

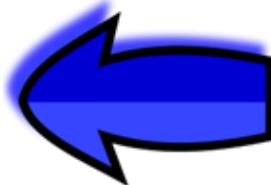


Chapter 2: The Language of Bits

Basic Computer Architecture

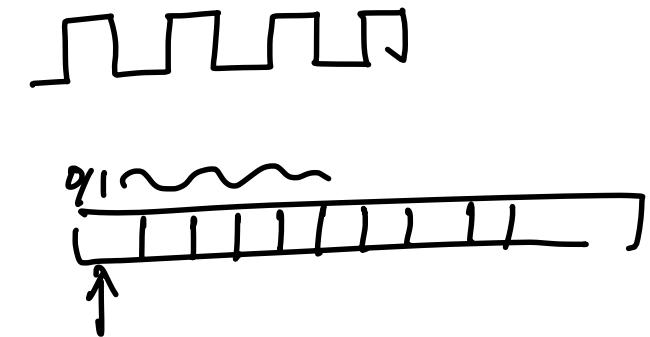
Outline

- * Boolean Algebra
- * Positive Integers
- * Negative Integers
- * Floating-Point Numbers
- * Strings



What does a Computer Understand ?

- * Computers do not understand natural human languages, nor programming languages
- * They only understand the language of **bits**



Bit	0 or 1
Byte	8 bits
Word	4 bytes
kiloByte	1024 bytes
megaByte	10^6 bytes

$$1 \text{ KA} = 10^3 \text{ gm}$$
$$1 \text{ KB} = 2^{10} \text{ bite}$$
$$\downarrow 1024$$
$$\downarrow 10^3$$
$$\text{bit} \downarrow 10^3$$

$$\Rightarrow 2^{10} \times 2^{10} \\ 10^3 \times 10^3 = 10^6 \\ 2^{10} \times 2^{10}$$

Review of Logical Operations

OR

$$* A + B \text{ (A or B)}$$

$A \text{ OR } B$
 $A \text{ AND } B$

O/I

A	B	A + B	O/I
0	0	0	
1	0	1	
0	1	1	
1	1	1	

= Truth Table

AND

$$* A.B \text{ (A and B)}$$

$A \text{ AND } B$

NOT

A	B	A.B
0	0	0
1	0	0
0	1	0
1	1	1

Review of Logical Operations - II

A	B	A NAND B
0	0	1
1	0	1
0	1	1
1	1	0

A	B	A NOR B
0	0	1
1	0	0
0	1	0
1	1	0

\bar{A}
 $\sim A$
 $A!$

- * NAND and NOR operations
- * These are **universal operations**. They can be used to implement any Boolean function.

Review of Logical Operations

* XOR Operation : $(A \oplus B)$

A	B	A XOR B
0	0	0
1	0	1
0	1	1
1	1	0

How many truth tables we can build?

Review of Logical Operations

✓ * NOT operator

- * Definition: $\overline{0} = 1$, and $\overline{1} = 0$
- * Double negation: $\overline{\overline{A}} = A$, NOT of (NOT of A) is equal to A itself

* OR and AND operators

- * Identity: $A + \underline{0} = A$, and $A.\underline{1} = A$
- * Annulment: $A + 1 = 1$, $A.0 = 0$

- * **Idempotence**: $A + A = A$, $A \cdot A = A$, The result of computing the OR and AND of A with itself is A.
- * **Complementarity**: $\underline{A} + \underline{\overline{A}} = 1$, $A \cdot \overline{A} = 0$
- * **Commutativity**: $A + B = B + A$, $A \cdot B = B \cdot A$, the order of Boolean variables does not matter
- * **Associativity**: $A + (B + C) = (A + B) + C$, $A \cdot (B \cdot C) = (A \cdot B) \cdot C$, similar to addition and multiplication.
- * **Distributivity**: $A \cdot (B + C) = \underline{A} \cdot B + \underline{A} \cdot C$, $A + (B \cdot C) = (A + B) \cdot (A + C)$ → Use this law to open up parentheses and simplify expressions

$$x(y+z) = xy + xz$$

$$A + B$$

$$A + (B + C)$$

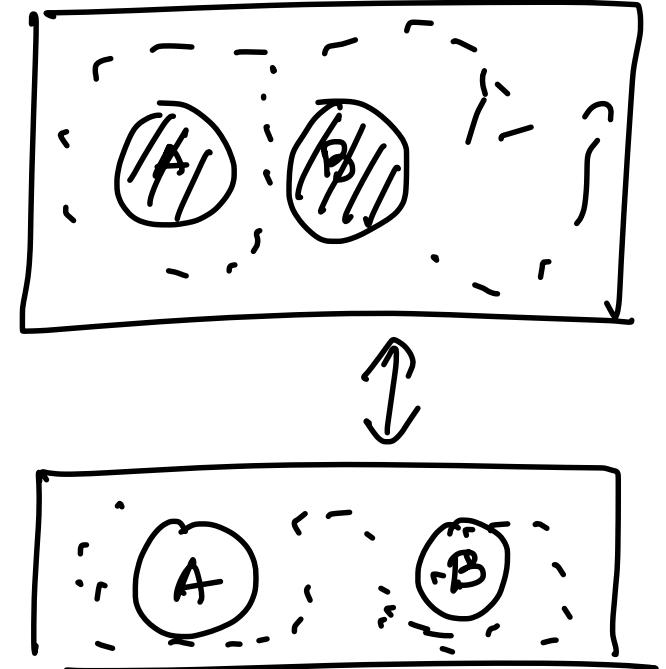
$$(A + B) + C$$

De Morgan's Laws

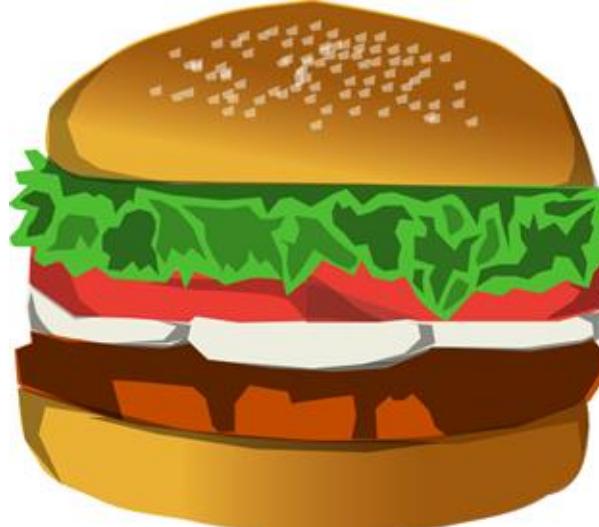
- * Two very useful rules

$$\neg \overline{A + B} = \overline{\overline{A} \cdot \overline{B}}$$

$$\neg \overline{A \cdot B} = \overline{\overline{A}} + \overline{\overline{B}}$$



Consensus Theorem



* Prove :

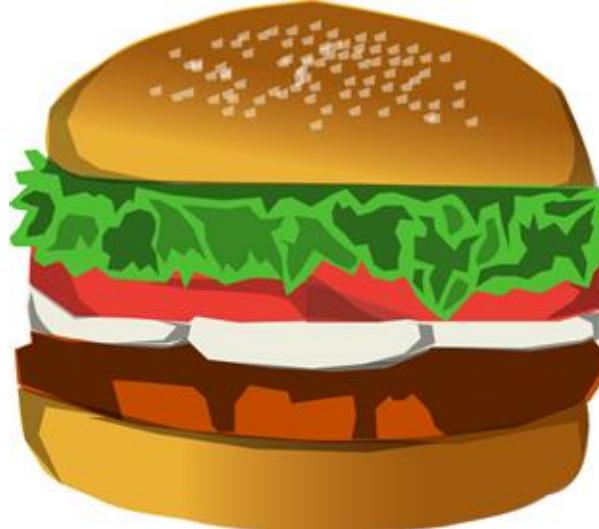
$$* \quad \boxed{X.Y + \bar{X}.Z + Y.Z = X.Y + \bar{X}.Z} \quad -$$

$$X.Y + \bar{X}Z + YZ (X + \bar{X})$$

$$\overbrace{XY}^{\leftarrow} + \overbrace{\bar{X}Z}^{\leftarrow} + \overbrace{YZ}^{\leftarrow} + \overbrace{\bar{X}YZ}^{\leftarrow}$$

$$XY(1+Z) + \bar{X}Z(1+\bar{Y}) = XY + \bar{X}Z$$

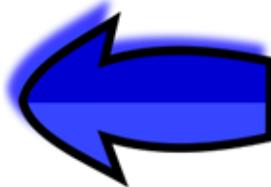
Consensus Theorem



- * Prove :
 - * $X.Y + \bar{X}.Z + Y.Z = X.Y + \bar{X}.Z$

Outline

- * Boolean Algebra
- * Positive Integers
- * Negative Integers
- * Floating Point Numbers
- * Strings



Representing Positive Integers

* Ancient Roman System

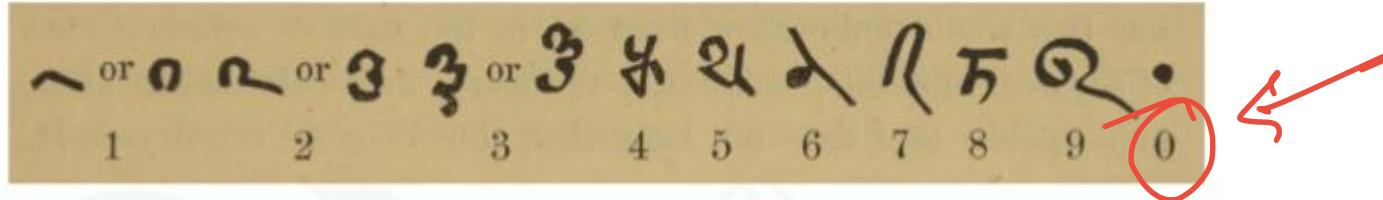
Symbol	I	V	X	L	C	D	M
Value	1	5	10	50	100	500	1000

place-value representation

* Issues :

- * There was no notion of 0
- * Very difficult to represent large numbers
- * Addition, and subtraction (**very difficult**)

Indian System (place -value system)



Bakshali numerals, 7th century AD

- * Uses the place value system

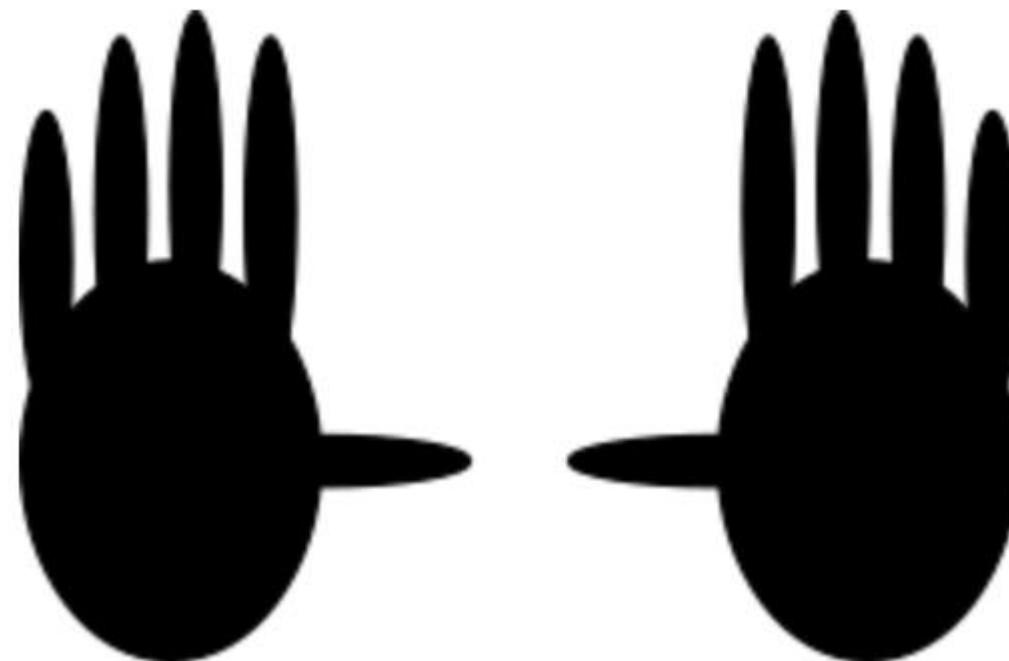
$$5301 = \underline{5} * \underline{10^3} + 3 * 10^2 + 0 * 10^1 + 1 * 10^0$$

Example in base 10

5301
= place value.
base value

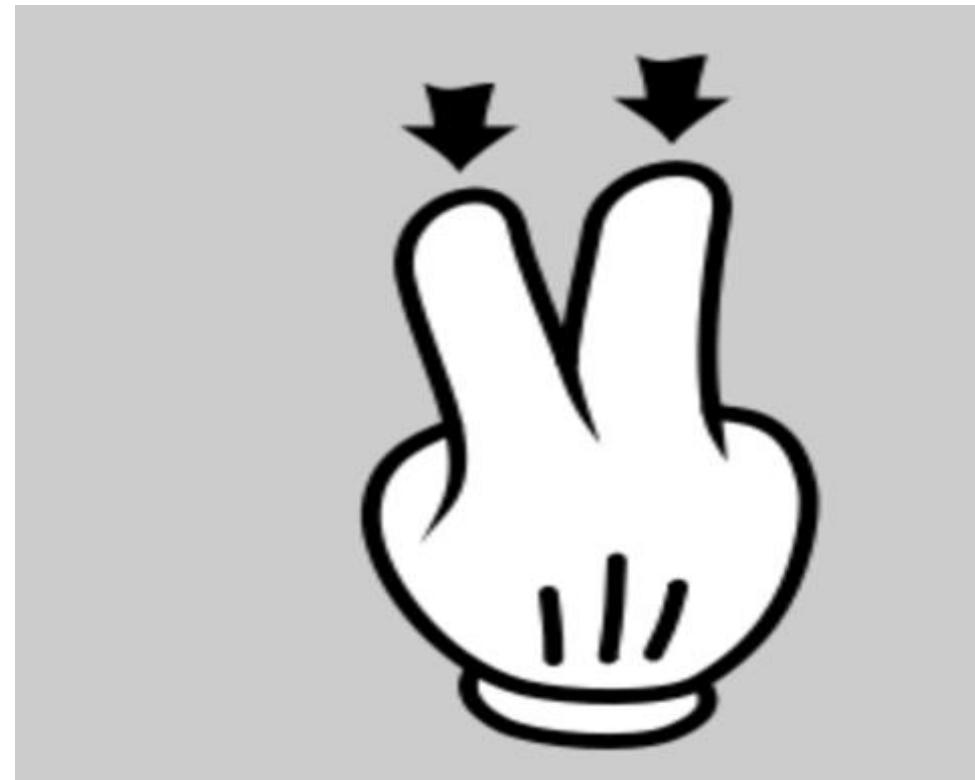
Number Systems in Other Bases

- * Why do we use base 10 ?
 - * because ...



What if we had a world in which ...

- * People had only two fingers.



Binary Number System

- * They would use a number system with base 2.

Number in decimal	Number in binary
5	101
100	1100100
500	111110100
1024	10000000000

19
= 10011
↑
MSB LSB
least significant bit

$$\begin{array}{l} \begin{array}{r} 2^4 \\ 2^3 \\ 2^2 \\ 2^1 \\ 2^0 \end{array} \end{array} \begin{array}{l} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{array} = \underline{\underline{101}}$$
$$= 10^0 \times 5 + 10^1 \times 1 + 10^2 \times 0 + 10^3 \times 1 + 10^4 \times 1$$
$$= 1 + 0 + 0 + 1 + 16$$
$$= 18$$

base value = 2
place value starts from 0

MSB and LSB

- * **MSB (Most Significant Bit)** → The leftmost bit of a binary number. E.g., MSB of 1110 is 1
- * **LSB (Least Significant Bit)** → The rightmost bit of a binary number. E.g., LSB of 1110 is 0

Hexadecimal and Octal Numbers

- * Hexadecimal numbers

- * Base 16 numbers – 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- * Start with 0x

- * Octal Numbers = base=8

- * Base 8 numbers – 0,1,2,3,4,5,6,7
- * Start with 0

1 Why Octal given that we have Hex a ?

0	Base = 16
1	
2	
3	
4	
5	
6	
7	
8	
9	
- A -	10
- B -	11
- C -	12
- D -	13
- E -	14
- F -	15

Examples

Convert 110010111 to the octal format : $\frac{110}{\underline{\quad}} \frac{010}{\underline{\quad}} \frac{111}{\underline{\quad}} = 0627$

Convert 11100010111 to the hex format : $\frac{1110}{\underline{\quad}} \frac{0010}{\underline{\quad}} \frac{1111}{\underline{\quad}} = 0xE2F$

$$\begin{aligned} N &= a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 \\ &= \underbrace{(a_0 \times 2^0 + a_1 \times 2^1 + a_2 \times 2^2)}_{2^0} + 2^3 (a_3 \times 2^0 + a_4 \times 2^1 + a_5 \times 2^2) + 2^6 (a_6 \times 2^0 + a_7 \times 2^1 + a_8 \times 2^2) \\ &= 2^0 (\underbrace{a_0}_{b_0} - \overline{f}) + 2^3 (\underbrace{a_3}_{b_1} - \overline{f}) + (2^3)^2 (\underbrace{a_6}_{b_2} - \overline{f}) + \dots \\ &= 8^0 b_0 + 8^1 b_1 + 8^2 b_2 8 + \dots \end{aligned}$$

Examples

$$(x)_2 \rightarrow (y)_5$$

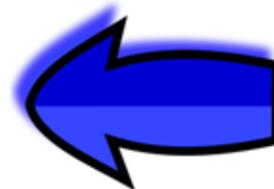
Convert 110010111 to the octal format : $\underbrace{110}_{} \underbrace{010}_{} \underbrace{111}_{} = 0627$

Convert 11100010111 to the hex format : $\underbrace{1110}_{} \underbrace{0010}_{} \underbrace{1111}_{} = \text{0xE2F}$

2. Can we convert a 2^n -base system to a base system which can't be represented by 2^m ; e.g.; binary $\rightarrow (y)_{\cancel{2}}$ without converting to decimal

Outline

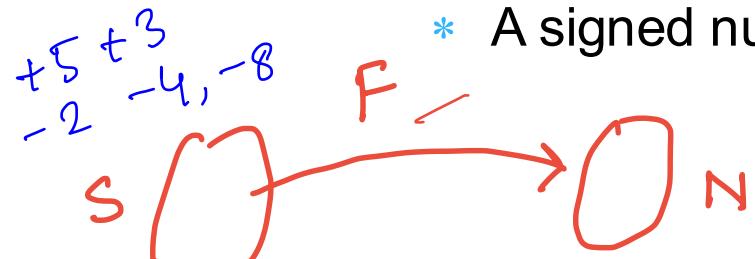
- * Boolean Algebra
- * Positive Integers
- * Negative Integers
- * Floating Point Numbers
- * Strings



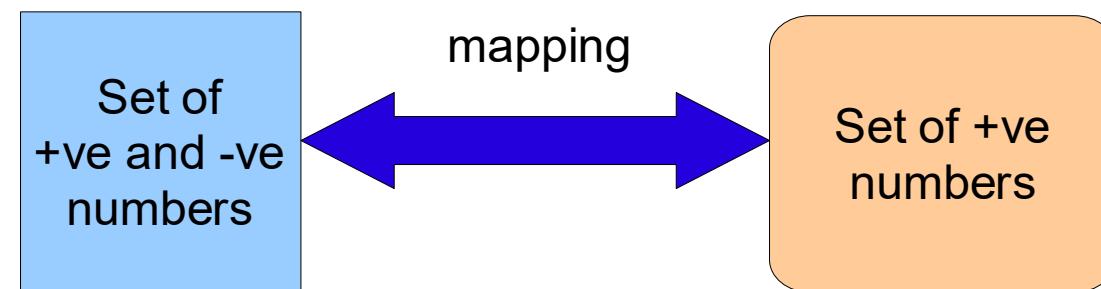
Representing Negative Integers

* Problem

- * Assign a **binary representation** to a **negative integer**
- * Consider a negative integer, S
- * Let its binary representation be : $x_nx_{n-1}\dots x_2x_1$
 $(x_i=0/1)$
- * We can also expand it to represent an unsigned,
+ve, number, N
- * If we interpret the binary sequence as :
 - * An unsigned number, **we get N**
 - * A signed number, **we get S**



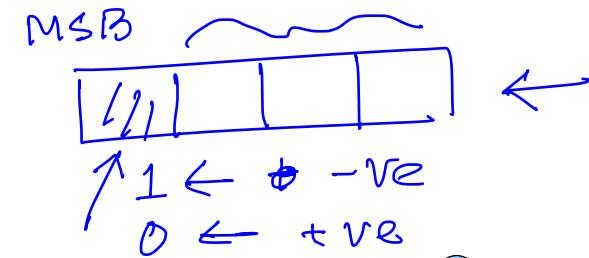
- * We need a mapping :
 - * $F : S \rightarrow N$ (mapping function)
 - * $S \rightarrow$ set of numbers (both positive and negative – signed)
 - * $N \rightarrow$ set of positive numbers (unsigned)



Properties of the Mapping Function

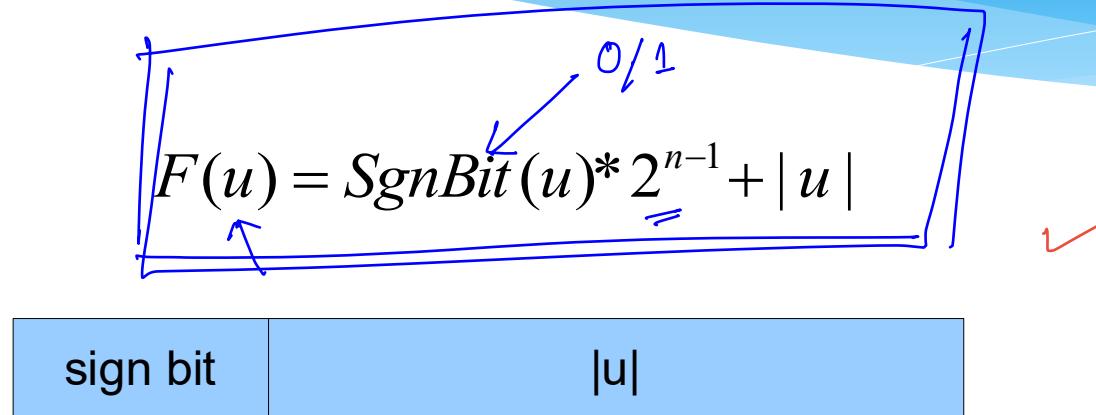
- * Preferably, needs to be a one to one mapping
- * All the entries in the set, S, need to be mapped
- * It should be easy to perform addition and subtraction operations on the representation of signed numbers
- * Assume an n bit number system

$$[-5] = 5$$



$$\text{SgnBit}(u) = \begin{cases} 1, & u < 0 \\ 0, & u \geq 0 \end{cases}$$

Sign-Magnitude Base Representation



✓

$$\begin{array}{c} -5 \\ \hline 1101 \\ \uparrow \quad \underbrace{\quad}_{1 \times 2^3} \quad |5| \end{array} \quad n=4 \quad n-1=3$$

13

$$+4 \quad \boxed{0100} = 4$$

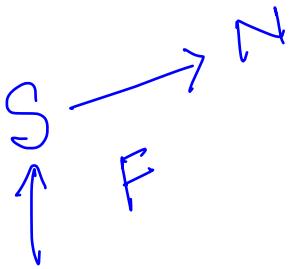
- * Examples :
 - * -5 in a 4 bit number system : 1101
 - * 5 in a 4 bit number system : 0101
 - * -3 in a 4 bit number system : 1011

Problems

- * There are two representations for 0
 - * 000000
 - * 100000
- * Addition and subtraction are difficult ↴
- * The most important takeaway point :
 - * Notion of the sign bit



1's Complement Representation



$$F(u) = \begin{cases} u, & u \geq 0 \\ \sim(|u|) \text{ or } (2^n - 1 - |u|), & u < 0 \end{cases}$$

$n = 4$
 $2^4 = 16$
 $2^4 - 1 = 15$

Notion of sign bit also exists

$$\begin{aligned} f(-3) &= 2^4 - 1 - |-3| \\ &= 15 - 3 \\ &= 12 \\ &= 1100 \end{aligned}$$

- * $3 \rightarrow 0011$
- * $-3 \rightarrow \cancel{1100} \Rightarrow 12$
- * $5 \rightarrow 0101$
- * $-5 \rightarrow 1010$
- Annotations in blue:
- $x^3 = 0011$
 - $\sim 3 = \cancel{1100}$
 - $\sim 12 = 1010$

Problems

- * Two representations for 0

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

$\begin{array}{r} 0000000 \\ 1111111 \\ \hline \end{array}$

- * 0000000
- * 1111111

- * Easy to add +ve numbers

- * Hard to add -ve numbers

- * Point to note :

- * The idea of a complement



$$\begin{array}{r} 3 + (-2) = 1 \\ 0011 \\ + 1101 \\ \hline 10000 \\ \xrightarrow{\text{+}} \\ 0001 \end{array}$$

Bias Based Approach

$$F(u) = u + \text{bias}$$

- * Consider a 4 bit number system with bias equal to 7

- * $-3 \rightarrow 0100$

$$-3 = -3 + 7 = +4$$

- * $3 \rightarrow 1010$

$$+2 = +2 + 7 = +9$$

- * $F(u+v) = F(u) + F(v) - \text{bias}$

- * Add and Sub are also easy

$$\begin{array}{r} +3 \\ -3 \\ \hline \end{array} = \begin{array}{r} +3 + 7 = 10 \\ -3 + 7 = 4 \\ \hline \end{array} \quad \begin{array}{r} 1010 \\ -0100 \\ \hline 1110 \end{array}$$

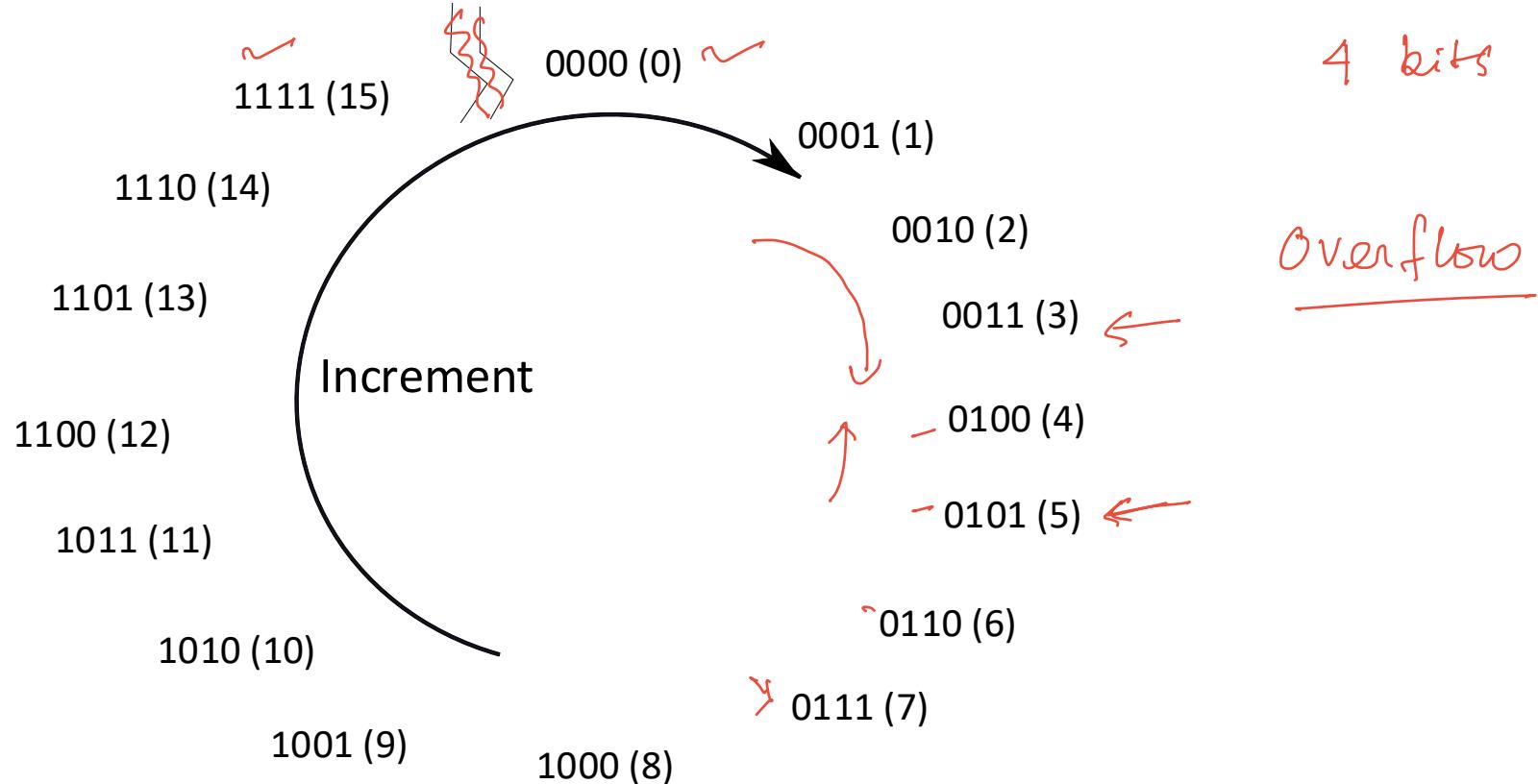
- * Multiplication is difficult

$$F(u-v) = F(u) - F(v) + \text{bias}$$

$$F(3-3) = F(3) - F(3) + 7 = 7 = 0$$

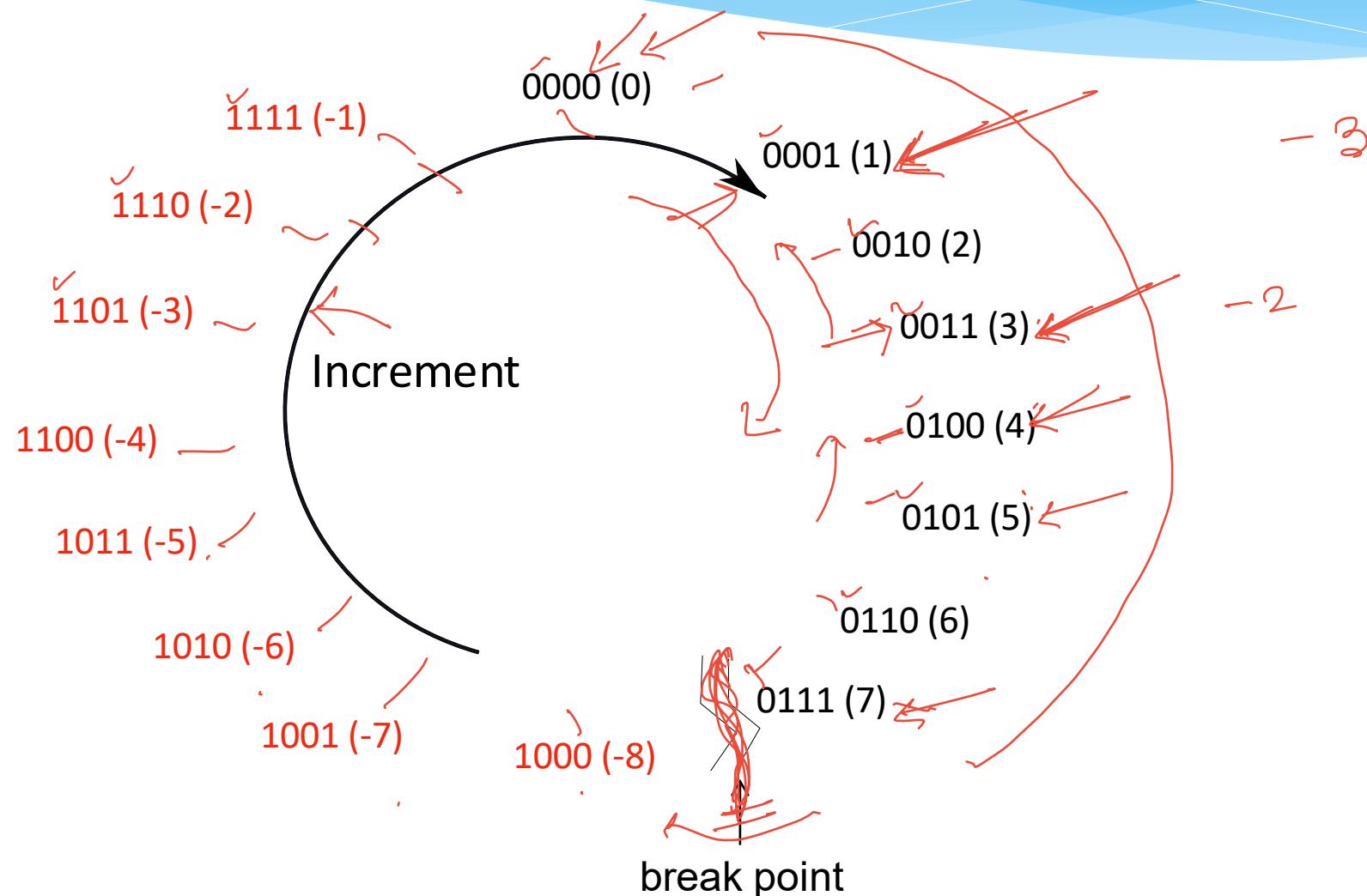
$$\begin{array}{r} 14 \\ -7 \\ \hline 7 \end{array}$$

The Number Circle



Clockwise: increment
Anti-clockwise: decrement

Number Circle with Negative Numbers



Using the Number Circle

- * To add \underline{M} to a number, \underline{N}

- * locate \underline{N} on the number circle

- * If \underline{M} is +ve

- * Move M steps clockwise

- * If \underline{M} is -ve

- * Move M steps anti-clockwise, or $2^n - \underline{M}$ steps clockwise

- * If we cross the break-point

- * We have an overflow / underflow

- * The number is too large/ too small to be represented

2
 $n = 16$

2's Complement Notation

$$F(u) = \begin{cases} u, & 0 \leq u \leq 2^{n-1} - 1 \\ 2^n - |u|, & -2^{n-1} \leq u < 0 \end{cases}$$

$$\begin{aligned} n &= 4 \\ 2^{3-1} &= 2^{n-1} = 8 \\ -2^{n-1} &= -8 \end{aligned}$$

- * $F(u)$ is the index of a point on the **number circle**. It varies from 0 to $2^n - 1$
- * Examples

$$* \rightarrow 4 \rightarrow \underline{0100}$$

$$* \rightarrow -4 \rightarrow \underline{1100}$$

$$* \rightarrow 5 \rightarrow \underline{0101}$$

$$* \rightarrow -3 \rightarrow \underline{1101}$$

$$\begin{aligned} -8 &\rightarrow 0000 \\ -2 &\equiv 14 \\ -3 &= 13 \end{aligned}$$

Properties of the 2's Complement Notation

- * Range of the number system :
 - * $-2^{(n-1)}$ to $2^{n-1} - 1$ ✓
- 8 + 7
- * There is a unique representation for 0 → 000000
- * msb of F(u) is equal to SgnBit(u)
 - * Refer to the number circle
 - * For a +ve number, $F(u) < 2^{(n-1)}$. MSB = 0
 - * For a -ve number, $F(u) \geq 2^{(n-1)}$. MSB = 1

Properties - II

- * Every number in the range $[-2^{(n-1)}, 2^{(n-1)} - 1]$

- * Has a unique mapping

- * Unique point in the number circle

$$a \equiv b \rightarrow (a = b \bmod 2^n)$$

$a - b \bmod 2^n = 0 \quad n = 4$

$+2 \equiv +18$

* \equiv means same point on the number circle

- * $F(-u) \equiv 2^n - F(u)$

- * Moving $F(u)$ steps counter clock wise is the same as moving $2^n - F(u)$ steps clockwise from 0

Prove : $F(u+v) \equiv F(u) + F(v)$

* Start at point u

- * Its index is $F(u)$
- * If v is +ve,
 - * move v points clockwise. We arrive at $F(u+v)$.
 - * Its index is equal to $(F(u) + v) \bmod 2^n$.
 - * Since $v = F(v)$, we have $F(u+v) = (F(u) + F(v)) \bmod 2^n$

$$u = 2$$

$$v = 3$$

$$\begin{aligned}F(2+3) &= F(2) + F(3) \\F(5) &= F(2) + F(3) \\&= 0010 + 0011 \\&= 0101\end{aligned}$$

Prove : $F(u+v) \equiv F(u) + F(v)$



- * If v is -ve,
 - * move $|v|$ points anti-clockwise.
 - * Same as moving $2^n - |v|$ points clockwise.
 - * We arrive at $F(u+v)$. ✓
 - * $F(v) = 2^n - |v|$
 - * The index – $F(u+v)$ – is equal to:
 - * $(F(u) + 2^n - |v|) \text{ mod } 2^n = (F(u) + F(v)) \text{ mod } 2^n$

$$\begin{aligned} u &= 2 \\ v &= -4 \\ F(2+(-4)) &= F(2) + F(-4) \\ &= 0010 + 1100 \\ &= 1110 \\ &= F(-2) \end{aligned}$$

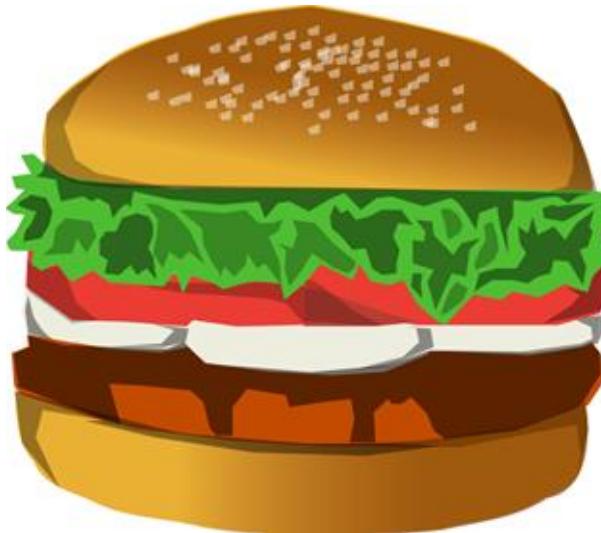
Subtraction

- * $F(u-v) \equiv F(u) + F(-v)$
 $\equiv F(u) + 2^n - F(v)$

$$\begin{aligned}F(3-5) &= F(3) + F(-5) \\&= 3 + 16 - 5 \\&= 14 = F(-2)\end{aligned}$$

- * Subtraction is the same as addition
- * Compute the 2's complement of $F(v)$

Prove that :



* Prove that :

$$F(u^*v) \equiv F(u) * F(v)$$

$$u = 2, v = -3$$

$$\begin{aligned} & F(2 \times (-3)) \\ & \equiv F(2) \times F(-3) \\ & = 2 \times 13 \\ & = 26 \bmod 16 \\ & \Rightarrow 10 = F(-6) \end{aligned}$$

① if u and v are +ve, it is trivial.

② if u and v are -ve, $u = -|u|$ $v = -|v|$

$$F(u) \times F(v) = \frac{(2^n - F(|u|)) \times (2^n - F(|v|))}{(2^n - F(|u|) + F(|v|))}$$

$$= \cancel{2^{2n}} - \cancel{2^n} (F(|u|) + F(|v|)) + F(|u|) \times F(|v|)$$

$$\approx \emptyset F(|u|) \times F(|v|)$$

$$= F(|u| \times |v|) = F(u \times v)$$

③ u is +ve

v is -ve

$$u = |u|$$

$$v = -|v|$$

$$F(u) \times F(v) = F(u) \times (2^n - F(|v|))$$

$$= 2^n \cancel{F(u)} - F(u) \times F(|v|)$$

$$= -F(u) \times F(|v|)$$

$$= -F(u \times |v|) \quad u > 0 \quad |v| > 0$$

$$F(-u) = 2^n - F(|u|)$$

negative rule

$$= 2^n - F(u \times |v|)$$

$$= F(- (u \times |v|))$$

$$= F(u \times \underline{(-|v|)}) = F(u \times v)$$

Computing the 2's Complement

* $2^n - u$

$$= \cancel{2^n} - 1 - u + 1$$

$$= \cancel{-u} + 1$$

* $\sim u$ (1's complement)

$$2^m - 1$$

* 1's complement of 0100

$$\begin{array}{r} & 1111 \\ - & 0100 \\ \hline & 1011 \end{array}$$

2's complement of
0100

$$\begin{array}{r} 1011 \\ + 0001 \\ \hline 1100 \end{array}$$

Sign Extension

- * Convert a n bit number to a m bit 2's complement number ($m > n$)

$$+4 = 0100$$

↓
0000 0100

- * +ve

- * Add $(m-n)$ 0s in the msb positions
- * Example, convert 0100 to 8 bits → 0000
0100

$$\begin{array}{r} 1100 \\ \boxed{1111\ 1100} = -282 \end{array}$$

- * -ve

- * $F(u) = \underline{2^n - |u|}$ (n bit number) system
- * Need to calculate $F'(u) = \underline{\underline{2^m - |u|}}$

Sign Extension - II

$m > n$

$$= 2^m + 2^{n+1} + \dots + 2^{m-1}$$

$$2^m - 2^n$$

$$= 2^m (2^{m-n} - 1)$$

$$2^{m-n-1} = 0 + 2 + 2^2 + \dots + 2^{n-1}$$

$$= 2^m (1 + 2 + 2^2 + \dots + 2^{m-n-1})$$

$$= 2^m (2 + 2^{n+1} + \dots + 2^{m-1})$$

$$* \quad 2^m - u - (2^n - u)$$

$$= \boxed{2^m - 2^n}$$

$$= 2^n + 2^{(n+1)} + \dots + 2^{(m-1)}$$

$$= \boxed{11110000} + (-2)$$

$m-n$

n

$m = 4 \quad n = 2$

$$2^2 + 2^3 =$$

$$2^4 - 2^2$$

$$= 16 - 4$$

$$= 12$$

$$= 2^2 + 2^3$$

$$4 - 8$$

$$\begin{array}{r} 1111\ 0000 \\ + 1100 \\ \hline 1111\ 0100 \end{array}$$

$$F'(u) = F(u) + 2^m - 2^n$$

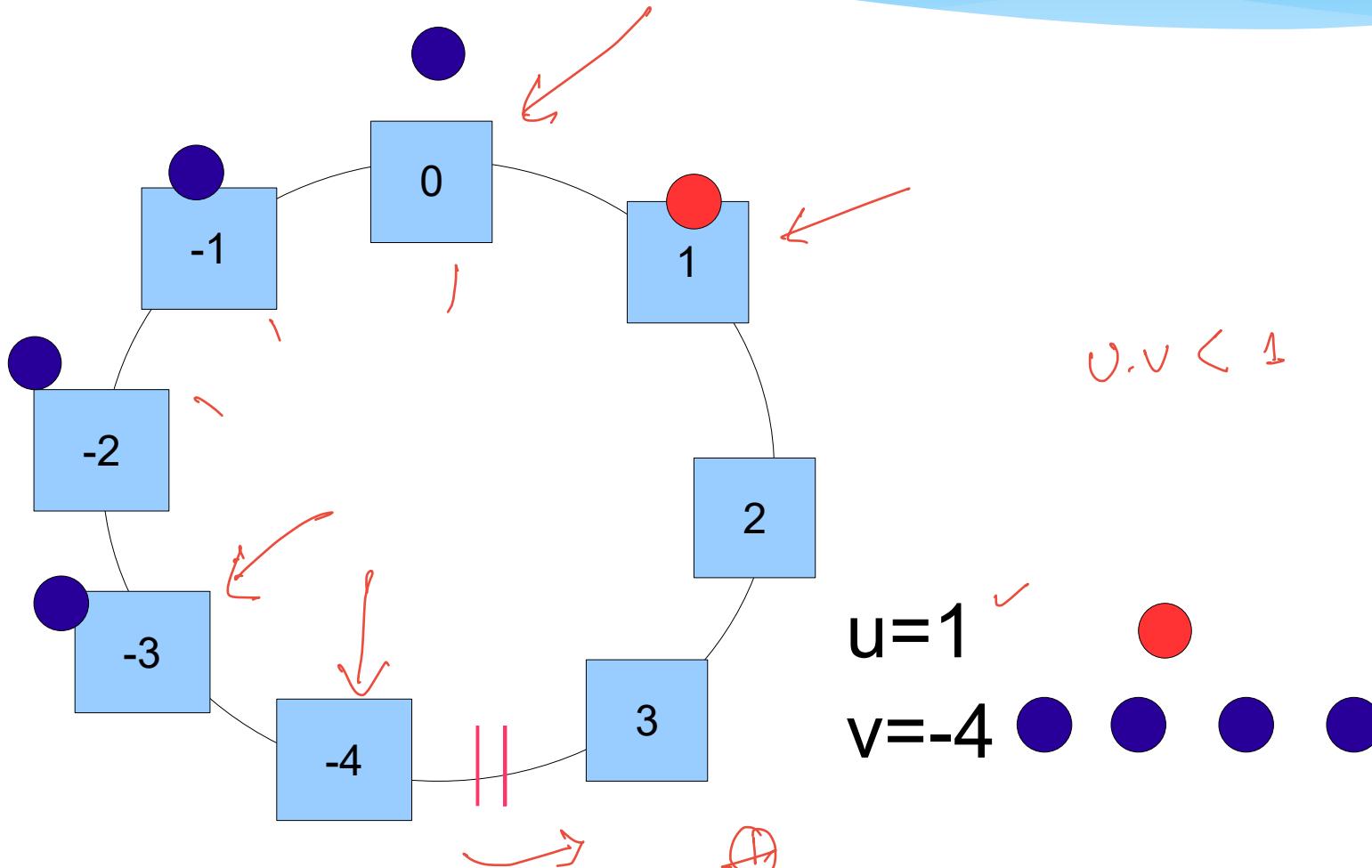
Sign Extension - III

- * To convert a negative number :
 - * Add $(m-n)$ 1s in the msb positions
- * In both cases, extend the sign bit by :
 - * $(m-n)$ positions

The Overflow Theorem

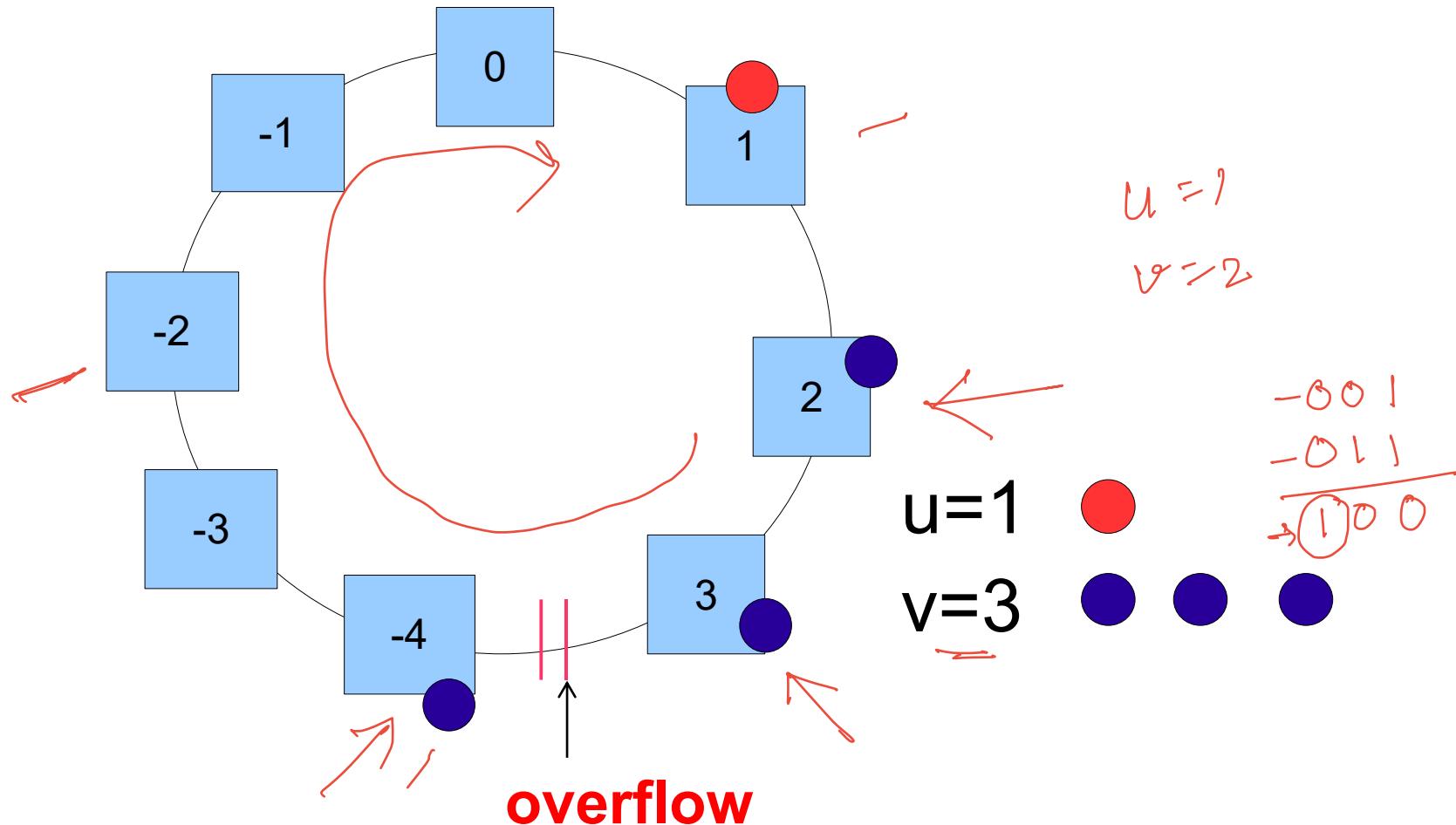
- * Add : $u + v$
- * If $\underline{uv < 0}$, there will **never be an overflow**
- * Let us go back to the number circle
 - * There is an overflow only when we cross the break-point
 - * If $\underline{uv = 0}$, one of the numbers is 0 (no overflow)
 - * If $\underline{uv > 0}$, an **overflow is possible**

Number Circle: $uv < 0$



(?) both the numbers are positive,
x and the resultant number is.
-ve and there is no overflow

Number Circle: $uv > 0$

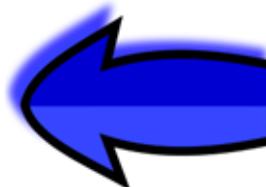


Conditions for an Overflow

- * $uv \leq 0$
 - * Never
- * $\underline{uv > 0}$ (u and v have the same sign)
 - * The sign of the result is different from the sign of u X

Outline

- * Boolean Algebra
- * Positive Integers
- * Negative Integers
- * Floating-Point Numbers
- * Strings



Floating-Point Numbers

* What is a floating-point number ?

- * 2.356
- * 1.3e-10
- * -2.3e+5

* What is a fixed-point number ?

- * Number of digits after the decimal point is fixed
- * 3.29, -1.83

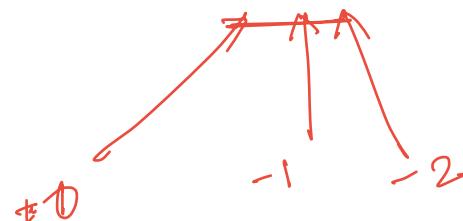
Generic Form for Positive Numbers

- * Generic form of a number in base 10

$$A = \sum_{i=-n}^n x_i 10^i$$

- * Example :

- * $3.29 = 3 * 10^0 + 2 * 10^{-1} + 9 * 10^{-2}$



Generic Form in Base 2

- * Generic form of a number in base 2

$$A = \sum_{i=-n}^n x_i 2^i$$

Number	Expansion
0.375	$2^{-2} + 2^{-3}$
1	2^0
1.5	$2^0 + 2^{-1}$
2.75	$2^1 + 2^{-1} + 2^{-2}$
17.625	$2^4 + 2^0 + 2^{-1} + 2^{-3}$

$$\begin{aligned} 0.375 \times 2 &= 0.750 \\ 0.750 \times 2 &= 1.5 \\ 1.5 \times 2 &= 1.0 \quad \checkmark \\ &\quad \text{↓} \\ &\quad \underline{0.11} \end{aligned}$$

Binary Representation

- * Take the base 2 representation of a floating-point (FP) number
- * Each coefficient is a binary digit

Number	Expansion	Binary Representation
0.375	$2^{-2} + 2^{-3}$	0.011
1	2^0	1.0
1.5	$2^0 + 2^{-1}$	1.1
2.75	$2^1 + 2^{-1} + 2^{-2}$	10.11
17.625	$2^4 + 2^0 + 2^{-1} + 2^{-3}$	10001.101

Normalized Form

- * Let us create a standard form of all floating point numbers

$$A = (-1)^S * P * 2^X, (P = 1 + M, 0 \leq M < 1, X \in Z)$$


* $\text{S} \rightarrow \text{sign bit}$, $P \rightarrow \text{significand}$ $1 \leq P < 2$

- * $M \rightarrow \text{mantissa}$, $X \rightarrow \text{exponent}$, $Z \rightarrow \text{set of integers}$

$$A = 2.6 \quad (-1)^0 \times 1.3 \times 2^1$$
$$P = 1.3 \quad Z = 1$$
$$M = -3$$

Examples (in decimal)

* $1.3827 * 1e-23$

* Significand (P) = 1.3827

* Mantissa (M) = 0.3827

* Exponent (X) = -23

* Sign (S) = 0 ✓

* $-1.2 * 1e+5$

* P = 1.2 , M = 0.2

* S = 1, X = 5

IEEE 754 Format

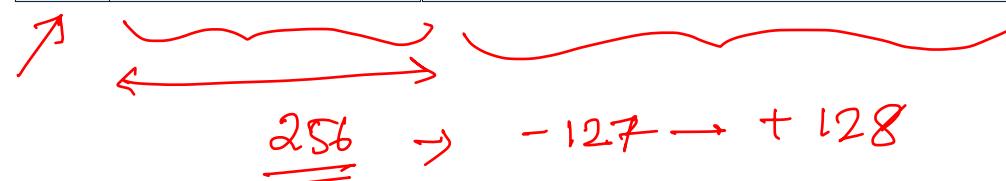
* General Principles

- * The **significand** is of the form : 1.xxxxx
- * No need to waste 1 bit representing (1.) in the significand
- * We can just save the **mantissa** bits
- * Need to also store the sign bit (S), exponent (X)

IEEE 754 Format - II

Sign(S) Exponent(X) Mantissa(M)

1	8	23
---	---	----



- * sign bit – 0 (+ve), 1 (-ve)
- * exponent, 8 bits
- * mantissa, 23 bits

Representation of the Exponent

- * Biased representation

- * bias = 127

- * $E = X + \underline{\text{bias}}$

- * Range of the exponent

- * $0 - 255 \leftrightarrow -127 \text{ to } +128$

- * Examples :

- * $X = 0, E = 127$

- * $X = -23, E = 104$

- * $X = 30, E = 157$

E

$$-127 \Rightarrow 0$$

$$-23 \Rightarrow -23 + 127$$

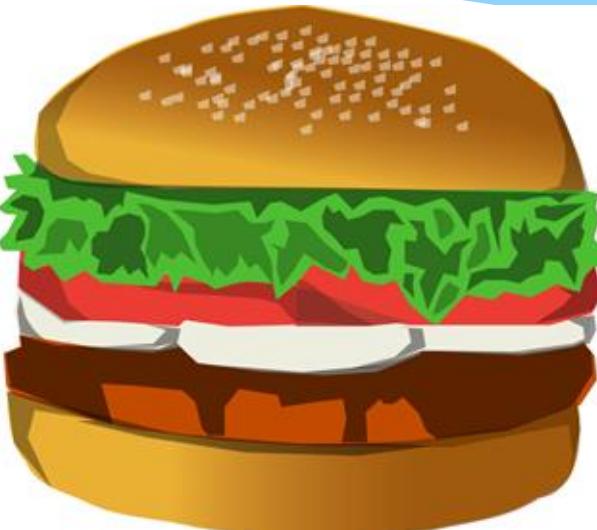
$$+128 \Rightarrow +128 + 127 = 255$$

Normal FP Numbers

- * Have an exponent between -126 and +127
- * Let us leave the exponents : -127, and +128 for special purposes.

$-127 \leftrightarrow +128$

$$A = \underbrace{(-1)^S * P * 2^{\frac{E}{-bias}}}_{(P=1+M, 0 \leq M < 1, X \in Z, 1 \leq E \leq 254)}$$



- * What is the largest +ve normal FP number ?
- * What is the smallest -ve normal FP number ?

$$\boxed{\pm (2 - 2^{-23}) \times 2^{127}}$$

Diagram illustrating the floating-point representation of a number:

S	8	23
0		111111

Annotations:

- Exp**: Exponent field, represented by the 8-bit binary string.
- M**: Mantissa field, represented by the 23-bit binary string.
- A green bracket underlines the mantissa, labeled $1 \cdot 111\dots1 \times 2^{127}$.
- A red double-headed arrow below the mantissa indicates its range from 0 to 1 (excluding 1).
- The formula for the number is given as:

$$= \sum_{i=0}^{-23} 2^i$$
- The final result is:

$$= 2^1 - 2^{-23}$$
- Another result is shown:

$$= (2 - 2^{-23})$$

Special Floating Point Numbers

E	M	Value
255	0 $\hat{=}$	∞ if $S = 0 \hat{=}$
255	0 ✓	$-\infty$ if $S = 1$
✓255 $\neq 0$		NAN(Not a number)
0	0 ✓	0
0	$\neq 0$	Denormal number

E
 $-127 > 0$
 $+128 = 255$

$1/0$ $5/0$

- * $\text{NAN} + x = \text{NAN}$

$$1/0 = \infty$$

$$M=0$$

- * $0/0 = \text{NAN}$

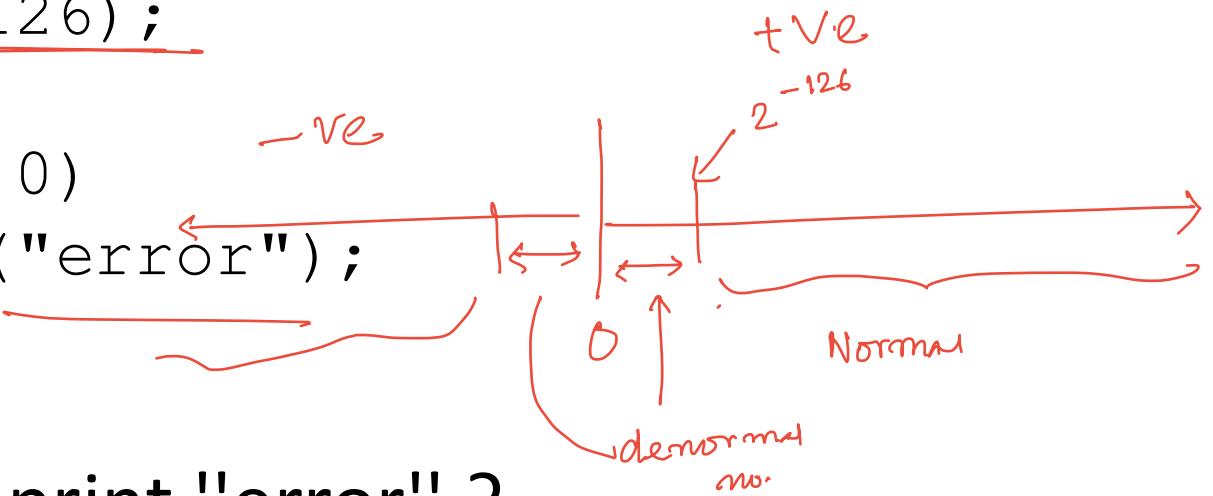
$$-1/0 = -\infty$$

- * $\sin^{-1}(5) = \text{NAN}$

$$\underline{E = -126}$$

Denormal Numbers

```
f = 2^(-126);  
g = f/2;  
if (g == 0)  
    print ("error");
```



- * Should this code print "error" ?
- * How to stop this behaviour ?

Denormal Numbers - II

$$A = \underbrace{(-1)^S * P * 2^{-126}}_{(P = 0 + M, 0 \leq M < 1)} \quad |$$

- * Significand is of the form : 0.xxxx
- * E = 0, X = -126 (why not -127?)
- * Smallest +ve normal number : 2^{-126}
- * Largest denormal number :
 - * $0.11\dots11 * 2^{-126} = (1 - 2^{-23}) * 2^{-126}$
 - * $= 2^{-126} - 2^{-149}$

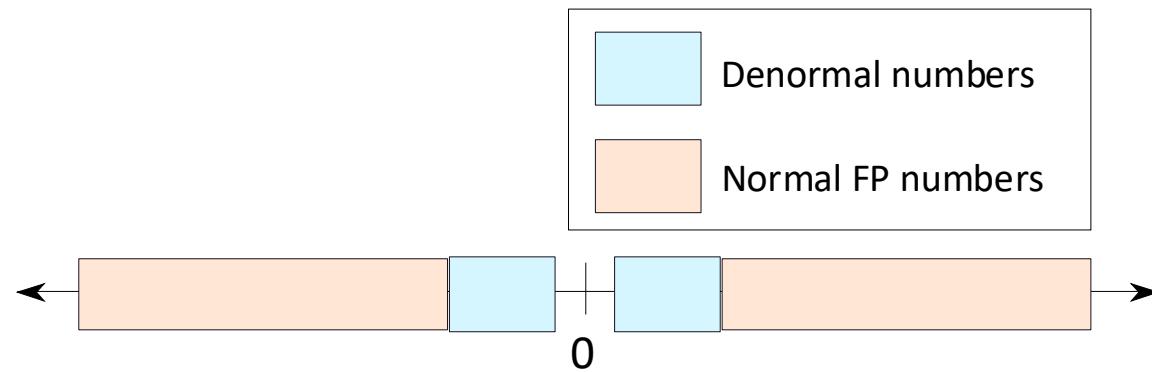
Example

Find the ranges of denormal numbers.

Answer

- For positive denormal numbers, the range is $[2^{-149}, 2^{-126} - 2^{-149}]$
- For negative denormal numbers, the range is $[-2^{-149}, -2^{-126} + 2^{-149}]$

Denormal Numbers in the Number Line



Extend the range of normal floating point numbers.

Double Precision Numbers

Field	Size(bits)
S	1
E	11
M	52

- Approximate range of **doubles**
 - $\pm 2^{1023} = \pm 10^{308}$
 - This is a lot !!!



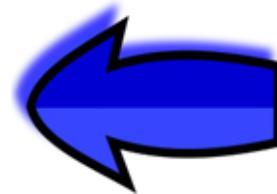
Floating Point Mathematics

```
A = 2^(50);  
B = 2^(10);  
C = (B+A) - A;
```

- * C will be computed to be 0
 - * There is no way of representing A+B in the IEEE 754 format
- * A **smart compiler** can reorder the operations to increase precision
- * Floating point math is **approximate**

Outline

- * Boolean Algebra
- * Positive Integers
- * Negative Integers
- * Floating Point Numbers
- * Strings



ASCII Character Set

- * ASCII – American Standard Code for Information Interchange
- * It has 128 characters
- * First 32 characters (control operations)
 - * backspace (8)
 - * line feed (10)
 - * escape (27)
- * Each character is encoded using 7 bits

ASCII Character Set

Character	Code	Character	Code	Character	Code
a	97	A	65	0	48
b	98	B	66	1	49
c	99	C	67	2	50
d	100	D	68	3	51
e	101	E	69	4	52
f	102	F	70	5	53
g	103	G	71	6	54
h	104	H	72	7	55
i	105	I	73	8	56
j	106	J	74	9	57
k	107	K	75	!	33
l	108	L	76	#	35
m	109	M	77	\$	36
n	110	N	78	%	37
o	111	O	79	&	38
p	112	P	80	(40
q	113	Q	81)	41
r	114	R	82	*	42
s	115	S	83	+	43
t	116	T	84	,	44
u	117	U	85	.	46
v	118	V	86	;	59
w	119	W	87	=	61
x	120	X	88	?	63
y	121	Y	89	@	64
z	122	Z	90	^	94

Unicode Format

- * **UTF-8 (Universal character set Transformation Format)**
 - * **UTF-8 encodes 1,112,064 characters** defined in the Unicode character set. It uses 1-6 bytes for this purpose.
E.g. ਅ ਆ ਕ ਖ, ਡ ମ ଙ ଲ
 - * UTF-8 is **compatible** with ASCII. The first 128 characters in UTF-8 correspond to the ASCII characters. When using ASCII characters, UTF-8 requires just one byte. It has a leading 0.
 - * Most of the languages that use variants of the Roman script such as French, German, and Spanish require 2 bytes in UTF-8. Greek, Russian (Cyrillic), Hebrew, and Arabic, also require 2 bytes.

UTF-16 and 32

- * **Unicode** is a standard across all browsers and operating systems
- * **UTF-8** has been superseded by UTF-16, and UTF-32
- * **UTF-16** uses 2 byte or 4 byte encodings (Java and Windows)
- * **UTF-32** uses 4 bytes for every character (rarely used)



THE END