CS 61C Fall 2020 — Number Representation

Discussion 1: August 31st, 2020

1 Pre-Check

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 Depending on the context, the same sets of bits may represent different things.

True. Like Two's complement and Bias notation.

1.2 It is possible to get an overflow error in Two's Complement when adding numbers of opposite signs.

True. Like 5 + (-5) = 0, but in binary, it's 0101b + 1011b = (1)0000, there's an overflow

False! Overflow means the result falls out of the range of [-2^(N-1), 2^(N-1) - 1].

1.3 If you interpret a N bit Two's complement number as an unsigned number, negative numbers would be smaller than positive numbers.

False. Like -1 = 1111b, 1 = 0001b, In unsigned number system, 1111b = 15, 15 > 1.

1.4 If you interpret an N bit Bias notation number as an unsigned number (assume there are negative numbers for the given bias), negative numbers would be smaller than positive numbers.

True. Let bias = 2^(N-1) - 1, Bias notation sets the zero point to bias in binary. In this case, all positive numbers will be larger than negative numbers when interpreted as unsigned numbers.

2 Unsigned Integers

2.1 If we have an n-digit unsigned numeral $d_{n-1}d_{n-2}\ldots d_0$ in *radix* (or *base*) r, then the value of that numeral is $\sum_{i=0}^{n-1} r^i d_i$, which is just fancy notation to say that instead of a 10's or 100's place we have an r's or r^2's place. For the three radices binary, decimal, and hex, we just let r be 2, 10, and 16, respectively.

Let's try this by hand. Recall that our preferred tool for writing large numbers is the IEC prefixing system:

Ki (Kibi) $= 2^{10}$	Gi (Gibi) $= 2^{30}$	Pi (Pebi) $= 2^{50}$	Zi (Zebi) $= 2^{70}$
Mi (Mebi) $= 2^{20}$	Ti (Tebi) $= 2^{40}$	Ei (Exbi) $= 2^{60}$	Yi (Yobi) $= 2^{80}$

(a) Convert the following numbers from their initial radix into the other two common radices:

1. 0b10010011 = 147= 0x93
2. 63 = 0b0111111 = 0x3F
3. 0b00100100 = 36 =0x24
4. 0 = 0b0 = 0x0
5. 39 = 0b100111 = 0x27
6. 437 = 0b000110110101 = 0x1B5
7. 0x0123 = 0b000100100011 = 291

(b) Convert the following numbers from hex to binary:

1. 0xD3AD = 0b1101 0011 1010 1101
2. 0xB33F= 0b 1011 0011 0011 1111
3. 0x7EC4 = 0b0111 1110 1100 0100

(c) Write the following numbers using IEC prefixes:

- 2^{16} 64Ki
- 2^{27} 128Mi
- 2^{43} 8Ti
- 2^{36} 64Gi
- 2^{34} 16Gi
- 2^{61} 2Ei
- 2^{47} 128Ti
- 2^{59} 512Pi

(d) Write the following numbers as powers of 2:

- 2 Ki 2(11)
- 512 Ki 2(19)
- 16 Mi 2(24)
- 256 Pi 2(58)
- 64 Gi 2(36)
- 128 Ei 2(67)

3 Signed Integers

3.1 Unsigned binary numbers work for natural numbers, but many calculations use negative numbers as well. To deal with this, a number of different schemes have been used to represent signed numbers, but we will focus on two's complement, as it is the standard solution for representing signed integers.

- Most significant bit has a negative value, all others are positive. So the value of an n-digit two's complement number can be written as $\sum_{i=0}^{n-2} 2^i d_i - 2^{n-1} d_{n-1}$.
- Otherwise exactly the same as unsigned integers.
- A neat trick for flipping the sign of a two's complement number: flip all the bits and add 1.

- Addition is exactly the same as with an unsigned number.
- Only one 0, and it's located at 0b0.

For questions (a) through (c), assume an 8-bit integer and answer each one for the case of an unsigned number, biased number with a bias of -127, and two's complement number. Indicate if it cannot be answered with a specific representation.

(a) What is the largest integer? What is the result of adding one to that number?

1. Unsigned? 255. Adding one to 127 will cause overflow. No overflow! The result will be 0.
2. Biased? 128. The result will be -127
3. Two's Complement? 127. The result will be -128.

(b) How would you represent the numbers 0, 1, and -1?

1. Unsigned? 0b0000 0000, 0b0000 0001. No -1.
2. Biased? 0b0111 1111, 0b1000 0000, 0b0111 1110
3. Two's Complement? 0b0000 0000, 0b0000 0001, 0b1111 1111

(c) How would you represent 17 and -17?

1. Unsigned? 0b 0001 0001, No -17.
2. Biased? 0b1001 0000 0b0110 1110
3. Two's Complement? 0b 0001 0001 0b 1110 1111

(d) What is the largest integer that can be represented by *any* encoding scheme that only uses 8 bits?

No such integer. It can represent from 0 to 255, but also anything else. Like 1 to 256,

(e) Prove that the two's complement inversion trick is valid (i.e. that x and $\overline{x}+1$ sum to 0).

It's obvious that x + x_hat always equals to all 1 in all bits. Then we add one to it, all 1s will be 0s, with a 1 above the top position.

(f) Explain where each of the three radices shines and why it is preferred over other bases in a given context.

4 Arithmetic and Counting

4.1 Addition and subtraction of binary/hex numbers can be done in a similar fashion as with decimal digits by working right to left and carrying over extra digits to the next place. However, sometimes this may result in an overflow if the number of bits can no longer represent the true sum. Overflow occurs if and only if two numbers with the same sign are added and the result has the opposite sign.

(a) Compute the decimal result of the following arithmetic expressions involving 6-bit Two's Complement numbers as they would be calculated on a computer. Do any of these result in an overflow? Are all these operations possible?

1. 0b011001 − 0b000111 =0x19 - 0x07 = 0x12 = 0b0001 0010 = 0b010010
2. 0b100011 + 0b111010 = 0x23 + 0x3A = 0x5D = 0b 0101 1101, there's an overflow.
3. 0x3B + 0x06 = 0x42 = 0b 0100 0010 = -30, 0x3b = 0b111011 = -5, 0x06 = 6, result = 1, overflow.
4. 0xFF − 0xAA These two numbers can not be represented with 6-bit numbers.

(b) What is the least number of bits needed to represent the following ranges using any number representation scheme.

Result should be 0x41, and we need to truncate the first 1.
So the actual result is 0b 00 0001 = 1 in decimal, no overflow!

1. 0 to 256 9 bits
2. -7 to 56 6 bits
3. 64 to 127 and -64 to -127 7 bits
4. Address every byte of a 12 TiB chunk of memory 3 * 2^(39) bytes. 41 bits needed.

TiB = Ti Bytes, not bits! Thus we need 44 bits.