Data Representation

Interpreting bits to give them meaning

Part 2: Hexadecimal and Practical Issues

Notes for CSC 100 - The Beauty and Joy of Computing The University of North Carolina at Greensboro

Class Reminders

For this week:

- · Assignment 1 due Friday (10:00am)
- Review Lab 3 solutions (in Blackboard)
- Do the Pre-Lab reading for Lab 4 (really!)

For the not-so-distant future:

• Blown to Bits Chapter 2 - reflection due Tues, Sept 17 (10:00am)

From Last Time...

Key points from "Data Representation, Part 1":

- · A number is an abstract idea
- Anything you can point at or write down is a *representation* of a number
- · Lots of different representations for the same number:
 - Written in decimal notation (what we're most familiar with)
 - Written in roman numerals (e.g., 6 is the same as VI)
 Written as a set of "tick marks" (e.g., 6 is the same as IIIIII)
 Written in binary (e.g., 6 is the same as 1102)

 - As a sequence of voltages on wires
- Computers work with binary because switches are off or on (0 or 1)
- Converting between number bases doesn't change the number, just chooses a different representation

_		

Hexadecimal - another useful base

Hexadecimal is base 16.

How do we get 16 different digits? Use letters!

Hexadecimal digits (or "hex digits" for short): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Counting - now our odometer has 16 digits:

$0_{16} (= 0_{10})$	6 ₁₆ (= 6 ₁₀)	C ₁₆ (= 12 ₁₀)	12 ₁₆ (= 18 ₁₀)	
1 ₁₆ (= 1 ₁₀)	7 ₁₆ (= 7 ₁₀)	D ₁₆ (= 13 ₁₀)	13 ₁₆ (= 19 ₁₀)	
2 ₁₆ (= 2 ₁₀)	8 ₁₆ (= 8 ₁₀)	E ₁₆ (= 14 ₁₀)	14 ₁₆ (= 20 ₁₀)	
3 ₁₆ (= 3 ₁₀)	9 ₁₆ (= 9 ₁₀)	F ₁₆ (= 15 ₁₀)	15 ₁₆ (= 21 ₁₀)	
$4_{16} (= 4_{10})$	A ₁₆ (= 10 ₁₀)	10 ₁₆ (= 16 ₁₀)	16 ₁₆ (= 22 ₁₀)	
5 ₁₆ (= 5 ₁₀)	B ₁₆ (= 11 ₁₀)	11 ₁₆ (= 17 ₁₀)	17 ₁₆ (= 23 ₁₀)	

Hexadecimal/Decimal Conversions

Conversion process is like binary, but base is 16

 $\underline{\textit{Problem 1}}$: Convert 423₁₀ to hexadecimal: 423/16 = quotient 26, remainder 7 (=7₁₆) 26/16 = quotient 1, remainder 10 (=A₁₆) 1/16 = quotient 0, remainder 1 (=1₁₆)

Reading digits bottom-up: 423₁₀ = 1A7₁₆

<u>Problem 2</u>: Convert 9C3₁₆ to decimal: Start with first digit, 9 9*16 + 12 = 156 156*16 + 3 = 2499

Hex Digit List

0₁₆ = 0₁₀ 1₁₆ = 1₁₀ 2₁₆ = 2₁₀ 3₁₆ = 3₁₀ 4₁₆ = 4₁₀ 5₁₆ = 5₁₀ 6₁₆ = 6₁₀ 7₁₆ = 7₁₀ 8₁₆ = 8₁₀

 $9_{16} = 9_{10}$ $9_{16} = 9_{10}$ $A_{16} = 10_{10}$ $B_{16} = 11_{10}$ $C_{16} = 12_{10}$ $D_{16} = 13_{10}$

E₁₆ = 14₁₀ F₁₆ = 15₁₀

• Therefore, 9C3₁₆ = 2499₁₀

Hexadecimal/Decimal Conversions

Conversion process is like binary, but base is 16

<u>Problem 1</u>: Convert 423₁₀ to hexadecimal: 423/16 = quotient 26, remainder 7 (=7₁₆) 26/16 = quotient 1, remainder 10 (=A₁₆) 1/16 = quotient 0, remainder 1 (=1₁₆)

Reading digits bottom-up: 423₁₀ = 1A7₁₆

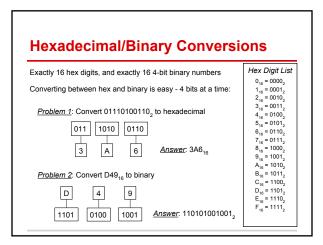
Problem 2: Convert 9C3₁₆ to decimal: Start with first digit, 9 9*16 + 12 = 156 156*16 + 3 = 2499

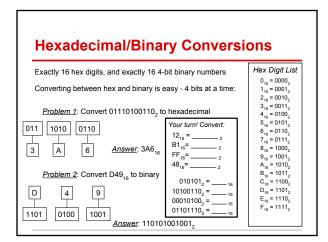
Therefore, 9C3₁₆ = 2499₁₀

Your turn! Convert: 103₁₀ = _ 247 10 = _ ___ 16

95210 = _ 3C₁₆ = _ - 10 B9₁₆ = _ - 10 357₁₆ = _ - 10 $\begin{array}{c} 0_{16} = 0_{10} \\ 1_{16} = 1_{10} \\ 2_{16} = 2_{10} \\ 3_{16} = 3_{10} \\ 4_{16} = 4_{10} \\ 0_{16} = 6_{10} \\ 0_{16} = 6_{10} \\ 0_{16} = 6_{10} \\ 0_{16} = 10_{10} \\ 0_{16} = 10_{10} \\ 0_{16} = 13_{10} \\ 0_{16} = 13_{10} \\ 0_{16} = 13_{10} \\ 0_{16} = 14_{10} \\ 0_{16} = 14_{10} \end{array}$ F₁₆ = 15₁₀

Hex Digit List





Use of hexadecimal in file dumps Binary is a very long format (8 bits per byte), but often data files only make sense as binary data. Hexadecimal is great for this - simple one-to-one correspondence with binary, and more compact. Sample "file dump":2011:07:14 1 5:09:27.!....

The same data, showing character representation

Remember	
Don't get lost in the details and manipulations:	
Any base is a representation of an abstract number	
We are interested in working with the number, and computations are not "in a base" - the base is only useful for having it make sense to us or the computer	
Practice!	
You should be able to convert from one base to another.	
Lots of ways to practice: • By hand: Pick a random number convert to binary and convert back - did you get the same value? • This isn't foolproof: You could have made two mistakes!	
With a calculator: Many calculators (physical and software) do base conversion - check your randomly selected conversions.	
With a web site: Several web sites provide says to practice For example, see http://cs.iupui.edu/~aharris/230/binPractice.html	
Practical Issues with Numbers Finite Length Integers	
Question (a little contrived):	
If a CPU has 4 single-bit storage locations for each number, what happens when you add:	
1111 ₂ + 0001 ₂ = ₂	
1111 ₂ + 0001 ₂ =	

Practical Issues with Numbers

Finite Length Integers

Question (a little contrived):

If a CPU has 4 single-bit storage locations for each number, what happens when you add:

Answer Part 1: If you did this on paper, you'd get 100002

Which leads to another question:

How do we store 5 bits when there are only storage locations for 4 bits?

Practical Issues with Numbers

Finite Length Integers

Question (a little contrived):

If a CPU has 4 single-bit storage locations for each number, what happens when you add:

Answer Part 1: If you did this on paper, you'd get 10000,

Which leads to another question:

How do we store 5 bits when there are only storage locations for 4 bits?

Answer Part 2: What CPUs do is throw out the 5th bit, storing 0000, Which means: To a 4-bit computer, 15 + 1 = 0

Practical Issues with Numbers

Finite Length Integers

On real computers:

- This happens, but with 32-bit numbers or 64-bit numbers instead of 4.
 When things "wrap around" it actually goes to negative values...
 On a 32-bit CPU: 2,147,483,647 + 1 = -2,147,483,648

However: Some programming languages/systems support numbers larger than the hardware, by using multiple memory locations.

Let's try this!

Practical Issues with Numbers Finite Length Integers In C: x = 1000*1000*1000*1000 print x Outputs: Outputs: -727379968 1000000000000 -727379968 **Practical Issues with Numbers** Finite Length Integers In C: In Python: x = 1000*1000*1000*1000 print x int val=1000*1000*1000*1000; printf("%d\n", val); Outputs: Outputs: Outputs: -727379968 1000000000000 -727379968 First thought: Python is cool! Second thought: Don't expect something for nothing... Let's do something pretty useless (that takes a lot of integer operations) Problem: Compute the last 6 digits of the billionth Fibonacci number **Practical Issues with Numbers** Finite Length Integers In C: In Java: In Python: x = 1000*1000*1000*1000 print x int val = 1000*1000*1000*1000; System.out.println(val); Outputs: Outputs: 1000000000000 -727379968 -727379968 First thought: Python is cool! Second thought: Don't expect something for nothing... Let's do something pretty useless (that takes a lot of integer operations) Problem: Compute the last 6 digits of the billionth Fibonacci number In C: In Java: In Python: 3.5 seconds 3.4 seconds 3 minutes. 56.2 seconds Times on my laptop: Intel i7-3740QM (2.7GHz)

Practical Issues with Numbers Finite Precision Floating Point Question: How do you write out 1/3 in decimal? Answer: 0.33333333333.... Observation: Impossible to write out exactly with a finite number of digits The same holds in binary! Can be written exactly Cannot be written exactly 0.5 = 0.12 1/3 = 0.0101010101...₂ 0.25 = 0.01, % = 0.001100110011..., 0.375 = 0.011, 1/10 = 0.0001100110011..., Imagine: How hard is it to write banking software when there is no finite representation of a dime (0.10 dollars)?!?!? Solutions people came up with: Work with cents (integers!) or special codings (BCD=Binary Coded Decimal) **Practical Issues with Numbers** Finite Precision Floating Point Question: How do you write out 1/3 in decimal? Bottom Line: Observation: There are a lot of subtle problems with numbers that go beyond the level of study in CSC 100 Can be These issues *usually* don't come up. 0.25 = But... when they matter, they can matter a LOT. 0.375 = For now: Be aware what the issues are. Imagine: Hov For a later class: Understand the details. Solutions people came up with: Work with cents (integers!) or special codings (BCD=Binary Coded Decimal) **Still More Data Representation for Later** Now we know all about representing numbers But computers also deal with text, web pages, pictures, sound/music, video, ... How does that work?