Module 3: Representing Values in a Computer

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.

Module 3: Coming up...

- Pre-class quiz #4 is due Wednesday September 30th at 19:00.
  - 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.

- Pre-class quiz #5 is tentatively due Monday October 5th at 19:00.
  - Assigned reading for the quiz:
    - Epp, 4th edition: 3.1, 3.3
    - Epp, 3rd edition: 2.1, 2.3
    - Rosen, 6th edition: 1.3, 1.4
    - Rosen, 7th edition: 1.4, 1.5
Module 3: Representing Values

- Quiz 3 feedback:
  - Well done overall.
  - Only one question had an average below 90%:
    What is the decimal value of the signed 6-bit binary number 101110?
  - Answer:

- By the end of this module, 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 and hexadecimal for human inspection of raw binary data.

Module 3: Representing Values

- CPSC 121: the BIG questions:
  - We will make progress on two of them:
    - How does the computer (e.g., Dr. Racket) decide if the characters of your program represent a name, a number, or something else? How does it figure out if you have mismatched " " or ( )?
    - How can we build a computer that is able to execute a user-defined program?
Module 3: Representing Values

- **Motivating examples:**
  - 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 $500 million space vehicle caused by failure of the guidance system, caused in turn by misuse of a 16 bit signed binary value.
  - We will discuss both of the representations that caused these catastrophes.

Module 3.1: Unsigned and signed binary integers

- **Notice the similarities:**

<table>
<thead>
<tr>
<th>Number</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>8</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>9</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>b_3</th>
<th>b_2</th>
<th>b_1</th>
<th>b_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Definitions:**
  - An **unsigned integer** is one we have decided will only represent integer values that are 0 or larger.
  - A **signed integer** is one we have decided can represent either a positive value or a negative one.
  - A sequence of bits
    - is intrinsically neither signed nor unsigned (nor anything else).
    - it's us who give it its meaning.
Module 3.1: Unsigned and signed binary integers

- Unsigned integers review: 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
  \[ \sum_{i=0}^{n-1} b_i 2^i \]

- We normally use base 10 instead of 2, but we could use 24 [clocks!] or 13 (maybe…) or any other value.

- “Magic” formula to negate a signed integer:
  - Replace every 0 bit by a 1, and every 1 bit by a 0.
  - Add 1 to the result.
  - This is called two's complement.

  Why does it make sense to negate a signed binary integer this way?

Module 3.1: Unsigned and signed binary integers

- For 3-bit integers, what is 111 + 1? Hint: think of a 24 hour clock.
  a) 110
  b) 111
  c) 1000
  d) 000
  e) Error: we can not add these two values.

- Using 3 bits to represent integers
  - let us write the binary representations for zero to eleven.

\[ \begin{array}{c|c}
0 & 000 \\
\end{array} \]
Module 3.1: Unsigned and signed binary integers

- Using 3 bits to represent integers
  - let us write the binary representations for zero to eleven.

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
000 & 001 & 010 & 011 \\
\end{array}
\]
Module 3.1: Unsigned and signed binary integers

- Using 3 bits to represent integers
  - let us write the binary representations for zero to eleven.
Module 3.1: Unsigned and signed binary integers

- Using 3 bits to represent integers
  - let us write the binary representations for zero to eleven.

Using 3 bits to represent integers

- let us write the binary representations for zero to eleven.

now let’s add the binary representation for zero to minus eight.
Module 3.1: Unsigned and signed binary integers

- Using 3 bits to represent integers
  - let us write the binary representations for zero to eleven.
  - now let’s add the binary representation for zero to minus eight.
Module 3.1: Unsigned and signed binary integers

- Using 3 bits to represent integers
  - let us write the binary representations for zero to eleven.
  - now let’s add the binary representation for zero to minus eight.
Module 3.1: Unsigned and signed binary integers

- What do you notice?
  
  - Taking two’s complement is the same as computing $2^n - x$ because

  $$2^n - x = (2^n - 1 - x) + 1$$

  Add 1

  Flip bits from 0 to 1 and from 1 to 0

Module 3.1: Unsigned and signed binary integers

- What does a sequence of bit actually mean?
  
  - If we know we won't need negative values: **unsigned**

    
    \[
    \begin{array}{ccccccccccc}
    -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
    000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 & 000
    \end{array}
    \]

  - If we need negative values: **signed**

    
    \[
    \begin{array}{ccccccccccc}
    -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
    000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 & 000
    \end{array}
    \]

- One way to convert a positive decimal integer $x$ to binary?
  
  - Divide $x$ by 2 and write down the remainder
    
    - The remainder is 0 if $x$ is even, and 1 if $x$ is odd.
    
    - Repeat this until the quotient is 0.
    
    - Write down the remainders from right (first one) to left (last one).

  - Example: convert 729 to binary.

- What do we do if $x$ is negative?

- 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?
Module 3.1: Unsigned and signed binary integers

- More summary questions:
  - Why are there more negative $n$-bit signed integers than positive ones?
  - How do we tell quickly if a signed binary integer is negative, positive, or zero?
  - 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?

Module 3: Representing Values

- Summary
  - Unsigned and signed binary integers.
  - Modular arithmetic.
  - Characters.
  - Real numbers.
  - Hexadecimal.

Module 3.2: Modular arithmetic

- First open-ended question from quiz #3:
  - Imagine the time is currently 15:00 (3:00PM, that is). How can you quickly answer the following two questions without using a calculator:
    - What time was it 8 * 21 hours ago?
    - What time will it be 13 * 23 hours from now?
Module 3.2: Modular arithmetic

- Modular arithmetic (continued):
  - We use the smallest non-negative element of the class as its representative. With $m = 5$:
    - $[0] = \{ \ldots, -15, -10, -5, 0, 5, 10, 15, \ldots \}$
    - $[1] = \{ \ldots, -14, -9, -4, 1, 6, 11, 16, \ldots \}$
    - etc.
  - We write $x \mod m$ to denote the representative for the class that $x$ belongs to.
    - $x \mod m$ is the remainder we get after dividing $x$ by $m$.

Example:
27 mod 4 is 3 ($27 = 6 \times 4 + 3$).
What is 57 mod 8?
   a) 1
   b) 3
   c) 5
   d) 7

If $x$ and $y$ belong to the same class modulo $m$ (have the same remainder) then we write $x \equiv y \mod m$.

Suppose that $x \equiv 34 \mod 6$. Which are possible values for $x$?
   a) 4, 17 and 28.
   b) 12, 28 and 38.
   c) 36, 72 and 216.
   d) 10, 16 and 52.

Fundamental Theorem of Modular Arithmetic:
- Suppose you want to compute $cx + d \mod m$
  - if $a \equiv c \mod m$ and $b \equiv d \mod m$ then $ax + b \equiv cx + d \mod m$
- This theorem means that it doesn’t matter if you
  (a) do a sequence of operations, and then take the remainder mod $m$ at the end.
  (b) or take the remainder mod $m$ every time you perform an operation in the sequence.
- Sequences of operations on integers do (b).
Module 3: Representing Values

- Summary
  - Unsigned and signed binary integers.
  - Modular arithmetic.
  - **Characters.**
  - Real numbers.
  - Hexadecimal.

Module 3.3: Characters

- How do computers represent characters?
  - It uses 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.

Module 3.3: Characters

- How do computers represent characters (continued)?
  - Examples:
    - 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 or 32 bits, extended ASCII for languages other than English

Module 3.3: Characters

- What does the 8-bit binary value 11111000 represent?
  - a) -8
  - b) The character ø
  - c) 248
  - d) More than one of the above
  - e) None of the above.
Module 3: Representing Values

- **Summary**
  - Unsigned and signed binary integers.
  - Modular arithmetic.
  - Characters.
  - **Real numbers**.
  - Hexadecimal.

Module 3.4: Real numbers

- **Can someone be 1/3rd Belgian?**
- **Here is a fun answer from this term:**
  - No, because you have to have two parents that have a fraction of an ethnicity and the denominator has to be a number that is a power of 2. You would need three parents for that to be possible. However, there is a medical procedure where you replace the nucleus of an egg with bad mitochondria into a nucleus-less egg with working mitochondria, which can then be fertilized with sperm. This procedure involves three people and if one of them is Scottish, then you can technically be one-third Scottish.

Module 3.4: Real numbers

- **Another answer from this term:**
  - It is possible for someone to be 50% Scottish by being the offspring of 1 Scottish person and 1 non-Scottish person. It is also possible for someone to be 25% Scottish by being the offspring of a parent who is 50% Scottish and another non-Scottish parent.

  Given that a person can be 50% or 25% Scottish, and that 33.33% is between the two, then it is possible that an offspring many generations down the line could be 33.33% Scottish by following this logic when selecting a partner:

  - If self is over 33.33% Scottish : procreate with a person who is 25% Scottish
  - If self is under 33.33% : procreate with a person who is 50% Scottish

  Repeat the process until an offspring is born who is 33.33% Scottish (probably over 10 generations down).

Module 3.4: Real numbers

- **An interesting older answer:**
  - Let's focus on Mom, suppose we are 1/3 Scottish, then your mom should be 2/3 Scottish and therefore your father is not Scottish. Given mom is 2/3 Scottish, then your grandparent should either be

    1) both 2/3 Scottish. But this will lead to infinite generations of 2/3 Scottish, which is impossible

    2) Grandma is 1/6 and grandfather is a pure Scottish. Then grandma's parent should now be 1/3 and not Scottish, then grandma's grand parent should now be 2/3 and not Scottish. Notice, this runs into a loop which is like you and your mom.

    Therefore, this is also an infinite loop and drives to the conclusion that we can't be one-third Scottish.
Module 3.4: Real numbers

- Here is an even older answer:
  - While debated, Scotland is traditionally said to be founded in 843AD, approximately 45 generations ago. Your mix of Scottish, will therefore be $n/2^{45}$; using $2^{45}/3$ (rounded to the nearest integer) as the numerator gives us $11728124029611/2^{45}$ which gives us approximately $0.333333333333342807236476801$ which is no more than $1/10^{13}$th away from $1/3$.

- Another old, interesting answer
  - In a mathematical sense, you can create $1/3$ using infinite sums of inverse powers of $2$
    - $1/2$ isn't very close
    - $1/4$ isn't either
    - $3/8$ is getting there...
    - $5/16$ is yet closer, so is $11/32$, $21/64$, $43/128$ etc
    - $85/256$ is $0.33203125$, which is much closer, but which also implies eight generations of very careful romance amongst your elders.
    - $5461/16384$ is $0.33331298828125$, which is still getting there, but this needs fourteen generations and a heck of a lot of Scots and non-Scots.

Module 3.4: Real numbers

- Can someone be $1/3^{rd}$ Belgian?
  - Suppose we start with people who are either 0% or 100% Belgian.
  - After 1 generation, how Belgian can a child be?
  - After 2 generations, how Belgian can a grand-child be?
  - What about 3 generations?
  - What about $n$ generations?

Module 3.4: Real numbers

- Numbers with fractional components in binary:
  - Example: $5/32 = 0.00101$
  - Which of the following values have a finite binary expansion?
    - a) $1/3$
    - b) $1/4$
    - c) $1/5$
    - d) More than one of the above.
    - e) None of the above.
Module 3.4: Real numbers

- Numbers with fractional components (cont):
  - In decimal:
    - \( \frac{1}{3} = 0.3333333333333333333333333333333333... \)
    - \( \frac{1}{4} = 0.25 \)
    - \( \frac{1}{5} = 0.2 \)
  - In binary:
    - \( \frac{1}{3} = \)
    - \( \frac{1}{4} = \)
    - \( \frac{1}{5} = \)

- Which fractions have a finite binary expansion?

Module 3.4: Real numbers

- How does Java represent values of the form \( \text{xxx.yyyy} \)?
  - It uses scientific notation
    - \( 1724 = 0.1724 \times 10^4 \)
  - But in binary, instead of decimal.
    - \( 1724 = 1.1010111100 \times 2^{1010} \)
  - Only the mantissa and exponent need to be stored.
  - The mantissa has a fixed number of bits (24 for float, 53 for double).

Module 3.4: Real numbers

- Scheme/Racket uses this for inexact numbers.

- 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:
    - \( (* \ (\text{sqrt} \ 2) \ (\text{sqrt} \ 2)) \)

- Consider the following:
  - (define (addfractions x)
    (if (= x 1.0)
        0
        (+ 1 (addfractions (+ x #i0.1)))))

- What value will \( \text{(addfractions 0)} \) return?
  - a) 10
  - b) 11
  - c) Less than 10
  - d) More than 11
  - e) No value will be printed
Module 3: Representing Values

- Summary
  - Unsigned and signed binary integers.
  - Modular arithmetic.
  - Characters.
  - Real numbers.
  - Hexadecimal.

Module 3.5: Hexadecimal

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

What does a machine language instruction look like?

- It is a sequence of bits!
- Y86 example: adding two values.
  - In human-readable form: `addl %ebx, %ecx`.
  - In binary: `0110000000110001`.

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: `24625`.
  - Problem: We can not tell what operation this is.

- Solution: use hexadecimal `6031`.
Module 3.5: Hexadecimal

- Another example:
  - Suppose we make the text in a web page use color 15728778.
  - What color is this?
    - Red leaning towards purple.
  - Written in hexadecimal: F00084