CPSC 121: Models of Computation

Unit 3

Representing Numbers and Other Values in a Computer

Based on slides by Patrice Belleville and Steve Wolfman

Pre-Class Learning Goals

- By the start of this class you should be able to
  - Convert unsigned integers from decimal to binary and back.
  - Take two's complement of a binary integer.
  - Convert signed integers from decimal to binary and back.
  - Convert integers from binary to hexadecimal and back.
  - Add two binary integers.

Quiz 3 Feedback

- Overall:
- Issues:

- Clock Arithmetic and Can you be 1/3rd Scottish?
- We will get back to these questions soon.

In-Class Learning Goals

By the end of this unit, you should be able to:

- Critique the choice of a digital representation scheme, including describing its strengths, weaknesses, and flaws (such as imprecise representation or overflow), for a given type of data and purpose, such as
  - fixed-width binary numbers using a two's complement scheme for signed integer arithmetic in computers
  - hexadecimal for human inspection of raw binary data.

Where We Are in The Big Stories

Theory:

- How do we model computational systems?

Now:

- Showing that our logical models can connect smoothly to models of number systems.

Hardware:

- How do we build devices to compute?

Now:

- Enabling our hardware to work with numbers.
  - And once we have numbers, we can represent pictures, words, sounds, and everything else!

Motivating Problem

Photo by Philippe Semanaz (CC by/sa)

- Understand and avoid cases like those at: http://www.ima.umn.edu/~arnold/455.f96/disasters.html
- Death of 28 people caused by failure of an anti-missile system, caused in turn by the misuse of one representation for fractions.
- Explosion of a $7 billion space vehicle caused by failure of the guidance system, caused in turn by misuse of a 16-bit signed binary value.
- (Both representations are discussed in these slides.)

Outline

- Unsigned and signed binary integers.
- Characters.
- Real numbers.

Representing Numbers

- We can choose any arrangement of Ts and Fs to represent numbers...
- If we use F -> 0 and T->1, the representation is a true base 2 numbers:

Number	V1	V2	V3	V4
0	F	F	F	F
1	F	F	F	T
2	F	F	T	F
3	F	F	T	T
4	F	T	F	F
5	F	T	F	T
6	F	T	T	F
7	F	T	T	T
8	T	F	F	F
9	T	F	F	T

Number	b_3	b_2	b_1	b_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Representing Unsigned Integers

- The binary value

$$b_{n-1}b_{n-2}\ldots b_2 b_1 b_0$$

- represents the integer

$$b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \ldots + b_2 2^2 + b_1 2^1 + b_0$$

or written differently

$$\sum b_i 2^i$$

where each b_i is either 0 or 1

Number Systems

- A number can be written using any base.
- What can you say about the following numbers?

100001_2

33_{10}

1020_3

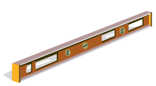
21_{16}

A. First number is different than the rest
B. Second number is different than the rest
C. Third number is different than the rest
D. All are the same number
E. All are different numbers

Clock Arithmetic

- Problem: It's 0500h. How many hours until midnight?

- Give an algorithm that requires a 24-hour clock, a level, and no arithmetic.



Clock Arithmetic

- 0500 is five hours from midnight.
- 1900 is five hours to midnight.

- 5 and 19 are "additive inverses" in clock arithmetic: 5 + 19 = 0.

- So. what is the additive inverse of 15?
  - How did you find it?

  - How else can you write 15?

That's even true for 12. Its additive inverse is itself!

Clock Arithmetic

If we wanted negative numbers on the clock, we'd probably put them "across the clock" from the positives.

After all, if 3 + 21 is **already** 0, why not put -3 where 21 usually goes?

Open-Ended Quiz # Question

- Suppose it is 15:00 (3:00PM, that is). What time was it 8* 21 hours ago?
  - Don't multiply 21 by 8!
  - Must not use numbers larger than 24 in your calculation'

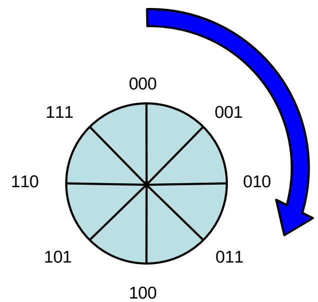
A. 21:00
B. 4:00
C. 8:00
D. 15:00
E. None of these

Open-Ended Quiz # Question

- Suppose it is 15:00 (3:00PM, that is). What time will it be 13 * 23 hours from now?
  - Don't multiply 13 by 23!

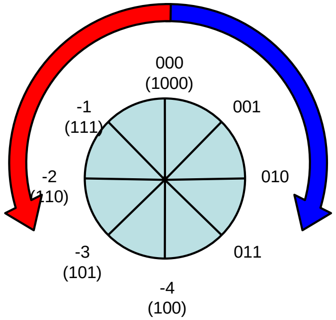
A. 13:00
B. 2:00
C. 22:00
D. 15:00
E. None of these

Unsigned Binary Clock



- Here's a 3-bit unsigned binary clock, numbered from 0 (000) to 7 (111).

Crossing the Clock



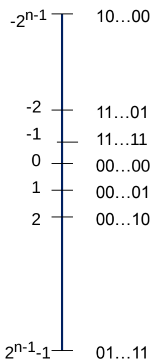
- To "cross the clock", go as many ticks left from the top as you previously went right from the top.
- Here's a clock labelled with 0 (000) to 3 (011) and -1 (111) to -4 (100).

Two's complement for Negative Integers

- In a 3-bit signed numbers:
  - 2^3 is the same as 000
  - the negation of a number x is the number $2^3 - x$.
- Why does this make sense?
  - For 3-bit integers, what is 111 + 001?
    - A. 110
    - B. 111
    - C. 1000
    - D. 000
    - E. Error: we can not add these two values.

Two's Complement for n-Bit Integers

- Unfolding the clock into a line:



- Also note that
  - first bit determines the sign
    - 0-positive, 1- negative
  - $-x$ has the same binary representation as $2^n - x$
- And

$$2^n - x = \left(2^n - 1 - x\right) + 1$$

Add 1

Flip bits from 0 to 1 and from 1 to 0

Exercice

- What is 10110110 in decimal, assuming it's a signed 8-bit binary integer?
  - A. 182
  - B. -74
  - C. 74
  - D. -182
  - E. None of the above

Converting Decimal to Binary

- How do we convert a positive decimal integer n to binary?
  - The last bit is 0 if n is even, and 1 if n is odd.
  - To find the remaining bits, we divide n by 2, ignore the remainder, and repeat.
- What do we do if n is negative?
- Examples:

Converting Binary to Decimal

- **Theorem:** for signed integers, the binary value

  $$b_{n-1}b_{n-2}...b_2b_1b_0$$

  represents the integer

  $$-b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + ... + b_2 2^2 + b_1 2^1 + b_0$$

  or written differently

  $$-b_{n-1}2^{n-1} + \sum b_i 2^i$$

- Examples:

Summary Questions

- With n bits, how many distinct values can we represent?
- What are the smallest and largest n-bit unsigned binary integers?
- What are the smallest and largest n-bit signed binary integers?
- Why are there more negative n-bit signed integers than positive ones?

More Summary Questions

- How do we tell quickly if a unsigned binary integer is negative, positive, or zero?
- How do we tell quickly if a signed binary integer is negative, positive, or zero?
- How do we negate a signed binary integer?
- There is one signed n-bit binary integer that we should not try to negate.
  - Which one?
  - What do we get if we try negating it?

Outline

- Unsigned and signed binary integers.
- Characters.
- Real numbers.

Representing Characters

- How do computers represent characters?
  - They use sequences of bits (like for everything else).
  - Integers have a “natural” representation of this kind.
  - There is no natural representation for characters.
  - So people created arbitrary mappings:
    - EBCDIC: earliest, now used only for IBM mainframes.
    - ASCII: American Standard Code for Information Interchange
      - 7-bit per character, sufficient for upper/lowercase, digits, punctuation and a few special characters.
    - UNICODE:
      - 16+ bits, extended ASCII for languages other than English

Representing Characters

- What may the binary value 11111000 represent?
  - A. -8
  - B. The character ø
  - C. 248
  - D. More than one of the above
  - E. None of the above.

Outline

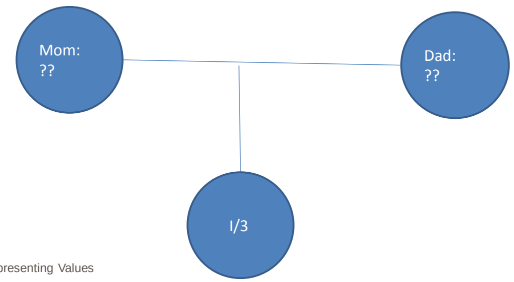
- Unsigned and signed binary integers.
- Characters.
- Real numbers.

Can you be 1/3rd Scottish?

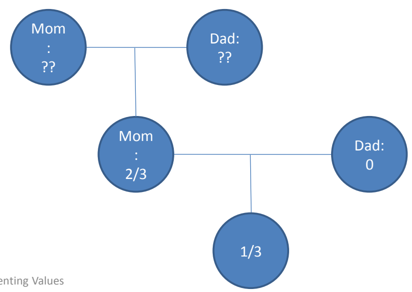
- To build a model, we must clearly specify the problem. Many problems admit multiple models that lead to fundamentally different results.
- We're going to use the model of parentage "endowing" 50% of each parent's "ish-ness". That's a coarse, even silly model. (By that model, none of us are Canadian, since humans did not originate in Canada.)
- Our model is handy for us, but it's not necessarily what people's identity is about!
- **Our model will help us understand how real numbers are represented in binary.**

Can you be 1/3rd Scottish?

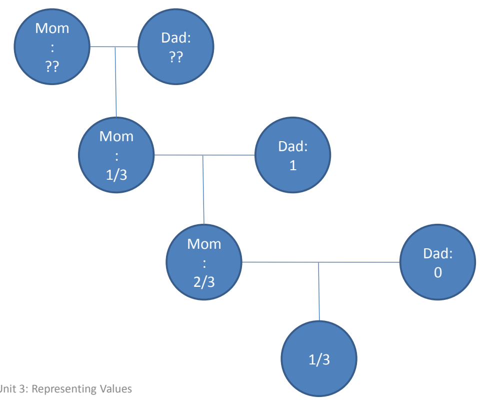
- Suppose we use the model in which a child gets 50% of each parent's nationality.
  - Focus on Mom (and Mom's Mom and so on).
  - We'll just make Dad "Scot" or "Not" as needed at each step.

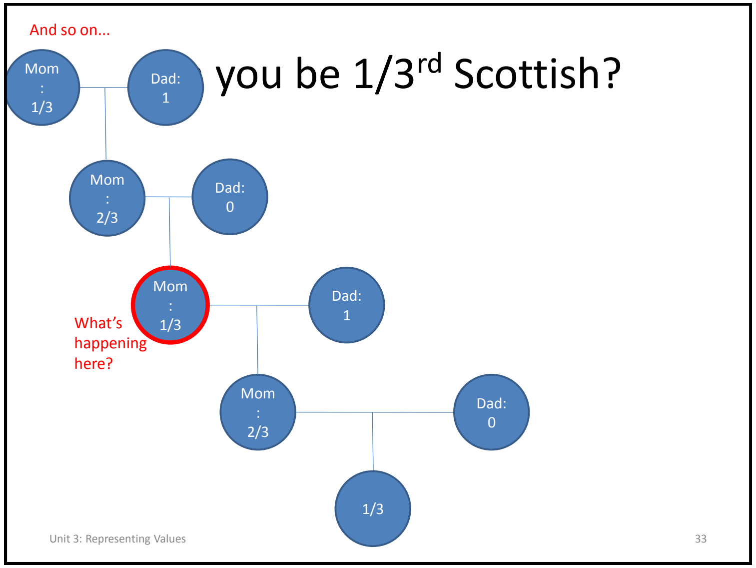


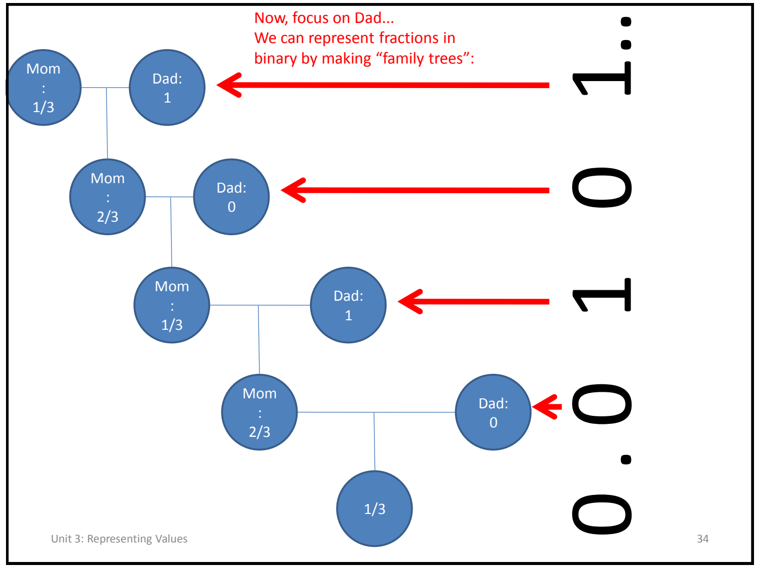
Can you be 1/3rd Scottish?

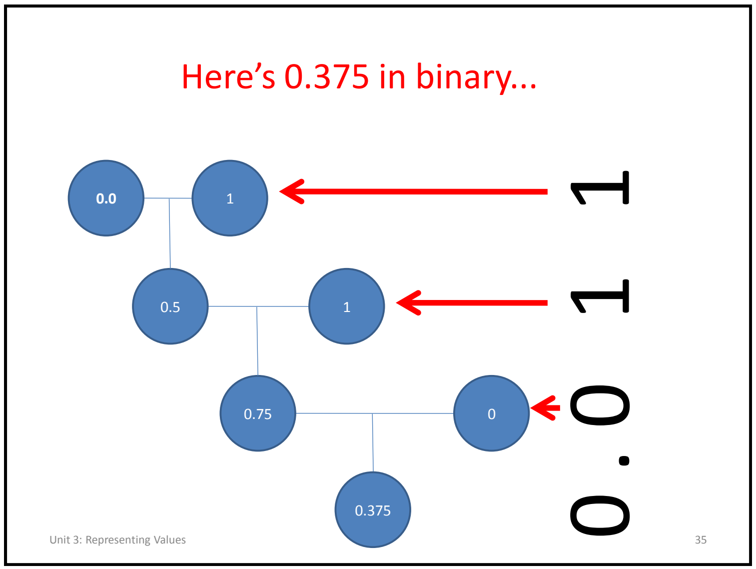


Can you be 1/3rd Scottish?









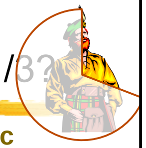
Representing Decimal Values

- Numbers with fractional components in binary:
  - Example: 5/32 = 0.00101
- Which of the following values have a finite binary expansion?
  - A. 1/10
  - B. 1/3
  - C. 1/4
  - D. More than one of the above.
  - E. None of the above.

Representing Decimal Values

- Numbers with fractional components (cont):
  - In decimal:
    - 1/3 = 0.33333333333333333333333333333333...
    - 1/8 = 0.125
    - 1/10 = 0.1
  - In binary:
    - 1/3 =
    - 1/8 =
    - 1/10 =
- Which fractions have a finite decimal expansion?

So... Computers Can't Represent 1/3?



- No! Using a different scheme (e.g. a **Sympolic Repesentation** like a rational number with a separate integer numerator and denominator), computers can perfectly represent 1/3!
- You know that from Racket!

- The point is: Numerical representations that use a finite number of bits have weaknesses.
  - Java's regular representation
  - Scheme/Racket's inexact numbers (#i) .
- We need to know those weaknesses and their impact on your computations!

Finite Representation of Decimal Values

- Consequences:
  - Computations involving floating point numbers are imprecise.
    - The computer does not store 1/3, but a number that's very close to 1/3.
    - The more computations we perform, the further away from the "real" value we are.
  - Example: predict the output of:

```
(* #i0.01 0.01 0.01 100 100 100)
```


Finite Representation of Decimal Values

- Consider the following:

```
(define (addDimes x)
  (if (= x 1.0)
    0
    (+ 1 (addfractions (+ x #i0.1)))))
```

- What output will (addDimes 0) produce?
  - A. 10
  - B. 11
  - C. More than 11
  - D. No value will be printed
  - E. None of the above

But Racket solves this…right?

Racket's regular number representation scheme is flexible and powerful (but maybe not as "efficient" or "compact" as Java's).

But no representation is perfect, and every representation could be *more* flexible.

```
;; What do these evaluate to?
(+ 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1)
(* (sqrt 2) (sqrt 2))
```


Machine Languages

- As you learned in CPSC 110, a program can be
  - **Interpreted**: another program is reading your code and performing the operations indicated.
    - Example: Scheme/Racket
  - **Compiled**: the program is translated into **machine language**. Then the machine language version is executed directly by the computer.

Machine Instructions

- What does a machine language instruction look like?
  - It is a sequence of bits!
  - Y86 (CPSC 213,313)example: adding two values.
    - In human-readable form: addl %ebx, %ecx.
    - In binary: 0110000000110001

%ecx

%ebx

Addition

Arithmetic operation

Machine Instructions and Hex Notation

- Long sequences of bits are painful to read and write, and it's easy to make mistakes.
- Should we write this in decimal instead?
  - Decimal version: 24577.
  - Problem: We can not tell what operation this is.
- Solution: use hexadecimal 6031

%ecx

%ebx

Addition

Arithmetic operation

Problem: Binary Code

```
0000 0059 b200 0212 03b6 0004 b200 0214
0005 b600 0710 643c 8401 011b 9dff fcb2
0002 1208 b600 04b2 0002 bb00 0959 b700
0a12 0bb6 000c 1bb6 000d b600 0eb6 0004
b200 02bb 0009 59b7 000a 120f b600 0c10
641b 64b6 000d b600 0eb6 0004 b100 0000
0200 1500 0000 2600 0900 0000 0600 0800
0700 1100 0c00 1400 0f00 1700 1000 1b00
1200 2300 1300 3c00 1400 5800 1500 1800
0000 0600 01fc 0014 0100 0100 1900 0000
0200 1a
```

Why would the task of finding a particular instruction in a memory-dump file be much more difficult if the file were represented in **binary**?

a. Because we would have to translate all the instruction codes and values to binary.
b. Because many memory-dump files would have no binary representation.
c. Because the binary representation of the file would be **much** longer.
d. Because data like 1100100 (100 in base 2) might not show up as the sequence of numbers 1 1 0 0 1 0 0.
e. It wouldn’t be much more difficult.

Problem: Decimal Code

```
0000 0059 b200 0212 03b6 0004 b200 0214
0005 b600 0710 643c 8401 011b 9dff fcb2
0002 1208 b600 04b2 0002 bb00 0959 b700
0a12 0bb6 000c 1bb6 000d b600 0eb6 0004
b200 02bb 0009 59b7 000a 120f b600 0c10
641b 64b6 000d b600 0eb6 0004 b100 0000
0200 1500 0000 2600 0900 0000 0600 0800
0700 1100 0c00 1400 0f00 1700 1000 1b00
1200 2300 1300 3c00 1400 5800 1500 1800
0000 0600 01fc 0014 0100 0100 1900 0000
0200 1a
```

Why would the task of finding a particular instruction in a memory-dump file be much more difficult if the file were represented in **decimal**?

a. Because we would have to translate all the instruction codes and values to decimal.
b. Because many memory-dump files would have no decimal representation.
c. Because the decimal representation of the file would be **much** longer.
d. Because data like 100 might not show up as the sequence of numbers 1 0 0.
e. It wouldn’t be much more difficult.

Quiz 4

- The 4th online quiz is due ______________________
- Assigned reading for the quiz:
  - Epp, 4th edition: 2.3
  - Epp, 3rd edition: 1.3
  - Rosen, 6th edition: 1.5 up to the bottom of page 69.
  - Rosen, 7th edition: 1.6 up to the bottom of page 75.