Bitwise Operators

Number Representation Recap

Humans think about numbers in decimal

Computers think about numbers in binary

Base conversion to go between

- Hex is more human-readable than binary

All information on a computer is in binary

- Nice because big difference between “high” and “low”

Binary encoding can represent *anything*!

- Program needs to know how to interpret bits

Operators Recap

- NOT: ~
  - This will flip all bits in the operand
- AND: &
  - This will perform a bitwise AND on every pair of bits
- OR: |
  - This will perform a bitwise OR on every pair of bits
- XOR: ^
  - This will perform a bitwise XOR on every pair of bits
- SHIFT: <<, >>
  - This will shift the bits right or left
    - logical vs. arithmetic

Operators Recap

- NOT: !
  - Evaluates the entire operand, rather than each bit
  - Produces a 1 if == 0, produces 0 if nonzero
- AND: &&
  - Produces 1 if both operands are nonzero
- OR: ||
  - Produces 1 if either operand is nonzero

Lab 1

- Worksheet in class
- Tips:
  - Work on 8-bit versions first, then scale your solution to work with 32-bit inputs
  - Save intermediate results in variables for clarity
  - Shifting by more than 31 bits is **UNDEFINED**. This will NOT yield 0

Examples

Create 0xFFFFFFFF using only one operator

- Limited to constants from 0x00 to 0xFF
- Naïve approach:

0xFF + (0xFF << 8) + (0xFF << 16) ...

- Better approach:

~0x00 = 0xFFFFFFFF

Examples

Replace the leftmost byte of a 32-bit integer with 0xAB

- Let our integer be x
- First, we want to create a mask for the lower 24 bits
  - `~(0xFF << 24)` will do that using just two operators
- (x & mask) will zero out the leftmost 8 bits
- Now, we want to OR in 0xAB to those zeroed-out bits
- Final result:
  - `(x & mask) | (0xAB << 24)`
- Total operators: 5