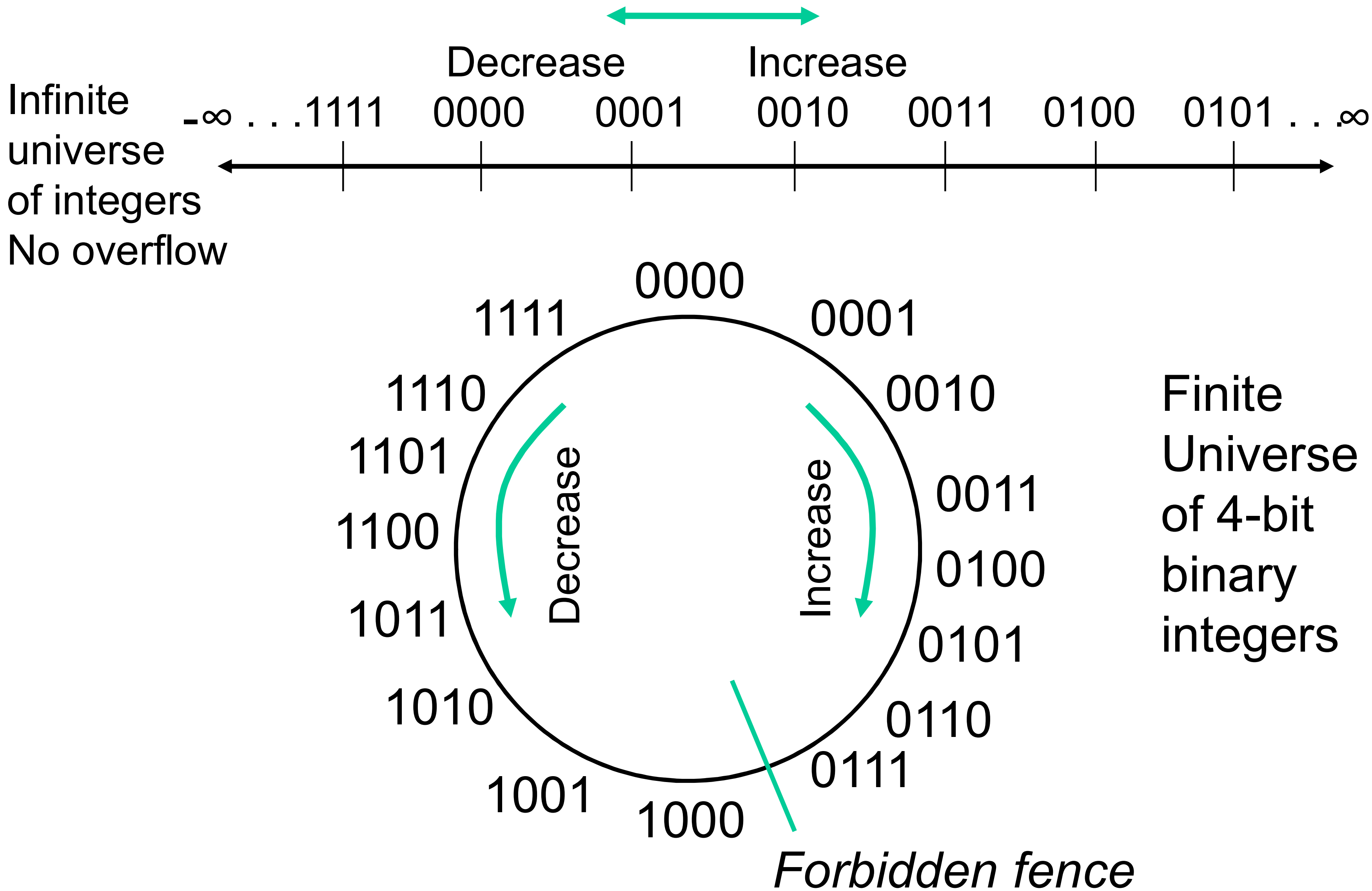
- 3+2 = 5

  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.

Overflow and Finite Universe



Overflow Conditions

	0 1 1 1
5	0 1 0 1
3	0 0 1 1
-8	1 0 0 0

Overflow

	1 0 0 0
-7	1 0 0 1
-2	1 1 0 0
7	1 0 1 1 1

Overflow

	0 0 0 0
5	0 1 0 1
2	0 0 1 0
7	0 1 1 1

No overflow

	1 1 1 1
-3	1 1 0 1
-5	1 0 1 1
-8	1 1 0 0 0

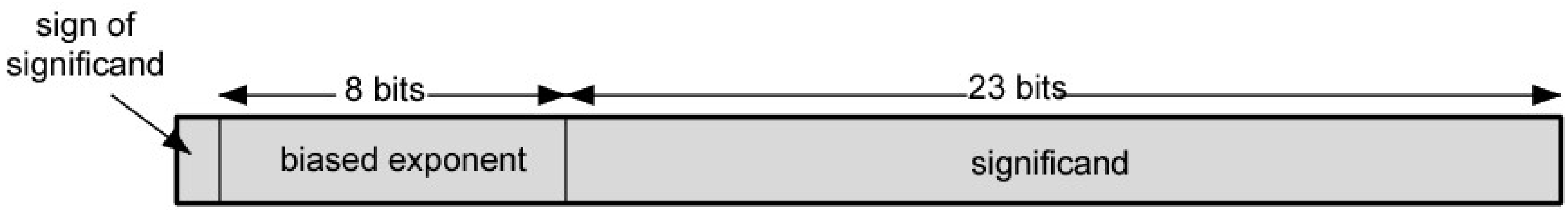
No overflow

Overflow when carry in to sign does not equal carry out

Real Numbers

- Numbers with fractions
- Could be done in pure binary
  - 1001.1010 = $2^4 + 2^0 + 2^{-1} + 2^{-3}$ =9.625
- Where is the binary point?
- Fixed?
  - Very limited
- Moving?
  - How do you show where it is?

Floating Point



(a) Format

- +/- .significand x $2^{exponent}$
- Misnomer
- Point is actually fixed between sign bit and body of mantissa
- Exponent indicates place value (point position)

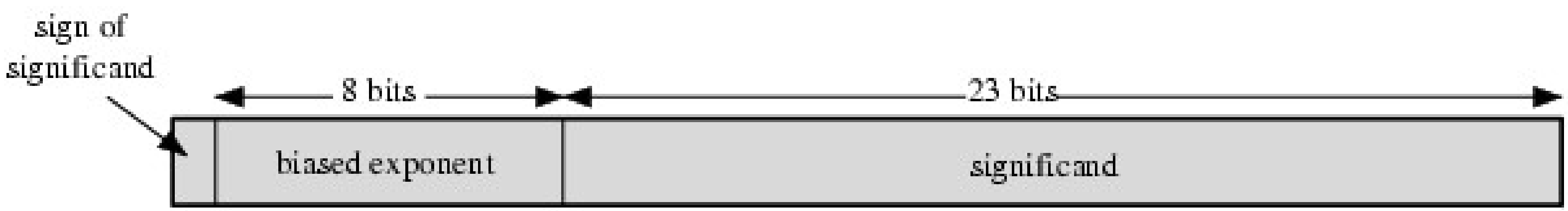
Signs of Floating Point

- Mantissa is stored in 2’s compliment
- Exponent is in excess or biased notation
  - e.g. Excess (bias) 128 means
  - 8 bit exponent field
  - Pure value range 0-255
  - Subtract 128 to get correct value
  - Range -128 to +127

Normalization

- FP numbers are usually normalized
- i.e. exponent is adjusted so that leading bit (MSB) of mantissa is 1
- Since it is always 1 there is no need to store it
- (c.f. Scientific notation where numbers are normalized to give a single digit before the decimal point
- e.g. 3.123×10^3)

Floating Point Examples



(a) Format

$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.638125 \times 2^{20}$

$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.638125 \times 2^{20}$

$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.638125 \times 2^{-20}$

$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.638125 \times 2^{-20}$

(b) Examples

Exponent is presented in biased-127 format

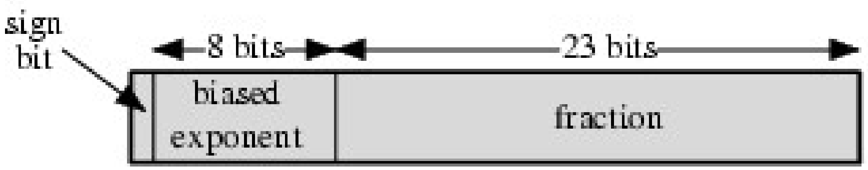
FP Ranges

- For a 32 bit number
  - 8 bit exponent
  - +/- $2^{127} \approx 1.5 \times 10^{77}$
- Accuracy
  - The effect of changing lsb of mantissa
  - 23 bit mantissa $2^{-23} \approx 1.2 \times 10^{-7}$
  - About 6 decimal places

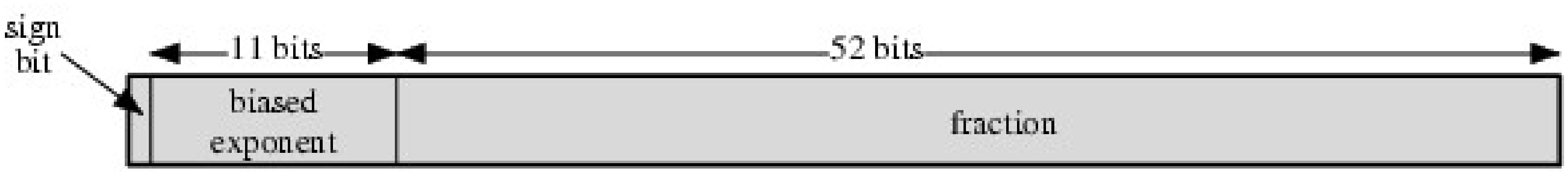
IEEE 754 Formats

- Standard for floating point storage
- 32 and 64 bit standards
- 8 and 11 bit exponent respectively

IEEE 754 Formats



(a) Single format



(b) Double format

Other Codes

- Excess Code (Excess-128)
- GREY Code
- BCD (Binary Coded Decimal)

Character Representation

- ASCII (American Standard Code for Information Interchange)
- EBCDIC (Extended Binary Coded Decimal Interchange Code)
- UNICODE