15-122 Review Session: ints and bits

Converting Between Bases

Converting between bases fundamentally means multiplying each digit by its base to the power of the position that digit is in. For example:

Example 1: Convert `0xB00F` to decimal

$$\begin{aligned}\texttt{B00F}_{[16]} &= \texttt{B}_{[16]} \times \texttt{1000}_{[16]} + \texttt{0}_{[16]} \times \texttt{100}_{[16]} + \texttt{0}_{[16]} \times \texttt{10}_{[16]} + \texttt{F}_{[16]} \times \texttt{1}_{[16]} \\ &= 11 \times 16^3 + 0 \times 16^2 + 0 \times 16^1 + 15 \times 16^0 \\ &= 45071\end{aligned}$$

Example 2: Convert `0xCAFE` to binary

0xC	0xA	0xF	0xE
1100	1010	1111	1110

We combine the above to form the final number: `1100 1010 1111 1110`

Pro Tip: Write down the hex / binary conversions on your cheat sheets!

Example 3: Convert $213_{[10]}$ to binary

Procedure to convert to binary:

1. Find i, the highest power of two that is less than the number n. So $2^i \leq n$ and $2^{i+1} > n$
2. Check every value of 2^j, starting from i and going to 0. If $2^j \leq n$, subtract 2^j from n and set the jth position of the result to 1.

Pro Tip: Write down the powers of two on your cheat sheets!

Current Number	**Highest Power of Two**	**Current Result**
213	128	1_______
85	64	11______
21	16	1101____
5	4	110101__
1	1	11010101

Negative Integer Representations

- We are generally interested in **integer representations**: how do we convert a string of bits into an integer?
- Our previous integer representation up until now did not consider **negative integers**.
- **Two's complement** is one representation that can handle negatives.

Two's Complement Description

One simple way of describing two's complement is that two's complement is the same as the unsigned case, except **the most significant digit is now flipped in sign**. Consider an 8-bit number. In the unsigned case, the 8th bit corresponds to 2^7 = **128**. In two's complement, the 8th bit corresponds to **-128** instead.

Example 4: Convert `11010101` to decimal using a two's complement representation.

-128	64	32	16	8	4	2	1
`1`	`1`	`0`	`1`	`0`	`1`	`0`	`1`

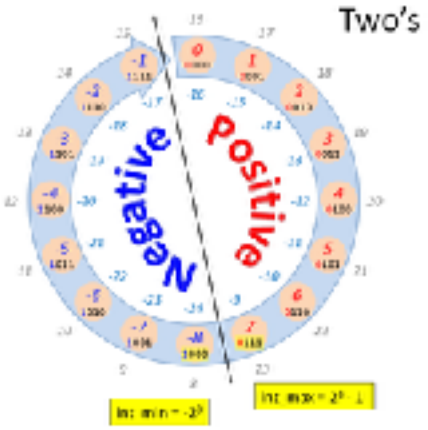
So as before we can add everything up:

$$
\begin{aligned}
\texttt{11010101}_{[2]} &= 1 \times -128 + 1 \times 64 + 0 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \\
&= -128 + 64 + 16 + 4 + 1 \\
&= -43
\end{aligned}
$$

Why Two's Complement?

Ask students: Why do we use two's complement instead of other representations (ex, same as before but with a sign bit)? What does two's complement give us?

Answer: Two's complement has the really nice properties of **modular arithmetic.** Importantly , the value of 0 - 1 is trivially -1 (without much extra work) and the value of int_max() + 1 is int_min().



Bit Operations

&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

~	0	1
	1	0

Above: Four common bit operations

Pro Tip: Write the bit operation truth tables in your cheat sheet!

Ask Students: What is the difference between the bitwise operators (&, |, ^, ~) and their corresponding logical operators (&&, ||, !=, !)?

Answer: The bitwise operators are used on ints and yield an int, with the operator applied to each corresponding bit. The logical operators are used on bools, with the operator applied to the values themselves.

- The **left and right shift** (<<, >>) operators are used to manipulate the positions of bits:
  - The left shift operator is a **logical left shift**. So bits "coming in" are set to zero.
  - The right shift operator is an **arithmetic right shift**. So bits "coming in" are set to the **sign bit** of the original number
    - **Ask students:** Why is this done? Answer: To preserve two's complement sign on right shift.
  - Note, **you can't right shift more than 31 times!** Leads to a divide by zero error.

Bit Operation Uses and Intuition

- The left / right shift operators (<< / >>), along with bitwise and (&), are useful for **masking.**
  - Example: Pixel assignment, colors
- XOR is useful for "flipping" bits.

Example 5: You are a developer for a revolutionary new API called HouseKit for the computing megacorporation Pear. HouseKit saves the on/off state of each appliance in the house with a bit being on/off (there are a max of 32 appliances).

Questions:
- How do you turn EVERYTHING on? Off?
- How do you toggle the state of individual appliances?
- How do you toggle an appliance or appliances on or off?
- How do you see which appliances two houses have turned on at the same time?
- How do you see which appliances either one of two houses has turned on?