Lecture 04:
Models and Numbers

Sierra College
CSCI-12
Spring 2015
Weds 02/04/15

Announcements

- **Schedule**
  - Spring Add/Drop/Refund deadline is THIS Sunday 2/8
    - If no assignments are submitted by this deadline, I will consider this as a no-continue decision on your part, and will instructor-drop you from the course
- **Past due assignments**
  - HW02: Canvas Intro, accepted thru Tues 2/10 @ 11pm (do ALL 3 parts!)
  - LAB02: Hello World, accepted thru Tues 2/10 @ 11pm
- **Current assignments**
  - HW03: Why Code, due Fri 2/6 @ 11pm
- **New assignments**
  - LAB04: Hello Again, due Tues 2/10 @ 11pm
    - We went thru this in class on Monday
    - Lab time today to complete this
    - Refactor Hello World into a more O-O structure

Lecture Topics

- **Last time**:
  - Hello World recap
  - Demo on refactoring Hello World into object-oriented version
- **Today**:
  - Models
  - Number systems
  - Number conversions

Why Do We Care About Computers?

- Because they are absolutely pervasive in modern-day society, and central to our 21st century economy!
- Science, technology, defense, communications, utilities, commerce, finance, transportation, farming, manufacturing, education, entertainment, …
  - Can you name a field today that DOES NOT involve computers??
- **But**, for all their importance, computers are:
  - Blindingly fast, but yet also in some sense… “stupid”
  - Only able to do what they’ve been instructed to do
  - Only able to understand specific instructions, expressed exclusively in patterns of 1s and 0s
  - This is **programming**: but, what is it??

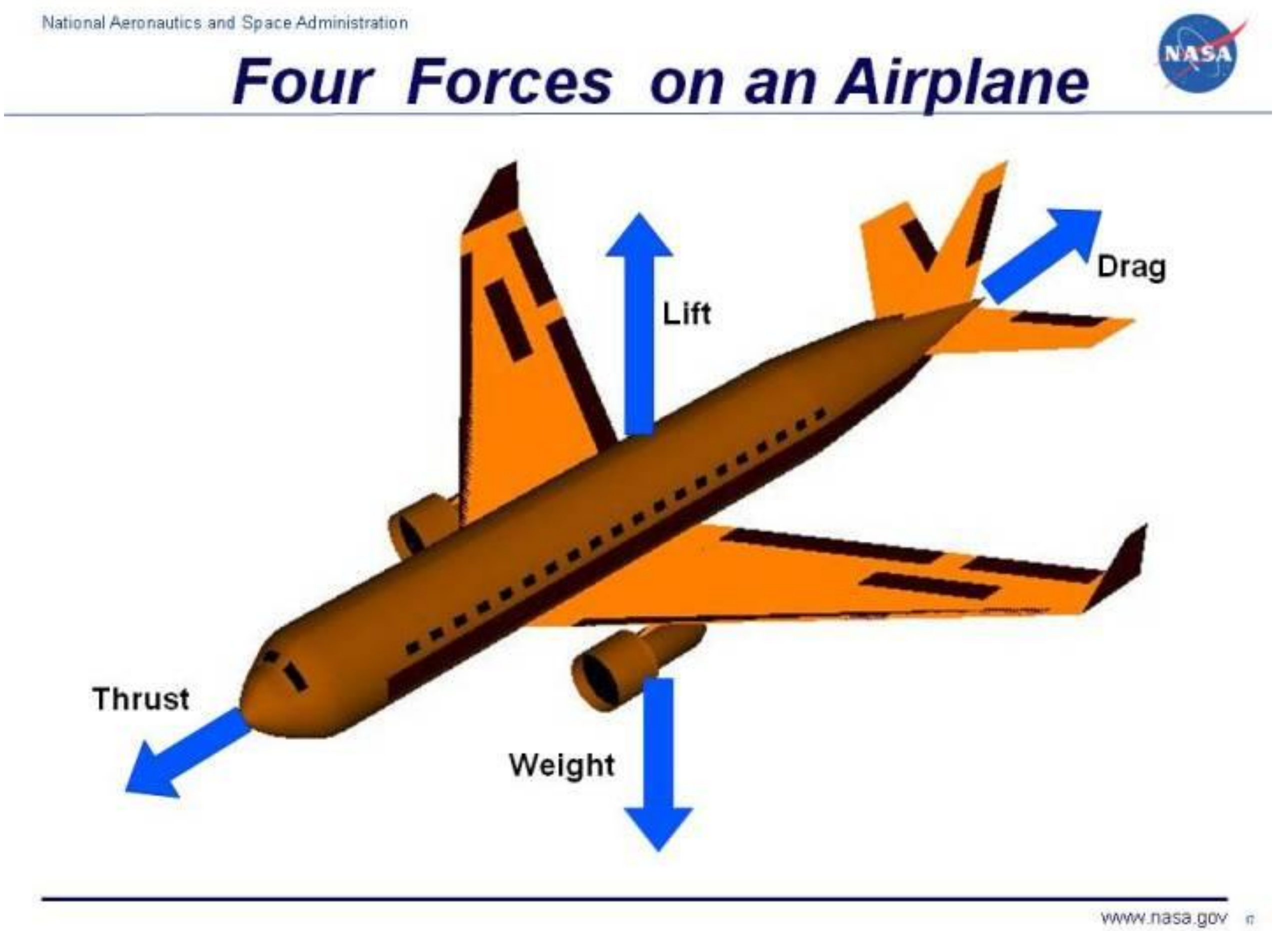
What Is Programming?

- Programming is the science/art of describing, in very specific terms, WHAT you want done, in such terms that the computer itself can understand and execute
- To some extent, to program a computer effectively, we must have some working **understanding of a computer on its own terms**
- Two ways of doing this (for us):
  - Models
  - Numbers

What's a Model?

- **Models** are just one tool in a software developer's toolkit
- A model is an **abstraction** (or a **simplification**, or a **generalization**) of reality
- Models are a big-picture understanding of some system, with some details sacrificed for the larger goal of more clarity and better understanding
- Models describe general, or idealized behavior, without getting too bogged down by all the details
- Models are frequently expressed visually, via diagrams
- ***"Everything should be made as simple as possible, but no simpler"*** - often attributed to Albert Einstein

Example: Model of an Aircraft



Examples of Models

- Physics/engineering: free-body diagrams, circuit diagrams
- Engineering simulation: wind tunnels, anechoic chambers, shaker tables
- Chemistry: complex molecules as “balls and sticks” (atoms and bonds)
- Meteorology: weather forecasting from atmospheric models
- Economics: economic models
  - (People win Nobel prizes for this stuff!)
- Ecology: predator-prey population models
- And on and on and on…
  - http://en.wikipedia.org/wiki/Model (see Canvas lecture module)

Computing Models

- There are a handful of models which are most pertinent to our study of programming
- These appear at various places throughout your textbook
  - Hopefully they are familiar from CS-10
- These models are*:
  - The standard computer model
  - The layered computer model
  - The memory model
  - The black box model
  - The UML Class model

* There may be multiple alternate terms for such models, but we'll use these terms

Standard Computer Model

- First off, there is no "standard" version of this!
  - This is the version shown in your textbook
  - MANY other depictions exist, see Canvas lecture module links for:
    - Von Neumann architecture
    - Harvard architecture
- As with any model, this is just an idealization
  - No real computer "looks" just like this
  - Tries to capture the "essence", the main elements

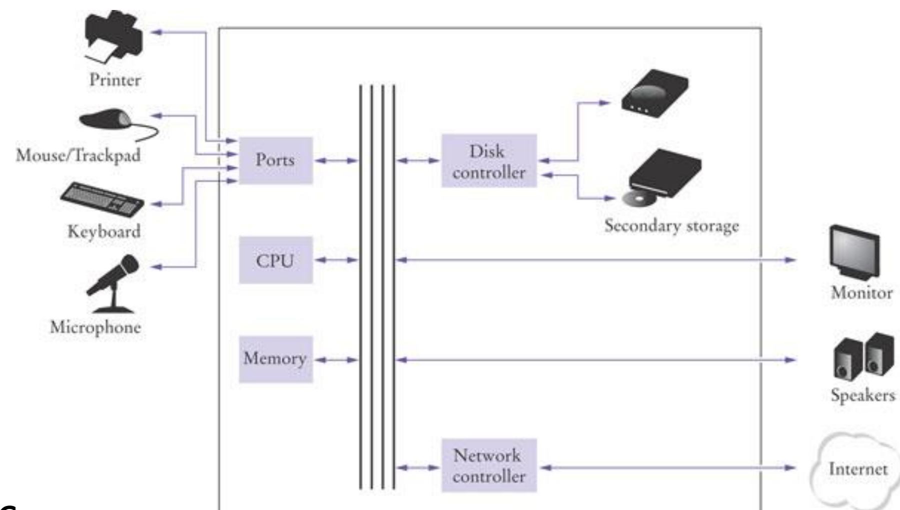


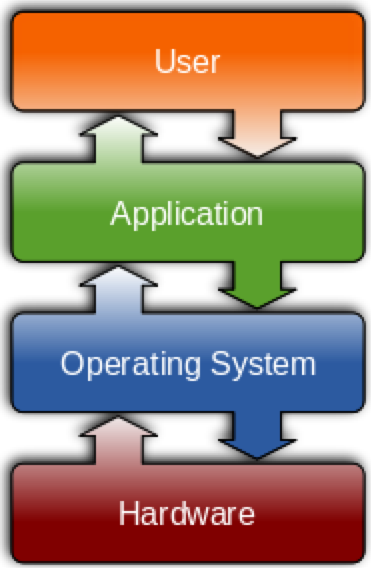
Figure 1-3

Typical Computer Hardware

- CPU
  - Executes the instructions of the program
- Data storage
  - Store instructions and data so program can be loaded into memory and executed
  - Traditionally, hard disk and CD-ROM
  - Nowadays, often web-based or cloud storage
- Main memory
  - Stores the program instructions and data while executing
- Input devices
  - Traditional PCs use keyboard/mouse for data input
  - Nowadays: <u>touch screens</u> for mobile devices
- Output devices
  - Used to display output from a program
  - Output screen, printer, etc.
- Other devices

Layered Computer Model

- Depicts a computer as various "layers"
  - Each layer only interacts with adjacent layers
  - Each layer only knows about the adjacent layers
  - Increased detail going DOWN
  - Increased simplification going UP
- Hardware
  - Microprocessor
  - Ultimately, only understands binary 1's and 0's
- Operating system (OS)
  - A "wrapper" around the hardware (CPU)
  - Provides a uniform interface for all services an application may need
  - Examples: Windows, Mac OS, Linux/UNIX, iOS, Android
- Applications
  - Run upon the stable foundation provided by the OS
  - Generally speaking, shouldn't know or care about the underlying hardware

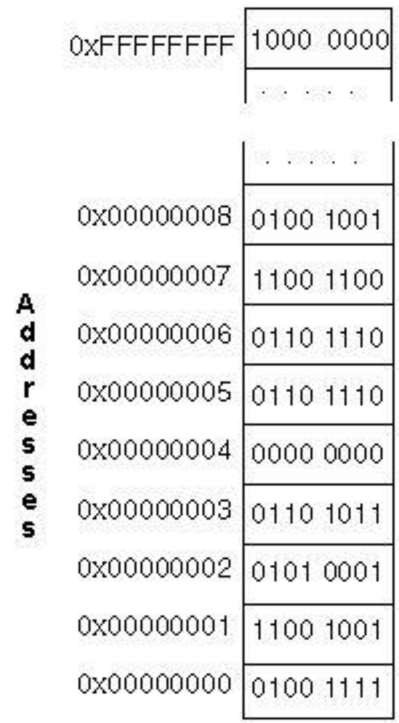


Roles of the Operating System (OS)

- OS **boots** when computer is turned on, and runs continuously
- Controls the peripheral devices (disks, keyboard, mouse, etc.)
- Supports multitasking (multiple applications executing simultaneously)
- Allocates memory to each application
- Prevents one application from damaging another application

Memory Model

- For programming purposes, we abstract memory as a linear array of memory **cells**
  - Each cell has a unique **address**
  - The addresses are commonly symbolically-named (**variables**)
  - Content is binary bits (1s and 0s)
  - <u>All</u> memory content is one of two things:
    - **program instructions**
    - **program data**
  - Without knowing some <u>context</u>, it is impossible to differentiate the two



Main Memory

Memory Storage

- Memory consists of cells that hold one **bit**
  - Bit → **Bi**nary Digi**t**
- A **bit**'s value can only be 0 or 1
- A **byte** is (typically) 8 binary digits (bits)
- Storage capacity is expressed using byte prefixes:
  - **Kilobytes [KB]** (2^{10} = 1,024 bytes, or ~1 thousand)
  - **Megabytes [MB]** (2^{20} = 1,048,576 bytes, or ~1 million)
  - **Gigabytes [GB]** (2^{30} = 1,073,741,824 bytes, or ~1 billion)
  - **Terabytes [TB]** (2^{40} = 1.09951 x 10^{12} bytes, or ~1 trillion)

Black Box Model

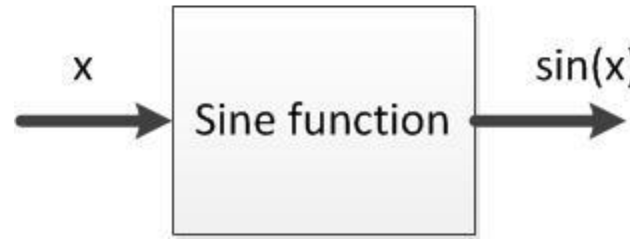
- For programming purposes, we often view individual methods as **black boxes**
  - Provide specified **Inputs**
  - **Actions** are taken upon and/or using those inputs
  - **Outputs** are returned
  - No visibility into the internal details

- From a user or application standpoint:
  - We don't know or care about internal details if we just want to use this
  - We only care about the internal details if we have to implement them
  - Focused on the what, not the how
  - Want the results to be accurate, repeatable, and delivered in a reasonable time

- If the **interface** (the inputs/outputs) is clearly specified:
  - The software **implementation** shell neatly follows
  - We can then focus on the details of the implementation

Black Box Model Example

- We need the sine of an angle
  - I'll give you the angle [degrees]
  - You give me the sine of that angle
- We don't know (or care) how this gets done internally
  - Trig table lookup?
  - Power series expansion?
  - Database query?
  - Google search?
  - Web service?
  - Flying monkeys with an abacus?
- We can easily set up the Java **method interface** from a black box depiction
  - Set up the shell of the code first
  - Supply the details later



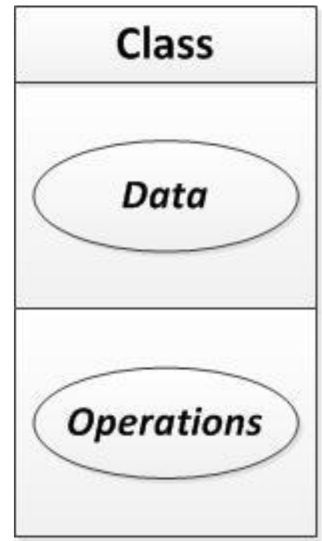

```java
public double calculateSine (double x) {
    double sineOfX;
    // the actual implementation details are TBD...


    return sineOfX;
}
```

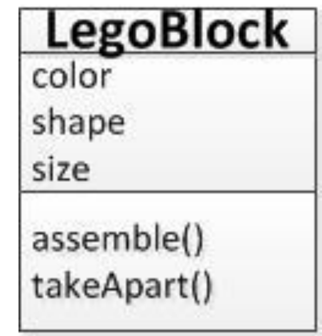
UML

- **UML → Unified Modeling Language**
  - *"A general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system"* (Wikipedia)
  - Synthesized others' work in the mid-1990s
  - Became an approved international ISO standard in 2000.
- There are ~15 UML diagram types, but we will only concern ourselves with one particular diagram
  - The **UML Class diagram**, used to show the structure of software objects

UML Class Diagram

- We have already encountered this model:
  - First lecture, when we talked about software “building blocks”
  - LAB04: Hello Again, where we refactored our code into a “Hello” class
- UML Class Diagrams are used to show the structure of new Java classes
- Three distinct sections
  - **Class name**
  - **Instance variables** → data (what is “is”)
  - **Methods** → operations (what it “does”)
    - **(Formal Java terms** → layman’s terms)





Numbering Systems: Base 10

10^1	10^0
	0
	1
	2
	3
	4
	5
	6
	7
	8
	9
1	0
1	1
1	2
1	3
1	4
1	5
1	6

and so on…

- We are familiar with this numbering system from early school days
- Also called the **decimal system**

- The **base** is 10
- There are 10 unique **digits** [0-9]

- Each column position, right to left , represents an increasing power (0 to N) of the base 10:

  $(\text{base})^{\text{exponent}} \rightarrow (10)^{\text{exponent}}$
- Column **place values** increase <u>right to left</u> by powers of the base (10):

  $10^0 = 1$, $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$, …

- When counting, upon running out of digits in one column, we:
  - "roll over"/increment the next column position
  - "zero out" the lower column position
  - continue counting

Base 10 Expanded Notation

- To write a base 10 number in expanded notation:
  - Working from right to left, determine the maximum place value needed, starting from 10^0
  - Working from left to right, express each decimal digit as the product of that digit times its place value: $\mathbf{a_i * 10^i}$
  - **Expanded notation** is simply the sum of those terms

- Example: convert 2485 to expanded notation
  - Maximum place value needed is 1000, or 10^3

  $2485 = (2 * 10^3) + (4 * 10^2) + (8 * 10^1) + (5 * 10^0)$ ← this is fine

  $2485 = (2 * 1000) + (4 * 100) + (8 * 10) + (5 * 1)$

  $2485 = 2000 + 400 + 80 + 5$

Why Not Base 10?

- Base 10 (decimal) is very natural for us humans
- But for computers, “natural” is only two opposite states:
  - true/false
  - high/low
  - on/off
  - yes/no
  - black/white
- This necessitates a whole new bi-state numbering system, based upon solely 0 & 1

Numbering Systems: Base 2

2^4	2^3	2^2	2^1	2^0
				0
				1
			1	0
			1	1
		1	0	0
		1	0	1
		1	1	0
		1	1	1
	1	0	0	0
	1	0	0	1
	1	0	1	0
	1	0	1	1
	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1
1	0	0	0	0

and so on…

- Now we consider a more PC-friendly numbering system, base 2
- Also called **binary**
- **Base 2 is used <u>exclusively</u> inside of computers**

- The **base** is now 2
- There are only 2 unique **bits** [0, 1]   bit → **bi**nary digi**t**

- Each column position, right to left , represents an increasing power (0 to N) of the base 2:

  $(\text{base})^{\text{exponent}} \rightarrow (2)^{\text{exponent}}$
- Column **place values** increase right to left by powers of the base (2):

  $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, …

- When counting, upon running out of bits in one column, we:
  - "roll over"/increment the next column position
  - "zero out" the lower column position
  - continue counting

- To clearly specify base 2, we often use a 2 subscript:
  - Example: 1101_2

Powers of 2

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

	Decimal
2^8	256
2^9	512
2^{10}	1,024
2^{11}	2,048
2^{12}	4,096
2^{13}	8,192
2^{14}	16,384
2^{15}	32,768

Binary/Decimal Equivalents

<u>Decimal</u>	<u>Binary</u>
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000

"There are only 10 types of people in the world:
Those who understand binary, and those who don't"

Base 2 Expanded Notation

- To write a base 2 number in expanded notation:
  - Working from <u>right</u> to <u>left</u>, determine the maximum place value needed, starting from 2^0
  - Working from <u>left</u> to <u>right</u>, express each decimal digit as the product of that digit times its place value: $\mathbf{a_i * 2^i}$
  - Expanded notation is simply the <u>sum</u> of those terms

- Example: convert 1101 to expanded notation
  - Maximum place value needed is 2^3, or 8

$\mathbf{1101 = (1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0)}$ ← this is fine

$1101 = (1 * 8) + (1 * 4) + (0 * 2) + (1 * 1)$

$1101 = 8 + 4 + 0 + 1 = \underline{13}$

Problems With Base 2

- Do you see a problem looming with base 2??
- Binary numbers grow very large, very quickly
  - Computers have no problems with this
  - But humans are very error-prone
  - Long sequences of 1s and 0s are very difficult for us to absorb without making mistakes
  - Here's an arbitrary 32-bit memory address, can you accurately repeat this?

01101001101011001100100111001001

- These issues motivate a new numbering system
  - Retain the underlying binary numbering
  - But consolidate the bits by groups of 4 (shorthand)
  - Better human readability
  - Provides a "compromise" middle ground between binary and base 10

Numbering Systems: Base 16

16^1	16^0
	0
	1
	2
	3
	4
	5
	6
	7
	8
	9
	A
	B
	C
	D
	E
	F
1	0
1	1
1	2

and so on…

- Finally, we consider a middle-ground numbering system, base 16
- Also called **hexadecimal**, or just **hex**
- **Hex is for human visualization of binary, computers still use binary internally**

- The **base** is now 16
- There are 16 unique **digits** [0-9, A-F]
- Upper or lower case letters are OK, but upper case preferred

- Each column position, right to left , represents an increasing power (0 to N) of the base 16:

  $(\text{base})^{\text{exponent}} \rightarrow (16)^{\text{exponent}}$
- Column **place values** increase right to left by powers of the base (16):

  $16^0 = 1$, $16^1 = 16$, $16^2 = 256$, $16^3 = 4096$, $16^4 = 65536$, …

- When counting, upon running out of bits in one column, we:
  - "roll over"/increment the next column position
  - "zero out" the lower column position
  - continue counting

- To clearly specify base 16, we often use either a 16 subscript or a 0x prefix:
  - Examples: $3F8A_{16}$ or 0x3F8A

Powers of 16

	Decimal	Binary
16^0	1	2^0
16^1	16	2^4
16^2	256	2^8
16^3	4096	2^{12}
16^4	65,536	2^{16}

Hex/Decimal/Binary Equivalents

Dec	**Hex**	**Binary**	**Dec**	**Hex**	**Binary**
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

Base 16 Expanded Notation

- To write a base 16 number in expanded notation:
  - Working from right to left, determine the maximum place value needed, starting from 16^0
  - Working from left to right, express each decimal digit as the product of that digit times its place value: $\mathbf{a_i * 16^i}$
  - Expanded notation is simply the sum of those terms

- Example: convert 0x3F8A to expanded notation
  - Maximum place value needed is 16^3, or 4096

0x3F8A = (3 * 16^3) + (F * 16^2) + (8 * 16^1) + (A * 16^0) ← this is fine

0x3F8A = (3 * 4096) + (15 * 256) + (8 * 16) + (10 * 1)

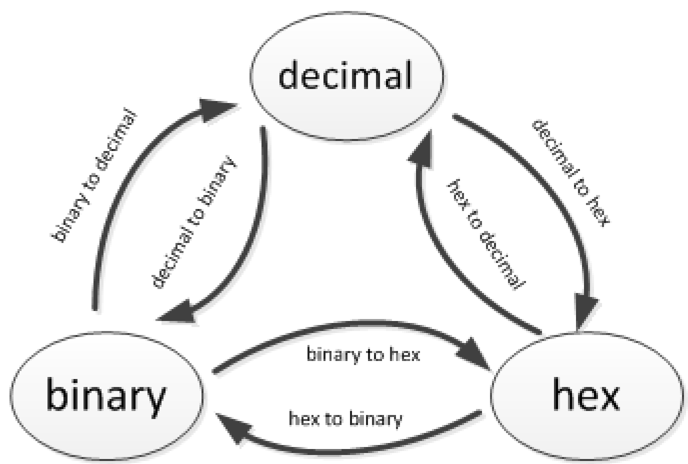
0x3F8A = 12,288 + 3840 + 128 + 10 = 16266

Number Conversions

- Different numbering systems are just different ways of representing the same numerical quantity

  $12 = 12_{10} = 1100_2 = \text{0x0C} = \text{twelve} = \text{one dozen}$

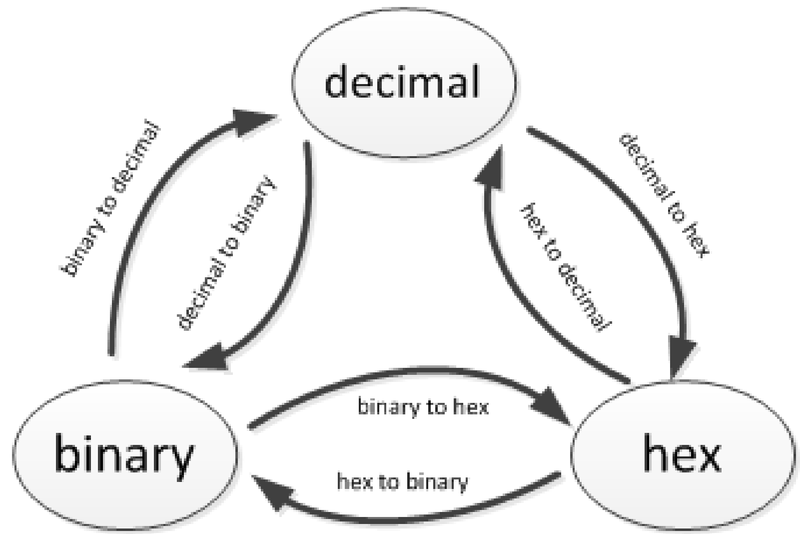
- Depending upon the situation, we need to be able to make all the following conversions:



Converting Between Numbering Systems

- Useful to have these at hand first:
  - Decimal/binary/hex patterns out to 16
  - Powers of 2
  - Powers of 16

- Use simple pattern substitution for:
  - binary to hex
  - hex to binary
- Use expanded notation for:
  - binary to decimal
  - hex to decimal
- Use conversion algorithms for:
  - decimal to binary
  - decimal to hex



Binary/Hex Conversions

- These are very straightforward:
  - For binary numbers, left-pad with 0s if needed, and break the number into 4-bit chunks, from right to left
  - Perform straight binary ←→ hex pattern substitutions using simple table lookups

Binary number:	0001	1010	1111	1001
Hex equivalent:	1	A	F	9

Hex number:	B	3	B	E
Binary equivalent:	1011	0011	1011	1110

Binary/Hex to Decimal Conversions

- To convert from binary to decimal, or hex to decimal, simply use expanded notation, then fully evaluate:

$$\mathbf{100110_2} = 1\text{x}2^5 + 0\text{x}2^4 + 0\text{x}2^3 + 1\text{x}2^2 + 1\text{x}2^1 + 0\text{x}2^0$$

$= 1\text{x}32 + 0\text{x}16 + 0\text{x}8 + 1\text{x}4 + 1\text{x}2 + 0\text{x}1$

$= 32 + 0 + 0 + 4 + 2 + 0$

$= \underline{38}$

$$\mathbf{0x14BD} = 1\text{x}16^3 + 4\text{x}16^2 + \text{B}\text{x}16^1 + \text{D}\text{x}16^0$$

$= 1\text{x}4096 + 4\text{x}256 + 11\text{x}16 + 13\text{x}1$

$= 4096 + 1024 + 176 + 13$

$= \underline{5309}$

Decimal to Binary Conversion Algorithm

1) For the decimal number, find the largest power of 2 that is smaller than or equal to the original decimal number
2) Subtract that power of 2 from the decimal number, to obtain a remainder
3) Find the largest power of 2 that is smaller than or equal to the remainder, and subtract to get a <u>new</u> remainder
4) Repeat step 3 until the remainder has been driven to 0
5) Express the resulting sum as powers-of-2 terms
6) For the resulting sum of power-of-2 terms:
   a) use a 1 for the coefficient of each power-of-2 term that is present
   b) use a 0 for the coefficient of each missing power-of-2 term

This is a mouthful! Best seen by an example (next slide)…

Decimal to Binary Conversion Example

$47 = \underline{32} + 15$ left over (and 16 won't fit into 15)

$= \underline{32} + \underline{8} + 7$ left over

$= \underline{32} + \underline{8} + \underline{4} + 3$ left over

$= \underline{32} + \underline{8} + \underline{4} + \underline{2} + 1$ left over

$= \underline{32} + \underline{8} + \underline{4} + \underline{2} + \underline{1}$ (remainder = 0)

$= 2^5 + 2^3 + 2^2 + 2^1 + 2^0$

$= \; 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \qquad = \mathbf{101111_2}$ (answer)

$\quad\; 2^5 \; 2^4 \; 2^3 \; 2^2 \; 2^1 \; 2^0$

Note that we have one missing (0) coefficient in this example

Decimal to Hex Conversion Algorithm

1) Find the largest power of 16 that is smaller than or equal to the original decimal number
2) Divide by that power of 16; then, how many whole times will it go into the original decimal number?
3) Subtract [(# whole times) x (power of 16)] from the original decimal number, to get a remainder
4) Use the hex digit for (# whole times) for the coefficient of the step 1 (power of 16)
5) Repeat steps 1-4, applied to the new remainder, until you have driven the remainder to 0.

This is a mouthful! Best seen by an example (next slide)…

Decimal to Hex Conversion Example

<u>Convert 500 from decimal to hex:</u>

256=16^2 is the largest power of 16, divides 500 by **1** time
500 – (**1** x 256) = 244, the new remainder
<u>so **1** is the coefficient for 256=16^2</u>

16=16^1 is the largest power of 16, divides 244 by 15 times
244 – (**15** x 16) = 4, the new remainder
<u>so **15=F** is the coefficient for 16=16^1</u>

1=16^0 is the largest power of 16, divides 4 by 4 times
4 – (4 x 1) = 0, the new remainder, so we are <u>done</u>
<u>so **4** is the coefficient for 1=16^0</u>

Final Answer is: <u>**0x01F4**</u>
(Note: By convention, we often left-pad hex numbers with 0s, so that there are an even number (pairs) of hex digits)

1 **R244**
256 | 500
256
244

15 **R4**
16 | 244
240
4

4 **R0**
1 | 4
4
0

What Is This Number?

- How can I tell what this number represents??

100

- Is it…
  - decimal 100? (as in, pennies to the dollar)
  - binary 4?
  - hex 256?
- We need to know some context:
  - If it's a base-10 number: $100_{10} = 100$
  - If it's a base-2 number: $100_{2} = 4$
  - If it's a base-16 number: 0x100 or $100_{16} = 256$

What Is (Or Isn't) This Number?

- **10010010**
  - Could be any one of binary, decimal, OR hex
  - 1's and 0's are <u>common</u> to binary, decimal, AND hex
- **10017010**
  - Could ONLY be decimal or hex
  - Cannot be binary (because of the 7 digit)
- **1001C010**
  - Can ONLY be hex
  - Cannot be binary or decimal (because of the C digit)

For Next Time

- **Lecture Prep**
  - Text readings and lecture notes
- **Assignments**
  - See slide 2 for new/current/past due assignments