Possible Word Sizes

The size of a word determines how big of number can be stored in it

- 8 bits - students in this class
- 16 bits - students in Maryland
- 32 bits - people in the US
- 64 bits - people in the world

Wider words also mean faster access (more bits read/written in parallel)

What to do with Memory

- Store information
  - data
  - code
- Need to decide how to represent data
  - depends on what we want to represent
- positive numbers can use binary
- BCD is also possible to represent positive numbers
  - each decimal digit is encoded in binary
  - good if we need to convert to decimal frequently
  - generally not used in new computers today
- Can also use
  - HEX
  - Octal

Representing Data (cont)

- how to represent letters
  - assign a bit pattern to each character
  - most common way is ASCII
  - 7 bits are enough for the alphabet, numbers, and punctuation

Character	Binary Code
A	100 0001
B	100 0010
...	
Z	101 1010
0	011 0000
...	
9	011 1001

Character	Binary Code
a	110 0001
z	111 0010
<spac>	110 0001

How do we represent negative numbers?

- Can use signed magnitude
  - binary representation of the number (absolute value)
  - extra bit to indicate if the number is negative
  - have two ways to represent zero
- Complements
  - 1's complement (flip all the bits)
  - 2's complement (flip all the bits and add one)
    - only one representation of zero

-1	1111
-3	1101
-4	1100

How Do computers Subtract?

- Use complements
  - take 2's complement of subtrahend
  - add to minuend

6 - 13	6 - (-13)
00000110	00000110
11110011	00001101
11111001 = -7	00010011

What if the result is too big to represent?

- This is call overflow
- Need to know if the numbers are signed or not
- for unsigned numbers
  - if we have a carry out from the most significant digit
- for signed numbers
  - can overflow when
    - adding two positive numbers
    - adding two negative numbers
  - can detect overflow using carry into and out of sign bit
    - if they are different, then an overflow has occurred

```
70    0 1000110        -70    1 0111010
80    0 1010000        -80    1 0110000
150   1 0010110        -150   0 1101010
```

How do we represent real numbers?

- Each number has two parts
  - mantissa (represents a number between -1 and 1)
  - exponent (designates the position of the decimal point)
  - number = $m\ r^{e}$ where r is the radix
    - $6132.789 = +0.6132789 \times 10^{4}$
- A number is normalized if the most significant digit of mantissa is non-zero.
- Computers normally use a radix of 2
  - $1001.11 = .1001110 \times 2^{4}$
  - $10.5 = 1010.1$ or 10101×2^{-1}
  - $17.451 = 111.011100 = 1110111 \times 2^{-4}$

Converting to Floating Point

- 17.451 = 111.011100 = 1110111 x 2^{-4}

0.451		1/2
2		
0.902	0	1/4
2		
1.802	1	1/8
2		
1.608	1	1/16
2		
1.216	1	1/32
2		
0.432	0	1/64
2		
0.864	0	1/128

Floating Point (cont)

- Computers have only a fixed number of digits so we must pick a size for the mantissa and exponent
  - may not be possible to represent the number exactly
  - some numbers may be too big (or too small) to represent
- Explain why we don't need to keep the leading one in a normalized floating point number.

Other ways to represent information

- Gray Codes
  - like we used in K-maps
  - only one bit position changes in each number
    - useful in A/D applications
  - review with quick example
- Excess-3 codes
  - add 3 to BCD representation
  - ensures that all zero's is not a valid bit pattern.
- 2 of 5 code
  - use 5 bits to represent 0-9
  - always have exactly two ones
    - provides error detection (if don't have two ones)

Codes (cont)

1 00011	6 01100
2 00101	7 10001
3 00110	8 10010
4 01001	9 10100
5 01010	10 11000

- Other codes are also used for error detection
  - parity (use one extra bit to check for an error)
    - odd parity - any info has an odd # of one's
    - even parity - any info had an even # of one's
- Can combine codes
  - Consider zip+4 code on mail
  - explain code and pass it around
- How do we build an error detection circuit?
  - show parity generator and detector.