1	2	4	8	16	32	64	128	256	512	1024
2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}

CS 61C Fall 2019 — Number Representation

Discussion 1: September 2, 2019

Notes

1 Unsigned Integers

We represent numbers as: $1028_{10} = 1\cdot10^3 + 0\cdot10^2 + 2\cdot10^1 + 8\cdot10^0$
base 10 = decimal
Binary is similar: $1101 = 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0$

1.1 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 the three radices binary, decimal, and hex, we just let r be 2, 10, and 16, respectively.

We don't have calculators during exams, so 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. 0b$\underbrace{1001}_{8+1=9}\underbrace{0011}_{2+1=3}$ = 147 = 0x93 $2^0+2^1+2^4+2^7 = 1+2+16+128 = 147$
2. 63 = 0b0011 1111 = 0x3F   we know 64 = 0b0100 0000 so 63 = 64−1 so 0b$\underbrace{0011}_{3}\underbrace{1111}_{F}$. now it's easy to convert it to hex: 0x3F
3. 0b$\underbrace{0010}_{2}\underbrace{0100}_{4}$ = 36 = 0x24 $2^2+2^5 = 4+32 = 36$
4. 0 = 0b0 = 0x0
5. 39 = 0b0010 0111 = 0x27   I know 32 in binary! (100000) 39 − 32 = 7 I know 7 in binary (111) combine then: 0b$\underbrace{0010}_{2}\underbrace{0111}_{7}$
6. 437 = 0b0001 1011 0101 = 0x1B5   see final page for how to
7. 0x0123 = 0b0000 0001 0010 0011 = 291 $2^0+2^1+2^5+2^8$ $1+2+32+256=291$

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

1. 0xD3AD = 0b1101 0011 1010 1101 = 54189 $2^0+2^2+2^3+2^5+2^7+2^8+2^9+2^{12}+2^{14}+2^{15}$
2. 0xB33F = 0b1011 0011 0011 1111 = 45887 $2^0+2^1+2^2+2^3+2^4+2^5+2^8+2^9+2^{12}+2^{13}+2^{15}$
3. 0x7EC4 = 0b0111 1110 1100 0100 = 32452 $2^2+2^6+2^7+2^9+2^{10}+2^{11}+2^{12}+2^{13}+2^{14}$

(c) Write the following numbers using IEC prefixes:

- 2^{16} = 64 Ki $2^{10}\cdot2^6$
- 2^{27} = 128 Mi $2^{20}\cdot2^7$
- 2^{43} = 8 Ti $2^{40}\cdot2^3$
- 2^{36} = 64 Gi $2^{30}\cdot2^6$
- 2^{34} = 16 Gi $2^{30}\cdot2^4$
- 2^{61} = 2 Ei $2^{60}\cdot2^1$
- 2^{47} = 128 Ti $2^{40}\cdot2^7$
- 2^{59} = 512 Pi $2^{50}\cdot2^9$

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

- 2 Ki = 2^{11} $2^1\cdot2^{10}$
- 512 Ki = 2^{19} $2^9\cdot2^{10}$
- 16 Mi = 2^{24} $2^4\cdot2^{20}$
- 256 Pi = 2^{58} $2^8\cdot2^{50}$
- 64 Gi = 2^{36} $2^6\cdot2^{30}$
- 128 Ei = 2^{67} $2^7\cdot2^{60}$

2 Signed Integers

2.1 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. 8 bits means 2^8 possible = 256 numbers

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

1. Unsigned? 255, 0
2. Biased? 128, -127 largest number + Bias = 255 − 127 = 128
3. Two's Complement? 127, -128 Two's complement has one positive zero. zero takes up the extra positive number.

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

1. Unsigned? 0b0000 0000, 0b0000 0001, N/A ← cannot represent unsigned numbers in unsigned
2. Biased? 0b0111 1111, 0b1000 0000, 0b0111 1110 0: 0 − Bias = −(−127) = 127 | 1: 1 − (−127) = 128 | −1: −1 − (−127) = 126

x + bias = n
so x = n − bias

3. Two's Complement? 0b0000 0000, 0b0000 0001, 0b1111 1111

↳ $-2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0 = -128+127=-1$
— or — flip bits of 1 and add 1:
0 0000001 → 1111 1110 + 1 = 1111 1111

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

1. Unsigned? 0b0001 0001, N/A $17 = 16+1 = 2^0+2^4$
2. Biased? 0b1001 0000, 0b0110 1110 17 + (128) = 144 | −17 + 127 = 110
3. Two's Complement? 0b0001 0001, 0b1110 1111

17: 0001 0001
like unsigned ↗ $-17 = -2^7+2^6+2^5+2^3+2^2+2^1+2^0$ or 17 = 1110 1110 + 1 = 1110 1111

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

There is no such integer. For example, an arbitrary 8-bit mapping could choose to represent the numbers from 1 to 256 instead of 0 to 255.

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

Note that for any x we have $x+\overline{x} = 0b1\ldots1$. A straightforward hand calculation shows that $0b1\ldots1 + 0b1 = 0$.

Ex: −17 : x = 1110 1111 $\overline{x}$ = 0001 0000

$x+\overline{x}$ = 1111 1111 + $x+\overline{x}+1$ = 1111 1111 + 0000 0001

overflow 0000 0000 ← all zeros.

(f) 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, as well 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 Counting

3.1 Bitstrings can be used to represent more than just numbers. In fact, we use bitstrings to represent *everything* inside a computer. And, because we don't want to be wasteful with bits it is important to remember that n bits can be used to represent 2^n distinct things. For each of the following questions, answer with the minimum number of bits possible.

(a) How many bits do we need to represent a variable that can only take on the values 0, π or e? 1 bit = 2 options

2 bits = 4 options

2

(b) If we need to address 3 TiB of memory and we want to address every byte of memory, how long does an address need to be? TiB = 2^{40}

$2^0 < 3$ & $2^1 < 3$ so $2^2 > 3$ so $2^2 \cdot 2^{40} =$ $\log_2(2^{42}) = 42$

42 bits

We need to be able to

(c) If the only value a variable can take on is e, how many bits are needed to represent it?

$\frac{1}{2}$ a bit (or, alternatively, 0 bits) since 1 bit can differentiate between two values

this may sound weird since you would need a bit to represent this.
The reason is zero or $\frac{1}{2}$ because you always know the values so you do not need a bit to reprepresent this & since the variable can take only one value, we do not need any bits to represent the single value.

1a6) ex: Convert from 10 -> 2. Division!

437:

$$2\overline{)437} = 218 \text{ r } 1, \quad 2\overline{)218} = 109 \text{ r } 0, \quad 2\overline{)109} = 054 \text{ r } 1, \quad 2\overline{)54} = 27 \text{ r } 0, \quad 2\overline{)27} = 13 \text{ r } 1, \quad 2\overline{)13} = 6 \text{ r } 1, \quad 2\overline{)6} = 3 \text{ r } 0, \quad 2\overline{)3} = 1 \text{ r } 1, \quad 2\overline{)1} = 0 \text{ r } 1$$

LSB   to   MSB

Finally get a zero.

We use floor division to get the remainder. Read from

This is general idea from going from higher radix to lower radix.

So

0b 1 1 0 1 1 0 1 0 1

Hex <-> Binary: Group into 4 ones + convert directly.

+

Binary -> Hex

A = 10 D = 13

B = 11 E = 14

C = 12 F = 15

1010 = 8+2 = 10 = A

B = 11 = 1011

Alternative for 437:

512 - 437 = 75

1000000000   1001011

$$\begin{array}{r} 1000000000 \\ -\ 0001001011 \\ \hline 0110110101 \end{array}$$