Bit Models

Section 2.1

Introduction

- What's the difference between an **int** and a **float** variable?

- One can hold integers and one can hold real numbers
- More specifically, the way the two types of numbers are represented in memory

Bits

- A **bit** is the basic unit of memory on computers
  - Short for **binary digit**
  - Can only have two possible values: **0** or **1**
  - In hardware, corresponds to voltage level
    - 0 is low
    - 1 is high

- Since a single bit can only hold two possible values, often group bits together
  - How many different possible values does four bits have?
  - How many different combinations of on / off could you have with four different light switches?

- $2^4 = 16$ possible combinations
- In general, if we have n bits, there's 2^n possible combinations
- A byte has 8 bits, so it has $2^8 = 256$ possible combinations

Bytes

- Most commonly, bits are grouped together in eights
- Eight bits together is known as a **byte**
- Measure variable sizes in bytes
  - char is 1 byte, int is 4 bytes, etc.
- Memory and storage often measured in forms of bytes
  - 4 GB, 1 TB, etc.

Bit Models

- A grouping of bits can have a variety of meanings
  - All data stored on the computer is represented as bits
- Part of the meaning of bits is how they're interpreted
- Interpretations of bits are known as **bit models**
- Different data types use different bit models

Magnitude Only Bit Model

- Represents nonnegative whole numbers
  - Unsigned numbers
- Each bit represents a power of 2
  - If a bit is **0**, its value is **0**
  - If a bit is **1**, its value is $\boldsymbol{2^{position}}$
- Added together, the bits represent the value stored in that collection of bits

Bits	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**
Place Value	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
In Base 10	128	64	32	16	8	4	2	1

- $0 + 0 + 0 + 2^4 + 0 + 0 + 2^1 + 2^0 = 19$
- $16 + 2 + 1 = 19$

- Left most bit is known as **most significant bit**
  - Has biggest impact on number
- Right most bit is known as **least significant bit**
  - Has smallest impact on number

- With 8 bits (1 byte)
  - Biggest value possible:
    - $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 255$
      - $128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$
  - Smallest value possible is:
    - $0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 = 0$

- Adding numbers using this bit model similar to how we usually add
- Might have to carry

7	00000111
+ 4	+ 00000100
= **11**	= **00001011**

- **$2^{number\_of\_bits}$ = number of possible values**
- **$2^{number\_of\_bits}$ - 1 = max possible value**
  - 0 counts as one of the values, so biggest possible value is one less than number of possible values
- **0 = smallest possible value**
  - This bit model used for nonnegative numbers

- Unsigned integer types use **magnitude only bit model**
  - unsigned char
  - unsigned short int
  - unsigned int
  - unsigned long int

Practice

- Supposing magnitude only bit model..
- What's 010101101 in decimal?
- What's 00110011 in decimal?
- What's 0101 + 0101?

Sign Magnitude Bit Model

- Often want to represent negative whole numbers as well
- In **sign magnitude bit model**, most significant bit represents the sign of the number (positive or negative)
  - 0 is positive, 1 is negative
- Other bits are like in magnitude only bit model

Bits	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**
Place Value	**sign**	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
In Base 10	+ or -	64	32	16	8	4	2	1

- Two problems with sign magnitude bit model:
  - Two zeros
    - 00000000 and 10000000
  - Addition more complicated
- Not used for any data types in C

Two's Complement Bit Model

- Represents integers (positive, zero, and negative)
- Zero and positive integers represented like in **magnitude only bit model**

- To write a negative number in two's complement:
  - Write the positive version of the number in binary
  - Invert / flip all the bits
  - Add 1
- Result is the negative number in two's complement

1. Write in Binary:

$11_{dec} = 00001011_{bin}$

2. Invert Bits:

00001011

→ **11110100**

3. Add 1:

11110100

+ 00000001

= 11110101

Result:

-11 is **11110101**

- In two's complement bit model, numbers that start with **1** are **negative**
- Numbers that start with **0** are **positive** (or zero)
- To convert a positive binary number to decimal, same as with magnitude only bit model
- To convert a negative binary number to decimal, similar steps as to convert a negative decimal to binary

1. Write in Binary:

11110101$_{bin}$ = ???$_{dec}$

2. Invert Bits:

11110101

→ **00001010**

3. Add 1:

00001010

+ 00000001

= 00001011

Result:

00001011 is 11,

thus **11110101**$_{bin}$ = -11$_{dec}$

Practice

- Supposing two's complement bit model
- What's -20 in binary?
- What's 110111110 in decimal?

Fixed Point Bit Model

- Thus far, bit models all been for integers
- Want to be able to represent real numbers
- One way would be to use some bits to represent left side of decimal point and some bits for right side of decimal point

Bits	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**
Place Value	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
In Base 10	16	8	4	2	1	0.5	0.25	0.125

- $2^4 + 2^1 + 2^{-2} + 2^{-3} = 18.375$
- $16 + 2 + 0.25 + 0.125 = 18.375$

- Decimal point doesn't move around, thus known as a **fixed point bit model**
- Generally not considered the best use of bits

Floating Point Bit Model

- Another way for representing real numbers is floating point bit model
  - Can represent a wider range of numbers with same amount of bits
- General idea is to use scientific notation
  - **$123.456 = 1.23456 \times 10^2$**

- In binary (base 2) scientific notation looks like:
  - **18.375 = 10010.011 = 1.0010011 × 2^4**
    - **$2^4 + 2^1 + 2^{-2} + 2^{-3}$**
    - **16 + 2 + 0.25 + 0.125**

- We know the number is always going to be 2 to some power, so it's not necessary to store the **2**
  - **1.0010011 × 2^4**

- Also, we can assume the number to the left of the decimal will always be **1** to further save space
  - **1.0010011 × 2^4**

- Thus, there's three pieces of information that need to be stored to represent a number in floating point bit model:
  - **Sign – either plus or minus**
  - **Fraction – numbers after decimal place**
  - **Exponent – power that 2 is raised to**
    - $\pm 1.fraction \times 2^{exponent}$

- Standard floating point bit model is called **IEEE 754 floating point standard**
  - Used in a lot of programming languages for floats
  - Uses 32 bits (4 bytes)
    - **1 bit for sign**
    - **8 bits for exponent**
    - **23 bits for fraction**

Parts	sign	exponent	fraction
Bit Places	31	30..23	22..0

- Sign is either 0 (positive) or 1 (negative)
- Exponent is an integer
  - In the range of 128 to -127
- Bits in the fraction represent negative powers of two
  - $2^{-1} = 0.5$, $2^{-2} = 0.25$, etc.
- Can represent about eight base 10 digits with a float

- To convert a decimal number to binary
  - Write the sign bit
  - Write the number in fixed point binary
    - Number to left of decimal as positive powers of two, fractional part as negative powers of two
  - Normalize
    - Move the decimal place to the right of the left most 1

- Take numbers to right of decimal point, zero padded, as fraction
  - Zero padded = add zeros to the right until the correct number of bits (23 bits)
- Take exponent (how many times moved the decimal point to the left), adding the bias of 127
  - Bias basically a way of storing positive and negative numbers not using two's complement
  - This is the value stored in the exponent portion of a float

Convert to Binary:

-118.625

1. Get Sign Bit:

s = 1

2. Write in Fixed Point:

118.625 = 1110110.101

$$2^6+2^5+2^4+2^2+2^1+2^{-1}+2^{-3}$$

3. Normalize:

$$1110110.101 \rightarrow 1.1101101101 \times 2^6$$

4. Get Fraction and Pad:

f = 110 1101 0100 0000 0000 0000

5. Get Exponent and Add Bias:

e = 6 + 127 = 133 = 1000 0101

Result:

1 1000 0101 110 1101 0100 0000 0000 0000

- To convert from float in binary to a float decimal..

Convert to Decimal:

11000010111011010100000000000000

1. Rewrite with Spaces:

1 1000 0101 110 1101 0100 0000 0000 0000

2. Break into Parts:

1 1000 0101 110 1101 0100 0000 0000 0000

sign = 1 (-)

exponent = 1000 0101 - 127

fraction = 110 1101 0100 0000 0000 0000

3. Convert Exponent to Decimal:

exponent = 1000 0101 - 127 =

$2^7 + 2^2 + 2^0 - 127 =$

133 - 127 = 6

4. Expand Scientific Notation:

$-1.110\ 1101\ 0100\ 0000\ 0000\ 0000 \times 2^6 =$

-1110 110.1 0100 0000 0000 0000

5. Convert the Left of Decimal:

-1110 110 =

$-(2^6 + 2^5 + 2^4 + 2^2 + 2^1) =$

-118

So far we have:

-118.1 0100 0000 0000 0000

6. Convert the Fractional Part:

1 0100 0000 0000 0000 =

$2^{-1} + 2^{-3} = 0.625$

Result:

-118.625

- float data type uses this bit model to store a real number in 4 bytes (32 bits)
- double uses same idea to store a real number in 8 bytes (64 bits)

- 1 bit for sign
- 11 bits for exponent
- 52 bits for fraction

ASCII Bit Model

- For representing non-numeric data, have **ASCII bit model**
  - American Standard Code of Information Interchange
- Really, ASCII uses 7 bits
  - 128 different bit patterns

- But since smallest typical grouping of bits is eight, its padded with an extra bit
- So all ASCII codes start with 0 in C

- ASCII values correspond to common keyboard characters
  - Letters of the alphabet
  - Digits
  - Punctuation symbols
- Also, some control characters
  - Some are obsolete nowadays

- char data type is used to store ASCII codes
- char can also be used as a signed integer (-128 to 127)
- And as an unsigned integer (0 to 255)
- Using %c in printf() interprets a char variable as an ASCII code
- Using %d in printf() interprets a char variable as a signed integer

Unicode Bit Model

- ASCII could only represent common keyboard characters
- Many more characters used in the world
- **Unicode bit model** was developed for representing more characters
  - Each character is a 16-bit pattern
  - 65,536 possible symbols

- In C, a short int is often used for representing a Unicode character