Binary Numbers

COS 217
Goals of Today’s Lecture

• Binary numbers
  ◦ Why binary?
  ◦ Converting base 10 to base 2
  ◦ Octal and hexadecimal

• Integers
  ◦ Unsigned integers
  ◦ Integer addition
  ◦ Signed integers

• C bit operators
  ◦ And, or, not, and xor
  ◦ Shift-left and shift-right
  ◦ Function for counting the number of 1 bits
  ◦ Function for XOR encryption of a message
Why Bits (Binary Digits)?

- Computers are built using digital circuits
  - Inputs and outputs can have only two values
  - True (high voltage) or false (low voltage)
  - Represented as 1 and 0

- Can represent many kinds of information
  - Boolean (true or false)
  - Numbers (23, 79, …)
  - Characters (‘a’, ‘z’, …)
  - Pixels
  - Sound

- Can manipulate in many ways
  - Read and write
  - Logical operations
  - Arithmetic
  - …
Base 10 and Base 2

• Base 10
  ◦ Each digit represents a power of 10
  ◦ \(4173 = 4 \times 10^3 + 1 \times 10^2 + 7 \times 10^1 + 3 \times 10^0\)

• Base 2
  ◦ Each bit represents a power of 2
  ◦ \(10110 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 22\)

  Divide repeatedly by 2 and keep remainders

  \[
  \begin{align*}
  12/2 &= 6 & R = 0 \\
  6/2 &= 3 & R = 0 \\
  3/2 &= 1 & R = 1 \\
  1/2 &= 0 & R = 1 \\
  \end{align*}
  \]

  Result = \(1100\)
Writing Bits is Tedious for People

- Octal (base 8)
  - Digits 0, 1, …, 7
  - In C: 00, 01, …, 07

- Hexadecimal (base 16)
  - Digits 0, 1, …, 9, A, B, C, D, E, F
  - In C: 0x0, 0x1, …, 0xf

<table>
<thead>
<tr>
<th>Binary</th>
<th>Octal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0x0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>0x1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>0x2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>0x3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>0x4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>0x5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>0x6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>0x7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>0x8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>0x9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>0xA</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>0xB</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>0xC</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>0xD</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>0xE</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>0xF</td>
</tr>
</tbody>
</table>

Thus the 16-bit binary number

1011 0010 1010 1001

converted to hex is

B2A9
Representing Colors: RGB

• Three primary colors
  ◦ Red
  ◦ Green
  ◦ Blue

• Strength
  ◦ 8-bit number for each color (e.g., two hex digits)
  ◦ So, 24 bits to specify a color

• In HTML, on the course Web page
  ◦ Red: <font color="#FF0000"><i>Symbol Table Assignment Due</i></font>
  ◦ Blue: <font color="#0000FF"><i>Fall Recess</i></font>

• Same thing in digital cameras
  ◦ Each pixel is a mixture of red, green, and blue
Storing Integers on the Computer

• Fixed number of bits in memory
  ◦ Short: usually 16 bits
  ◦ Int: 16 or 32 bits
  ◦ Long: 32 bits

• Unsigned integer
  ◦ No sign bit
  ◦ Always positive or 0
  ◦ All arithmetic is modulo $2^n$

• Example of unsigned int
  ◦ 00000001 $\rightarrow$ 1
  ◦ 00001111 $\rightarrow$ 15
  ◦ 00010000 $\rightarrow$ 16
  ◦ 00100001 $\rightarrow$ 33
  ◦ 11111111 $\rightarrow$ 255
Adding Two Integers: Base 10

- From right to left, we add each pair of digits
- We write the sum, and add the carry to the next column

\[
\begin{array}{c}
\text{Sum} & 4 & 6 & 2 \\
\text{Carry} & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Sum} & 1 & 0 & 0 & 0 \\
\text{Carry} & 0 & 1 & 1 \\
\end{array}
\]
# Binary Sums and Carries

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Sum</th>
<th>a</th>
<th>b</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**XOR**

| 0100 | 0101 | 69 |
| 0110 | 0111 | 103 |
+------|------|----|
| 0110 0111 |   | 172 |
| 1010 1100 |   |     |
Modulo Arithmetic

- Consider only numbers in a range
  - E.g., five-digit car odometer: 0, 1, …, 99999
  - E.g., eight-bit numbers 0, 1, …, 255

- Roll-over when you run out of space
  - E.g., car odometer goes from 99999 to 0, 1, …
  - E.g., eight-bit number goes from 255 to 0, 1, …

- Adding $2^n$ doesn’t change the answer
  - For eight-bit number, n=8 and $2^n=256$
  - E.g., $(37 + 256) \mod 256$ is simply 27

- This can help us do subtraction…
  - Suppose you want to compute $a - b$
  - Note that this equals $a + (256 - 1 - b) + 1$
One’s and Two’s Complement

• One’s complement: flip every bit
  ◦ E.g., b is 01000101 (i.e., 69 in base 10)
  ◦ One’s complement is 10111010
  ◦ That’s simply 255-69

• Subtracting from 11111111 is easy (no carry needed!)

\[
\begin{array}{c}
11111111 \\
- 01000101 \\
\hline
10111010 \\
\end{array}
\]

b

one’s complement

• Two’s complement
  ◦ Add 1 to the one’s complement
  ◦ E.g., \((255 - 69) + 1 \Rightarrow 1011 1011\)
Putting it All Together

• Computing “a – b” for unsigned integers
  ◦ Same as “a + 256 – b”
  ◦ Same as “a + (255 – b) + 1”
  ◦ Same as “a + onecomplement(b) + 1”
  ◦ Same as “a + twocomplement(b)”

• Example: 172 – 69
  ◦ The original number 69: 0100 0101
  ◦ One’s complement of 69: 1011 1010
  ◦ Two’s complement of 69: 1011 1011
  ◦ Add to the number 172: 1010 1100
  ◦ The sum comes to: 0110 0111
  ◦ Equals: 103 in base 10

\[
\begin{array}{c}
1010 1100 \\
+ 1011 1011 \\
\hline
1 0110 0111
\end{array}
\]
Signed Integers

• Sign-magnitude representation
  ◦ Use one bit to store the sign
    – Zero for positive number
    – One for negative number
  ◦ Examples
    – E.g., 0010 1100 ➔ 44
    – E.g., 1010 1100 ➔ -44
  ◦ Hard to do arithmetic this way, so it is rarely used

• Complement representation
  ◦ One’s complement
    – Flip every bit
    – E.g., 1101 0011 ➔ -44
  ◦ Two’s complement
    – Flip every bit, then add 1
    – E.g., 1101 0100 ➔ -44
Overflow: Running Out of Room

• **Adding two large integers together**
  - Sum might be too large to store in the number of bits allowed
  - What happens?

• **Unsigned numbers**
  - All arithmetic is “modulo” arithmetic
  - Sum would just wrap around

• **Signed integers**
  - Can get nonsense values
  - Example with 16-bit integers
    - Sum: 10000+20000+30000
    - Result: -5536
  - In this case, fixable by using “long”…
Bitwise Operators: AND and OR

- **Bitwise AND (&)**
  
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

  - Mod on the cheap!
    - E.g., \( h = 53 \& 15; \)

- **Bitwise OR (|)**
  
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

  ```plaintext
  53 0 0 1 1 0 1 0 1
  \& 15 0 0 0 0 1 1 1 1
  ---------------------
  5 0 0 0 0 0 1 0 1
  ```
Bitwise Operators: Not and XOR

• One’s complement (~)
  ◦ Turns 0 to 1, and 1 to 0
  ◦ E.g., set last three bits to 0
    – \( x = x \& \sim 7; \)

• XOR (^)
  ◦ 0 if both bits are the same
  ◦ 1 if the two bits are different

\[
\begin{array}{ccc}
^ & 0 & 1 \\
0 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]
Bitwise Operators: Shift Left/Right

• Shift left (<<): Multiply by powers of 2
  ○ Shift some # of bits to the left, filling the blanks with 0
    
    $53 \ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0$
    
    $53<<2 \ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0$

• Shift right (>>): Divide by powers of 2
  ○ Shift some # of bits to the right
    – For unsigned integer, fill in blanks with 0
    – What about signed integers? Varies across machines…
      • Can vary from one machine to another!
    
    $53 \ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0$
    
    $53>>2 \ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1$
Count Number of 1s in an Integer

• Function bitcount(unsigned x)
  ◦ Input: unsigned integer
  ◦ Output: number of bits set to 1 in the binary representation of x

• Main idea
  ◦ Isolate the last bit and see if it is equal to 1
  ◦ Shift to the right by one bit, and repeat

```c
int bitcount(unsigned x) {
    int b;
    for (b=0; x!=0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```
XOR Encryption

• Program to encrypt text with a key
  ◦ Input: original text in stdin
  ◦ Output: encrypted text in stdout

• Use the same program to decrypt text with a key
  ◦ Input: encrypted text in stdin
  ◦ Output: original text in stdout

• Basic idea
  ◦ Start with a key, some 8-bit number (e.g., 0110 0111)
  ◦ Do an operation that can be inverted
    – E.g., XOR each character with the 8-bit number

\[
\begin{align*}
0100 & \quad 0101 & & \quad 0010 & \quad 0010 \\
\wedge & \quad 0110 & \quad 0111 & & \wedge & \quad 0110 & \quad 0111 \\
\hline
0010 & \quad 0010 & & \quad 0100 & \quad 0101
\end{align*}
\]
XOR Encryption, Continued

• But, we have a problem
  ◦ Some characters are control characters
  ◦ These characters don’t print

• So, let’s play it safe
  ◦ If the encrypted character would be a control character
  ◦ … just print the original, unencrypted character
  ◦ Note: the same thing will happen when decrypting, so we’re okay

• C function `iscntrl()`
  ◦ Returns true if the character is a control character
#define KEY ‘&’

int main() {
    int orig_char, new_char;

    while ((orig_char = getchar()) != EOF) {
        new_char = orig_char ^ KEY;
        if (iscntrl(new_char))
            putchar(orig_char);
        else
            putchar(new_char);
    }
    return 0;
}
Conclusions

- Computer represents everything in binary
  - Integers, floating-point numbers, characters, addresses, …
  - Pixels, sounds, colors, etc.

- Binary arithmetic through logic operations
  - Sum (XOR) and Carry (AND)
  - Two’s complement for subtraction

- Binary operations in C
  - AND, OR, NOT, and XOR
  - Shift left and shift right
  - Useful for efficient and concise code, though sometimes cryptic
Next Week

- **Canceling second precept**
  - Monday/Tuesday precept as usual
  - Canceling the Wednesday/Thursday precept due to midterms

- **Thursday lecture time**
  - Midterm exam
  - Open book and open notes
  - Practice exams online