Numerical formats

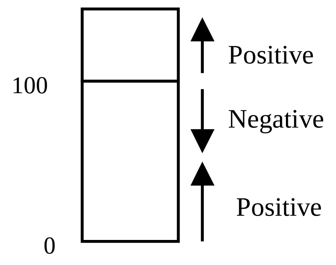
- What’s the main idea? Want to represent numbers (eg: 45, -12, 2.4556) using only bits.
- We’ve already seen (or you can read in the book) how to all these numbers to binary. We can then use one bit for each binary digit. Doesn’t that solve the problem???
- Well, not quite: we still need a way to take into account the sign, and the decimal point.

Adding signs

- Let’s now add the concept of signs
- Simplest way: Add a sign bit. This is called sign magnitude.
  - Example: 93 is 0101 1101. Then -93 is: **1**101 1101
  - Advantage: Very easy to take the negation of a number.
  - Disadvantage: hard to do additions. Need something better!

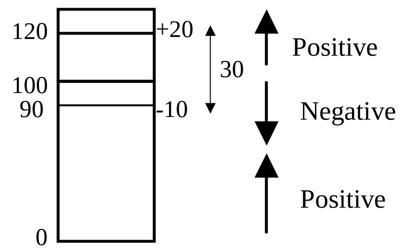
2's complement

- The motivation here is to make adding signed numbers VERY easy. Notice the following: if you ignore errors of "off by 100", then 0 and 100, look very similar...



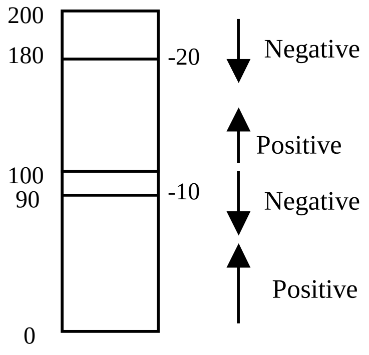
2's complement (cont'd)

- So, a positive number A is represented by A itself.
- A negative number -A is represented by 100 - A.
- Example: -10 + 30 = ??



2’s complement (cont’d)

- Another example: (-10) + (-10) = ??



2’s complement (cont’d)

- Wow, this works! We can add numbers without thinking about their sign! Let’s do this in binary.
- Example with 4 bits. Instead of using decimal 100 as 0 mark, we use binary 10000. So -A is represented by 10000 - A. Everything is the same just that we do things in binary.
- Shortcut for computing 2’s complement.
  - 10000 - A = (1111 + 1) - A = (1111 - A) + 1
  - What does 1111 - A mean? It’s the same as inverting...
  - So, to take 2’s complement: invert bits and add 1.

2’s complement (cont’d)

- Example in 4 bits:
  - How do we represent -6?
  - 6 in binary is 0110
  - Invert: 1001
  - Add 1: 1010
  - Note that if we do the same again, we get the original bit stream.
  - Also note that the msb conveniently tells us the sign of the number.

2’s complement (cont’d)

- Example in 4 bits: -1 + 3

```
   1111
+  0011
-------
  10010
```

2’s complement (cont’d)

- Just one thing: Recall the example with decimals. What happens if we add 40 and 40? We get 80, but we interpret 80 as -20…
- This is called overflow. In 2’s complement addition, overflow occurs if both operands are positive but the result is negative, or if both operands are negative and the result is positive.
- In 2’s complement subtraction, overflow occurs if a positive number is subtracted from a negative number and the result is positive, or if a negative number is subtracted from a positive number and the result is negative.
- Don’t confuse carry and overflow...

1’s complement (cont’d)

- Problem with 2’s complement: hard to find the negation of a number because it involves “adding 1”
- This leads to 1’s complement. In 1’s complement, instead of subtraction from 100, you subtract from 99. This means that in binary, you only need to invert the bits!
- The price to pay is that when we pass the 100 mark, we’ll be off by 1 (eg: -10 + 30 leads to 89 + 30 = 119. But we want 120!).
- So, every time you pass the 100 mark, you need to add an extra 1.

1’s complement (cont’d)

- Example in 4 bits: -1 + 3

```
   1110
+  0011
-------
  10001
      1
-------
  10010
```