

# CS61C Summer 2016 Discussion 0 – Number Representation

## 1 Unsigned Integers

If we have an  $n$ -digit unsigned numeral  $d_{n-1}d_{n-2}\dots d_0$  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 binary, decimal, and hex we just let  $r$  be 2, 10, and 16, respectively.

Recall also that we often have cause to write down large numbers, and our preferred tool for doing that is the IEC prefixing system:

- Kilo- (Ki) =  $2^{10} \approx 10^3$
- Mega- (Mi) =  $2^{20} \approx 10^6$
- Giga- (Gi) =  $2^{30} \approx 10^9$
- Tera- (Ti) =  $2^{40} \approx 10^{12}$
- Peta- (Pi) =  $2^{50} \approx 10^{15}$
- Exa- (Ei) =  $2^{60} \approx 10^{18}$
- Zetta- (Zi) =  $2^{70} \approx 10^{21}$
- Yotta- (Yi) =  $2^{80} \approx 10^{24}$

### 1.1 We don't have calculators during exams, so let's try this by hand

1. Convert the following numbers from their initial radix into the other two common radices:

(a)  $0b10010011 = 147 = 0x93$

(e)  $0xB33F = 0b1011\ 0011\ 0011\ 1111 = 45887$

(b)  $0xD3AD = 0b1101\ 0011\ 1010\ 1101 = 54189$

(f)  $0 = 0b0 = 0x0$

(c)  $63 = 0b0011\ 1111 = 0x3F$

(g)  $0x7EC4 = 0b0111\ 1110\ 1100\ 0100 = 32452$

(d)  $0b00100100 = 36 = 0x24$

(h)  $437 = 0b0001\ 1011\ 0101 = 0x1B5$

2. Write the following numbers using IEC prefixes:

(a)  $2^{16} = 64\ \text{Ki}$

(e)  $2^{43} = 8\ \text{Ti}$

(b)  $2^{34} = 16\ \text{Gi}$

(f)  $2^{47} = 128\ \text{Ti}$

(c)  $2^{27} = 128\ \text{Mi}$

(g)  $2^{36} = 64\ \text{Gi}$

(d)  $2^{61} = 2\ \text{Ei}$

(h)  $2^{58} = 256\ \text{Pi}$

3. Write the following numbers as powers of 2:

(a)  $2\ \text{Ki} = 2^{11}$

(d)  $64\ \text{Gi} = 2^{36}$

(b)  $256\ \text{Pi} = 2^{58}$

(e)  $16\ \text{Mi} = 2^{24}$

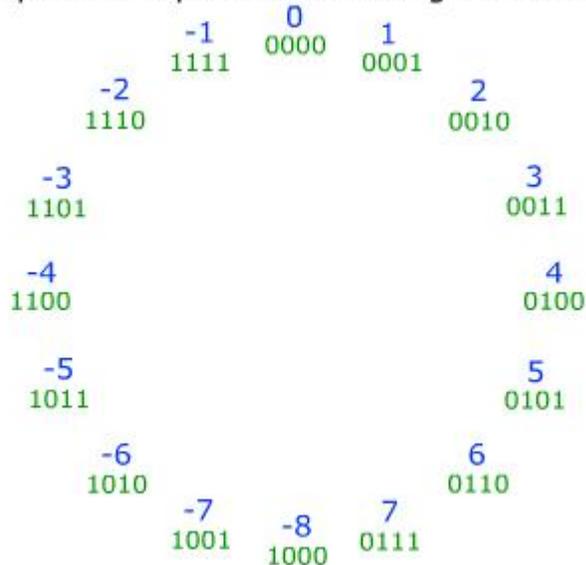
(c)  $512\ \text{Ki} = 2^{19}$

(f)  $128\ \text{Ei} = 2^{67}$

## 2 Signed Integers w/ Two's Complement

- Two's complement is the standard solution for representing **signed** integers.
  - Most significant bit has a negative value, all others have positive.
  - 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**.
- A schematic representation of a 4-bit two's complement is below.

### Two's Complement representation using 4 bit binary strings



Notice how the largest number is 7 whereas the smallest number is -8. And -8 corresponds to 0 in the circle.

- Addition is exactly the same as with an unsigned number.
- Arithmetic operations on binary numbers can lead to overflow. The easiest way to detect overflow is to keep in mind of the representational range of binary numbers.

### 2.1 Exercises

For the following questions assume an 8 bit integer. Answer each question for the case of a two's complement number and an unsigned number.

1. What is the largest integer? The largest integer + 1?
  - (a) [Unsigned:] 255, 0
  - (b) [Two's Complement:] 127, -128
2. How do you represent the numbers 0, 1, and -1?
  - (a) [Unsigned:] 0b0000 0000, 0b0000 0001, N/A
  - (b) [Two's Complement:] 0b0000 0000, 0b0000 0001, 0b1111 1111
3. How do you represent 17, -17?

- (a) [Unsigned:] 0b0001 0001, N/A
- (b) [Two's Complement:] 0b0001 0001, 0b1110 1111

4. Prove that the two's complement inversion trick is valid (i.e. that  $x$  and  $\bar{x} + 1$  sum to 0). Note that for any  $x$  we have  $x + \bar{x} = 0b1\dots1$ . A straightforward hand calculation shows that  $0b1\dots1 + 0b1 = 0$ .
5. Explain where each of the three radices shines and why it is preferred over other bases in a given context. Decimal is the preferred radix for human hand calculations, likely related to the fact that humans have 10 fingers.

Binary numerals are particularly useful for computers. Binary signals are less likely to be garbled than higher radix signals, as there is more “distance” (voltage or current) between valid signals. Additionally, binary signals are quite convenient to design circuits with, as we'll see later in the course.

Hexadecimal numbers are a convenient shorthand for displaying binary numbers, owing to the fact that one hex digit corresponds exactly to four binary digits.

### 3 Bitwise Operator Trick: Masking

Masking is very commonly used in bitwise operations. Masking helps to either clear to extract bits from a bit string. AND a mask of all ones with the bit string extract the bits. XOR a mask of all ones with the bit string give the complement of the bit string.

#### 3.1 Exercise

1. Fill in the blank below to complete the function `mod32` such that the function will return the remainder of `x` when divided by 32. The first blank is a bitwise operator. The second blank should be a decimal number:

```
int mod32(int x)
{
    return x & 31;
}
```

2. Finish the function `check_nth_bit`. The function checks and returns the `n`th bit of the given argument `x`.

```
int checkNthBit (int x, int n)
{
    temp = x >> (n-1);
    return temp & 1;
}
```