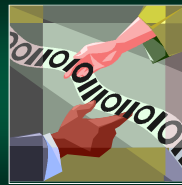




Set Theory in Computer Science

Part 1B



Binary Numbers

Bit of This and a Bit of That

What is a Number?

- We use the Hindu-Arabic Number System
 - positional grouping system
 - each position is a power of 10
- Binary numbers
 - based on the same system
 - use powers of **2** rather than 10
 - each digit is in the set $\{0, 1\}$



2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

3

Base 10 Number

The number **1783** is ...

10^4	10^3	10^2	10^1	10^0
10000	1000	100	10	1
0	1	7	8	3

$$1000 + 700 + 80 + 3 = 1783$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

4

Binary Number Example

The number **0110 1001** is ...

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	1	0	1	0	0	1

$$64 + 32 + 8 + 1 = 105$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

5

Binary Number Example

The number **1101 1011** is ...

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	1	0	1	1	0	1	1

$$128 + 64 + 16 + 8 + 2 + 1 = 219$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

6

Numbers are Tuples

- In Hindu-Arabic system, the order of the symbols is important – **so they are tuples**
- e.g. $123 \neq 321$
- Other number styles use sets – i.e. the ancient Egyptian system



2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

7

Looking at Numbers

- Numbers are tuples $1947 \neq 1974$
- Members of the decimals number are also members of the set $\{0, 1, 2, \dots, 9\}$

$1947 \rightarrow (1, 9, 4, 7)$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

8

Looking at Binary Numbers

- Binary numbers are tuples $10010100 \neq 11100000$
- Members of the binary number are also members of the set $\{0, 1\}$

$10100111 \rightarrow (1, 0, 1, 0, 0, 1, 1, 1)$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

9

Looking at Binary Numbers

- So, for a binary number **B**, all $x \in B$ holds the following: $x \in \{0, 1\}$

$10100111 \rightarrow (1, 0, 1, 0, 0, 1, 1, 1)$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

10

So....

$\{1776, 1846, 1947\} \rightarrow$
 $\{ (1, 7, 7, 6), (1, 8, 4, 6), (1, 9, 4, 7) \}$

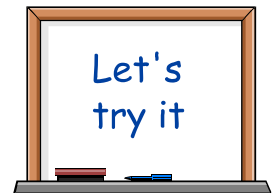
2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

11

Let's Make a Set-Based System

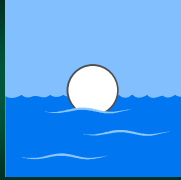
- We are mostly used to tuple-based number systems
- But, for most of history, people used sets
- Let's create one



2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

12

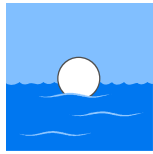


Floating Point Numbers

Real numbers are *real* complex

Floating Point Numbers

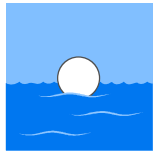
- Often, programs need to perform mathematics on *real* numbers
- Floating point numbers* are used to represent quantities that cannot be represented by integers



2/2/2018 Sacramento State - Cook - CSc 28 - Spring 2018 14

Floating Point Numbers

- Why?
 - regular binary numbers can *only* store *whole* positive and negative values
 - many numbers outside the range representable within the system's bit width (too large/small)



2/2/2018 Sacramento State - Cook - CSc 28 - Spring 2018 15

IEEE 754

- Practically modern computers use the *IEEE 754 Standard* to store floating-point numbers
- Represent by a mantissa and an exponent
 - similar to scientific notation
 - the value of a number is: $\text{mantissa} \times 2^{\text{exponent}}$
 - uses signed magnitude

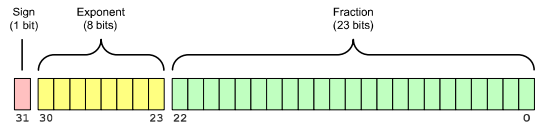
2/2/2018 Sacramento State - Cook - CSc 28 - Spring 2018 16

IEEE 754

- Comes in three forms:
 - single-precision: 32-bit
 - double-precision: 64-bit
 - quad-precision: 128-bit
- Also supports special values:
 - negative and positive *infinity*
 - and "not a number" for errors (e.g. 1/0)

2/2/2018 Sacramento State - Cook - CSc 28 - Spring 2018 17

IEEE 754 Single Precision (32 bit)



2/2/2018 Sacramento State - Cook - CSc 28 - Spring 2018 18

Fractional Field

- The fraction field number that represents part of the mantissa
- If a number is in proper scientific notation...
 - it always has a single digit before the decimal place
 - for decimal numbers, this is 1..9 (never zero)
 - for base-2 numbers, it is always 1

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

19

Fractional Field

- So, do we need to store the leading 1? It will always be a 1
- The fraction field, therefore...
 - only represents the fractional portion of a binary number
 - the integer portion is assumed to be 1
 - this increases the number of significant digits that can be represented (by not wasting a bit)

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

20

Exponent Field

- The exponent field supports negative and positive values but does not use sign-magnitude or 2's complement
- Uses a "biased" integer representation
 - fixed value is added to the exponent *before* storing it
 - when interpreting the stored data, this fixed value is then subtracted

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

21

Exponent Field

- Bias is different depending on precision
 - single precision: 127
 - double precision: 1023
 - quad precision: 16383
- For example, for single precision...
 - exponent of 12 stored as: $(+12 + 127) = 139$
 - exponent of -56 stored as: $(-56 + 127) = 71$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

22

Interpretation: Normal Case

- Exponent Field: not all 0's or all 1's
- Fraction Field: Any

$$\pm (1.\text{fraction}) \times 2^{(\text{exponent} - \text{bias})}$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

23

Interpretation: Zero

- Exponent Field: all 0's
- Fraction Field: all 0's

0

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

24

Interpretation: Tiny Numbers

- Exponent Field: all 0's
- Fraction Field: Any

$$\pm (0.\textit{fraction}) \times 2^{(1 - \textit{bias})}$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

25

Interpretation: Infinity

- Exponent Field: **All** 1's
- Fraction Field: 0

$$\pm \textit{infinity}$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

26

Interpretation: Invalid Numbers

- Exponent Field: **All** 1's
- Fraction Field: Not 0

Not a number (NaN)

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

27

Interpretation: Invalid Numbers

NaN → 1 / 0

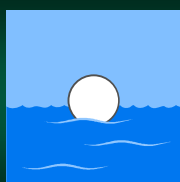
Naan →



2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

28



Let's Encode
Some
Numbers!

This is actually fun!

Something Else About Numbers...

The number **36.74** is ...

10^2	10^1	10^0	10^{-1}	10^{-2}
100	10	1	1/10	1/100
0	3	6	7	4

$$= (3 \times 10) + (6 \times 1) + 7/10 + 4/100$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

30

Binary Fractions!

5 and 3 / 8 is ...

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
16	8	4	2	1	1/2	1/4	1/8
0	0	1	0	1	0	1	1

$$= 4 + 1 + 1/4 + 1/8 = 101.011$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

31

Let's Encode 14.25 (32 bit)

- First, we need to convert 14.25 to its binary equivalent
- So, we need to calculate the integer part and fraction part of the number
- Everything after the decimal is a fraction
 - 0.25 is actually 25 / 100
 - we need to find the base 2 equivalent (1/4)

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

32

Step 1: Convert to binary

14 → 1110

0.25 → 1/4 → 0.01

binary 01 / 100

Hence:

14.25 → 1110.01

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

33

Step 2: Scientific Notation

- IEEE stores the data in scientific notation
- So we move the "binary point" over

1110.01 → 1.11001 × 2³

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

34

Step 2: Scientific Notation

- In binary scientific notation, the leading digit is always going to be 1
- Why store it? IEEE doesn't.
- Only data after the point is encoded

1.11001 × 2³ → (1 + .11001) × 2³

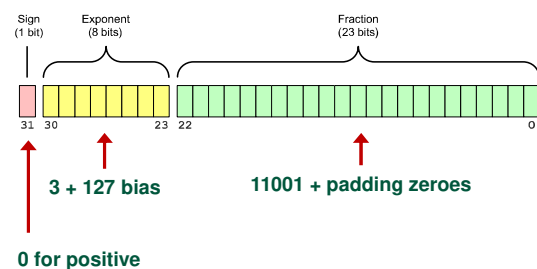
Fraction

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

35

Step 3: Encode



2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

36

Result: 14.25 (32 bit)

- The following is the encoded version of 14.25
- The rules are similar for double-precision

0 10000010 110010000000000000000000

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

37

Result: 14.25 (32 bit)

- We can also convert this into bytes
- Note: the floating point fields don't really "fit" cleanly into actual bytes

01000001 01100100 00000000 00000000
41 64 00 00

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

38

Example 2: Encode 13.75 (32 bit)

- First, we need to convert 13.75 to its binary equivalent
- So, we need to calculate the integer part and fraction part of the number
- Everything after the decimal is a fraction
 - 0.75 is actually 75 / 100
 - we need to find the base 2 equivalent (3/4)

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

39

Step 1: Convert to binary

13 → 1101

0.75 → 3/4 → 0.11

binary 11 / 100

Hence :

13.75 → 1101.11

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

40

Step 2: Scientific Notation

- IEEE stores the data in scientific notation
- So we move the "binary point" over

1101.11 → 1.10111 × 2³

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

41

Step 2: Scientific Notation

- In binary scientific notation, the leading digit is always going to be 1
- Why store it? IEEE doesn't.
- Only data after the point is encoded

1.10111 × 2³ → (1 + .10111) × 2³

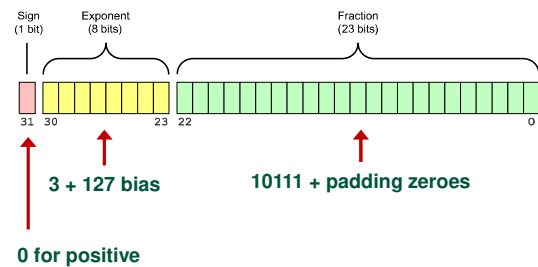
Fraction

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

42

Step 3: Encode



2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

43

Result: 13.75 (32 bit)

- The following is the encoded version of 13.75
- The rules are similar for double-precision

```
0 10000010 101110000000000000000000
```

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

44

Result: 13.75 (32 bit)

- We can also convert this into bytes
- Note: the floating point fields don't really "fit" cleanly into actual bytes

```
01000001 01011100 00000000 00000000
41 5C 00 00
```

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

45

More Single-Precision Examples

Zero:

```
0 00000000 000000000000000000000000
```

Positive Infinity:

```
0 11111111 000000000000000000000000
```

Negative Infinity:

```
1 11111111 000000000000000000000000
```

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

46



Tuples and Floats

Its all set theory, folks!

Floats Are Tuples

- Just like regular binary numbers, floating-point numbers of tuples
- They consist of three fields making them 3-tuples

```
(sign, exponent, fraction)
```

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

48

Encoding of 14.25

- Sign is a 1-tuple
- Exponent is a 8-tuple
- Fraction is a 23-tuple

0 10000010 110010000000000000000000

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

49

Set Notation of 14.25

((0),
 (1, 0, 0, 0, 0, 0, 1, 0),
 (1, 1, 0, 0, 1, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0))

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

50

Bit Vectors

Sets and Bits

Bit Vectors

- A *bit vector* is a way to store sets using bits
- Also known as a *bit array*, *bit set*, and *bit map*
- Compact format that can perform a set operations with a single operation (*fast!*)

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

52

Bit Vectors

- Each object in the universe is given a single bit in the bit array
- If the $x \in A$, then that bit is 1, otherwise 0
- Order is important, so this is a tuple approach

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

53

Example 1

- $U = \{ \text{fry, bender, farnsworth, leela, zoidberg} \}$
- $A = \{ \text{fry, bender, leela} \}$

$U = 11111$
 $A = 11010$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

54

Example 2

- $U = \{2, 3, 5, 7, 11, 13, 17, 19\}$
- $A = \{3, 5, 11, 19\}$

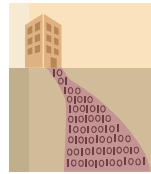
$U = 11111111$
 $A = 01101001$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

55

Why this is useful



- Computers can easily perform **and** & **or** operations on bytes (or multiple bytes)
- This means set operations can be performed amazingly fast

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

56

Let's look at the definitions again...

- The definitions of union and intersection are nearly identical
- The relationship between the elements is defined using an **and** or **or**

$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
 $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

57

Let's look at the definitions again...

- We can apply a **bit-wise-and** & a **bit-wise-or** to our bit array
- It will apply the operation to each of the bits in matching columns

$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
 $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

58

Let's look at the definitions again...

- So, each bit in **A** will be compared to its matching bit in **B**
- Bit match can do sets!

$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
 $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

59

Example: Union (using or)

$U = \{a, b, c, d, e, f, g\}$
 $A = 0111000 = \{b, c, d\}$
 $B = 0001110 = \{d, e, f\}$

0111000
 or 0001110
 0111110 = $\{b, c, d, e, f\}$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

60

Example: Intersection (using and)

$U = \{a, b, c, d, e, f, g\}$
 $A = 0111000 = \{b, c, d\}$
 $B = 0001110 = \{d, e, f\}$

$$\begin{array}{r} 0111000 \\ \text{and } 0001110 \\ \hline 0001000 = \{d\} \end{array}$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

61

Complement

- Then, how do we do a complement of a set A ?
- We must flip all the bits from 1 to 0, and 0 to 1
- We can use a *binary-not* or the *XOR* operation

$$A' = \{x \mid x \notin A\}$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

62

Java/C Code

`&&` and `||` are Boolean.
These are bit-wise.

Intersection : `a & b`
Union : `a | b`
Complement : `~a`

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

63

Exclusion

- Finally, how do we do set difference?
- The subtract operator will *not* work
- Let's look at the definition a bit more closely

$$A - B = \{x \mid x \in A \text{ and } x \notin B\}$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

64

Exclusion

- It's essentially the definition of *intersection*
- Except, the second operand is the definition of *complement*.

$$A - B = \{x \mid x \in A \text{ and } x \notin B\}$$

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

65

Java/C Code

Just complement the second operand

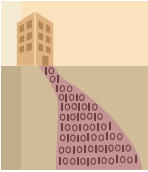
Exclusion : `a & ~b`

2/2/2018

Sacramento State - Cook - CSc 28 - Spring 2018

66

Limits of Bit Vectors



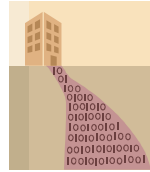
- Bit vectors, while useful, do have some notable limitations
- They only work on finite, countable sets
- For all other cases, you will have to work use a more advanced ADT

2/2/2018

Sacramento State - Cook - CS& 28 - Spring 2018

67

CSC 130 is waiting for you!



- For most cases, a very sophisticated list or tree can be used
- You will need to know:
 - lists / trees
 - sorting
 - binary-searches
 - Big-O

2/2/2018

Sacramento State - Cook - CS& 28 - Spring 2018

68