

- Binary is a base-2 numbering system
- Easy to tell difference between on and off
- All data is represented in binary
  - Byte 8 bits
  - Short/char 16 bits
  - Int/float 32 bits
  - Long 64 bits

#### **Understanding Binary**

- Binary to integer
- Binary to hexadecimal
- Integer to binary
- Hexadecimal to binary
- Floating point numbers
  - IEEE 754 floating point specification
- Bit operations
  - Masking, shifting, ANDing, ORing, XORing

#### Counting in Binary

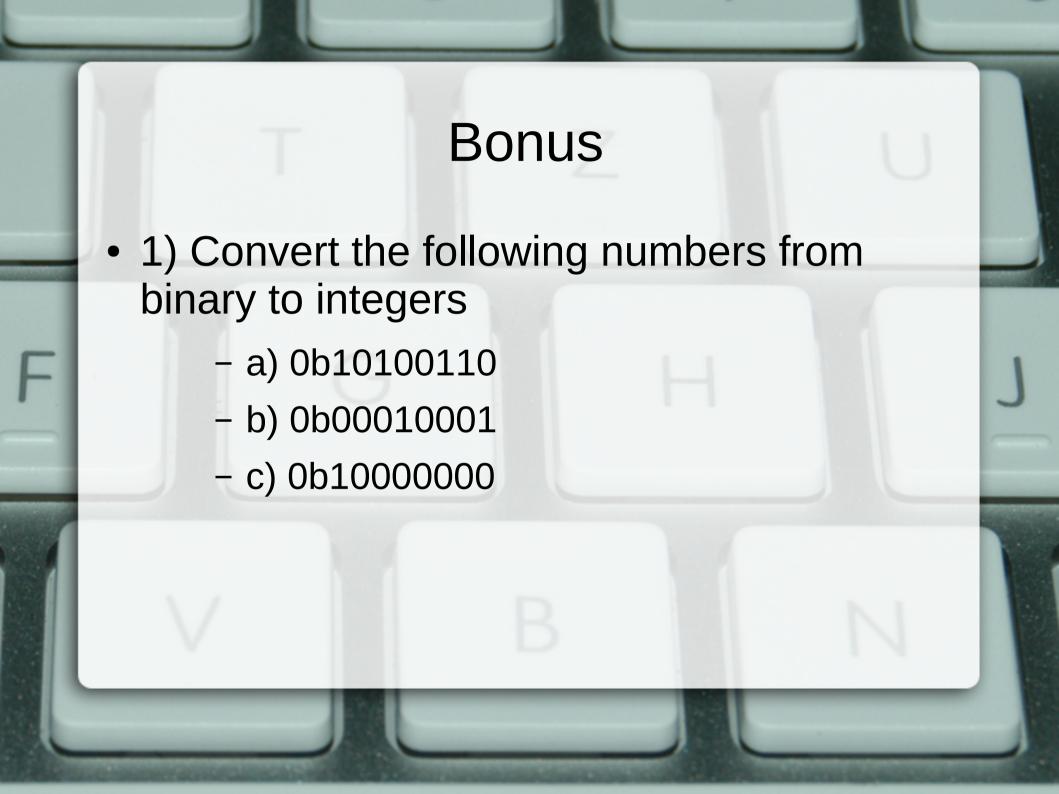
- 0 0000 7 0111 14 1110
- 1 0001 8 1000 15 1111
- 2 0010 9 1001
- 3 0011 10 1010
- 4 0100 11 1011
- 5 0101 12 1100
- 6 0110 13 1101

# Binary to Integer

- To determine the value of a binary number (assuming non-signed)
  - Take a binary number 0b00100110
  - Go from right to left adding in which positions are set

-----

$$0 + 0 + 32 + 0 + 0 + 4 + 2 = 38$$



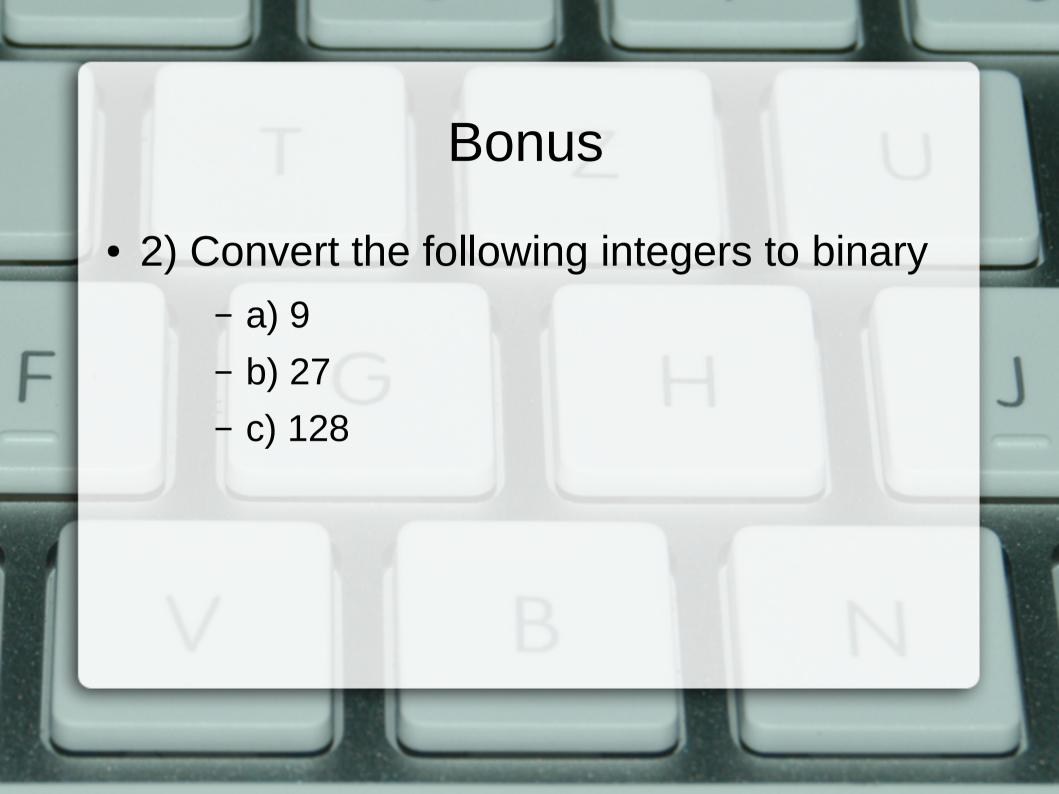
## Integer to Binary

- To convert an integer to binary
  - Repeatedly mod and then divide the integer by 2, storing the remainder from right to left to create a binary number
  - Example 8

• 
$$8\%2 = 0$$
  $8/2 = 4$ 

• 
$$4\%2 = 0$$
  $4/2 = 2$  00

• 
$$2\%2 = 0$$
  $2/2 = 1$  000



#### What is hexadecimal?

- Base-16 numerical system
- Hexadecimal is a shortcut for binary
- Each of the 16 hex digits represent a binary nibble

$$0 - 0000 4 - 0100 8 - 1000 C - 1100$$

$$1 - 0001$$
  $5 - 0101$   $9 - 1001$  D - 1101

$$2 - 0010 6 - 0110 A - 1010 E - 1110$$

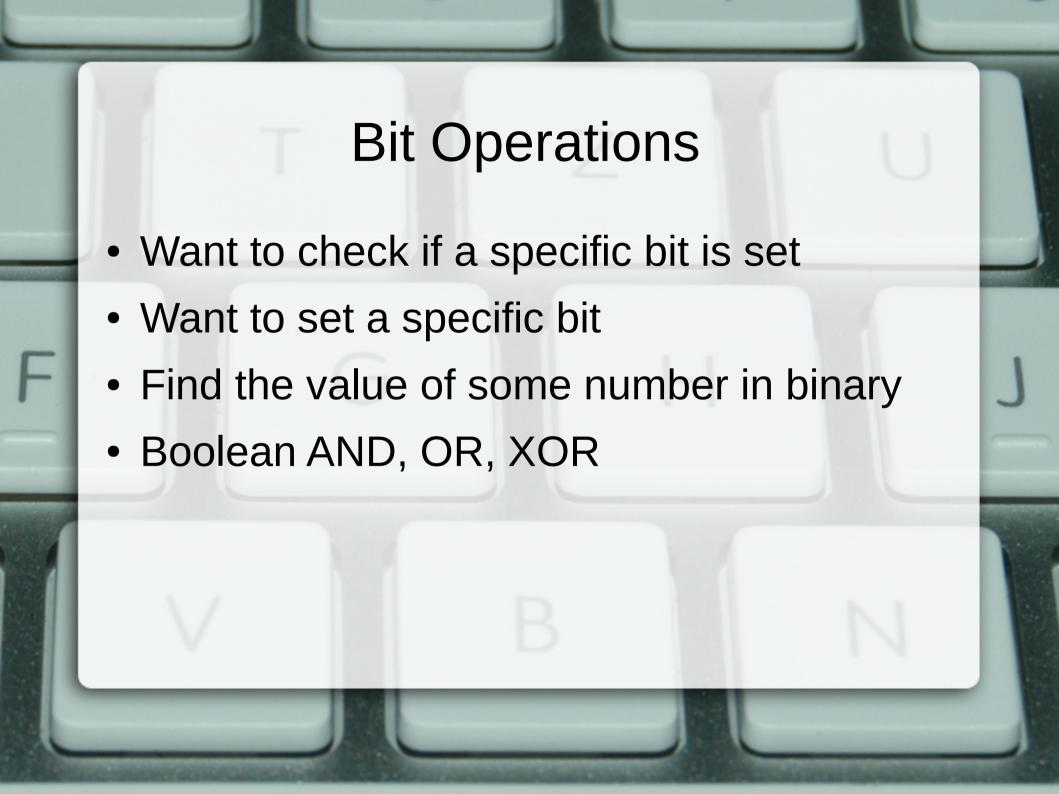
$$3 - 0011$$
  $7 - 0111$   $B - 1011$   $F - 1111$ 

#### What is hexadecimal?

- 0xAB => 0b1010 1011
- 0x10 => 0b0001 0000
- 0x3F => 0b0011 11111
- 0b111111000 => 0xF8
- $0b0010\ 0010 => 0x22$
- 0b0000 0000 => 0x00

#### Bonus

- 3) Convert the following hexadecimal to binary
  - a) 0xF0
  - b) 0x45
- 4) Convert the following binary to hexadecimal
  - a) 0b0001 0011
  - b) 0b1100 1000
- 5) Convert 0x0F to decimal





True if and only if (IFF) both items are true

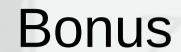
• 0b 0110 1010

AND 0b 0010 0110

-----

0b 0010 0010

	0	1
0	0	0
1	0	1



• 6) What is the result of the following operation?

0b 1110 0001

AND 0b 1011 1001

-----



True if one or both items are set

• 0b 0110 1010

OR 0b 0010 0110

-----

0b 0110 1110

	0	1
0	0	1
1	1	1

#### Bonus

• 7) What is the result of the following operation?

- Ob 0110 1011

OR 0b 1001 1001

-----



True IFF one item is set

• 0b 0110 1010

XOR 0b 0010 0110

-----

0b 0100 1100

	0	1
0	0	1
1	1	0

# Bonus

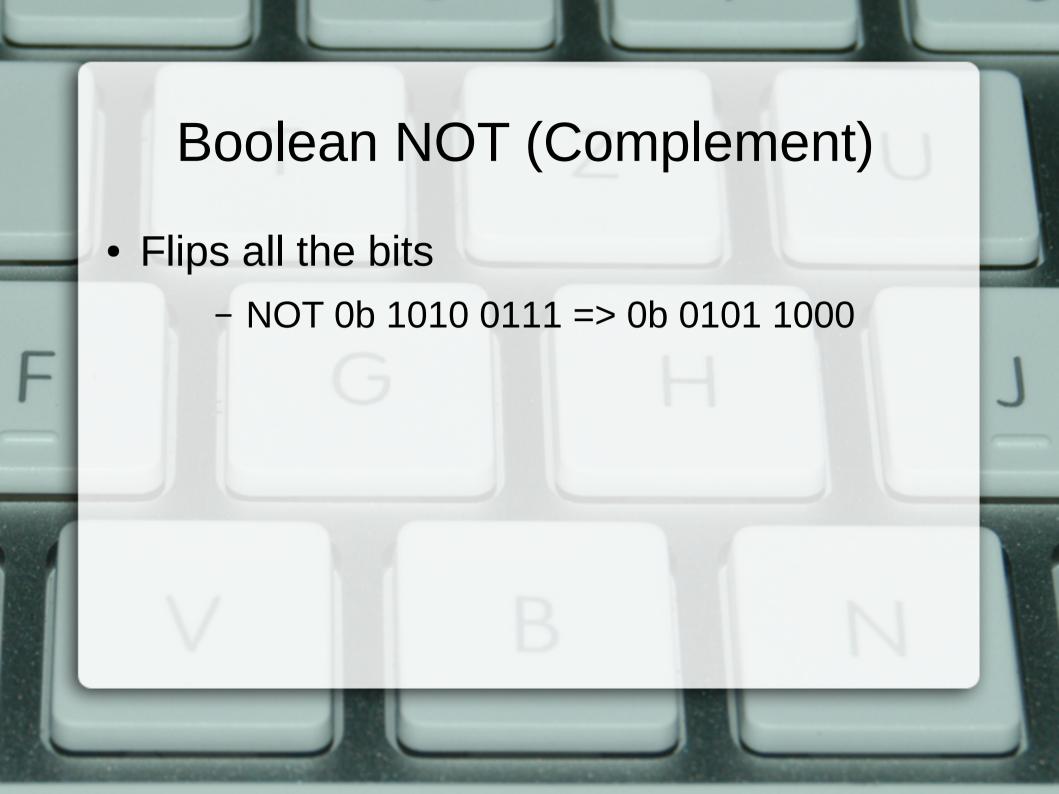
• 8) What is the result of the following operation?

- 0b 1010 1010

XOR 0b 1011 0101

-----

?



#### Boolean Bit Operations in Java

- All of the Boolean bitwise operations can be used in Java
  - AND &
  - OR |
  - XOR ^
  - NOT ~
- We can also use several methods in the Integer API to help us debug
  - Integer.toBinaryString() and toHexString()

## Boolean Bit Operations in Java

- byte a = 0x11 (0b 0001 0001)
- byte b = 0xFF (0b 1111 1111)
- a & b => 0x11 (0b 0001 0001)
- $a \mid b => 0xFF (0b 1111 1111)$
- a ^ b => 0xEE (0b 1110 1110)
- $\sim$ a => 0xEE (0b 1110 1110)
- $\sim$ b => 0x00 (0b 0000 0000)

# Left-Shift and Right-Shift

- Binary values can be shifted N positions to the left or the right
- 0's are shifted into the newly vacated spot
  - With << or >>>
  - 1's can be shifted on using signed shift >>
- Let A = 0xAB (0b 0010 1011)
- Left-Shift(A, 1) => 0x56 (0b 0101 0110)
- Right-Shift(A, 3) => 0x05 (0b 0000 0101)

#### Shifts and Bonus

- Shifts in Java can be performed with the >> and << operators. << indications a left shift and >> indicates a right shift. >>> represents a special unsigned shift.
- 0x01 << 1 = 0x02
- 0x02 << 1 = 0x04
- 0x04 << 1 = 0x08
- 0x01 << 3 = 0x08
- Bonus 9) Find 0x0F << 2</li>

#### Bitmasks and other operations

- Some common operations when working with binary numbers include
  - Setting a bit(s)
  - Checking the status of a bit
  - Toggling a bit
- We can create bit masks and then use Boolean bit operations to operate on binary data

#### Setting a bit

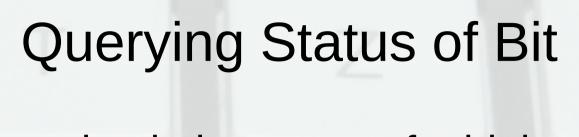
- Using masks to take a food order
- [drink, fries, burgher, onion rings, salad, nuggets, apple, wrap]
- drink\_mask = 0x80 fries\_mask = 0x40
- burgher\_mask = 0x20 onions\_mask = 0x10 salad\_mask = 0x08 nuggets\_mask = 0x04 apple\_mask = 0x02 wrap\_mask = 0x01

#### Taking the order

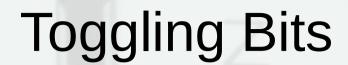
- When we're taking an order, we can set individual bits using OR and their masks
- byte order = 0; (0b 0000 0000)
- order = order | fries\_mask
- order = order | drink\_mask
- order => 0xC0 (0b 1100 0000)
- Can also be written
  - order = order | fries\_mask | drink\_mask

# Canceling items in order

- It may also be necessary to set individual bits to 0. We can do this using AND with masks
- The bit you want to turn off must be 0, and everything else in the mask should be 1
- Cancel drink order = order & (~drink\_mask)
  - Ob 1100 0000 AND Ob 0111 1111 0100 0000



- We can check the status of a bit by using AND with our bitmasks
- Check if customer ordered a salad
- return (order & salad\_mask) > 0



- Bits can be toggled using a mask where each 1 bit flips the bit
- 0b 0101 0101

XOR 0b 0000 1111 (mask)

0b 0101 1010

