

```
Base 10 and Base 2

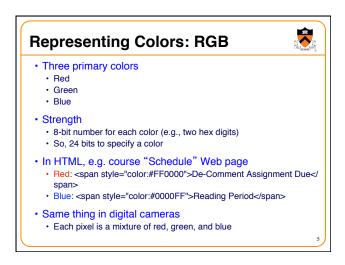
• Decimal (base 10)
• Each digit represents a power of 10
• 4173 = 4 x 10³ + 1 x 10² + 7 x 10¹ + 3 x 10⁰

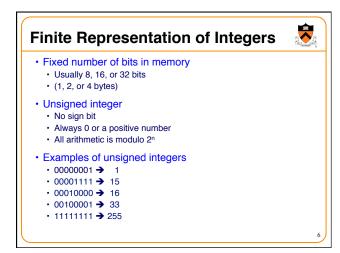
• Binary (base 2)
• Each bit represents a power of 2
• 10110 = 1 x 2⁴ + 0 x 2³ + 1 x 2² + 1 x 2¹ + 0 x 2⁰ = 22

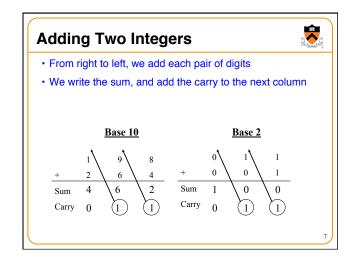
Decimal to binary conversion:
Divide repeatedly by 2 and keep remainders

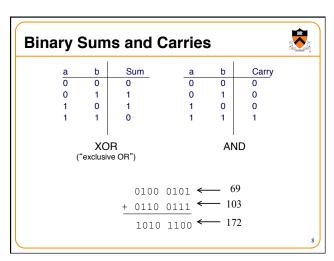
12/2 = 6 R = 0
6/2 = 3 R = 0
3/2 = 1 R = 1
1/2 = 0 R = 1
Result = 1100
```

```
Writing Bits is Tedious for People
 • Octal (base 8)
• Digits 0, 1, ..., 7
 • Hexadecimal (base 16)
    • Digits 0, 1, ..., 9, A, B, C, D, E, F
      0000 = 0
                    1000 = 8
                                    Thus the 16-bit binary number
                    1001 = 9
      0001 = 1
      0010 = 2
                    1010 = A
                                         1011 0010 1010 1001
      0011 = 3
                    1011 = B
      0100 = 4
                    1100 = C
                                         converted to hex is
      0101 = 5
                    1101 = D
      0110 = 6
                    1110 = E
                                                B2A9
      0111 = 7
                    1111 = F
```









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 2ⁿ doesn't change the answer For eight-bit number, n=8 and 2ⁿ=256 E.g., (37 + 256) mod 256 is simply 37 This can help us do subtraction by changing it to addition... Suppose you want to compute a - b Note that this equals a - b + 256 = a + (256 - b) How to compute 256 - b?

```
One's and Two's Complement

• What's easy is computing 255 - b (in 8 bits)

• Because it's 11111111 - b, so just flip every bit of b

• E.g., if b is 01000101 (i.e., 69 in decimal)

• 255 - b

1111 1111

- 0100 0101 ← b

1011 1010 ← 255 - b = 88

• This is called the one's complement of b; just flip all the bits of b

• Two's complement

• Add 1 to the one's complement

• E.g., 256 - 69 = (255 - 69) + 1 → 1011 1011
```

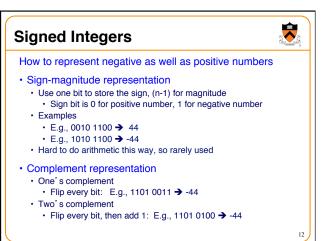
```
Putting it All Together
 • Computing "a - b"
   • Same as "a + 256 – b" (in 8-bit representation)
   • Same as "a + (255 - b) + 1"
   • Same as "a + onesComplement(b) + 1"

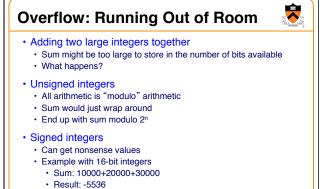
    Same as "a + twosComplement(b)"

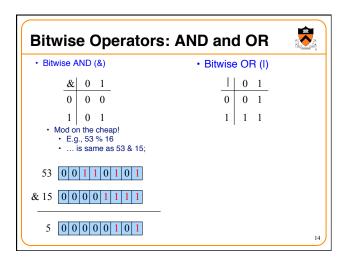
 • Example: 172 – 69
   • The original number 69: 0100 0101
   • One's complement of 69: 1011 1010

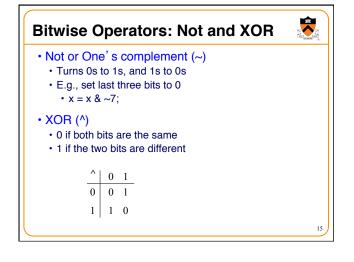
    Two's complement of 69: 1011 1011

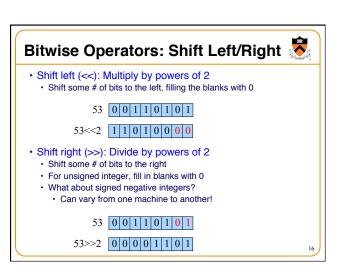
   • Add to the number 172: 1010 1100
   • The sum comes to:
                            0110 0111
                                              1010 1100
   · Equals: 103 in decimal
                                            + 1011 1011
                                             10110 0111
```











Example: Counting the 1's



- How many 1 bits in a number?
 - · E.g., how many 1 bits in the binary representation of 53?

0 0 1 1 0 1 0 1

- Four 1 bits
- · How to count them?
 - · Look at one bit at a time
 - · Check if that bit is a 1
 - · Increment counter
- · How to look at one bit at a time?
 - · Look at the last bit: n & 1
 - Check if it is a 1: (n & 1) == 1, or simply (n & 1)

17

#include <stdio.h> #include <stdib.h> int main(void) { unsigned int n; unsigned int count; printf("Number: "); if (scanf("%u", &n) != 1) { fprintf(stderr, "Error: Expect unsigned int.\n"); exit(EXIT_FAILURE); } for (count = 0; n > 0; n >>= 1) count += (n & 1); printf("Number of 1 bits: %u\n", count); return 0; }

Summary



- 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
- · Bitwise operators
 - AND, OR, NOT, and XOR
 - · Shift left and shift right
 - Useful for efficient and concise code, though sometimes cryptic

19