1 Lecture 3: Number bases

Information in a computer is best visualized as a string of 1's and 0's. If the information is a number, it is natural to store the number using (perhaps a small modification of) base 2. Before going on, we recall how we represent numbers in different bases.

1.1 Numbers in base β

If $\beta \geq 2$ is a natural number, and the digits c_j come from $\{0, 1, \ldots, \beta - 1\}$, then the string

$$c_N c_{N-1} \ldots c_0 . c_{-1} c_{-2} \ldots$$

is the *base β representation* of the number

$$x = \sum_{k=-\infty}^{N} c_k \beta^k. \tag{1}$$

Note that by comparing with the geometric series, we can show that this series converges. We will typically write $x = (c_N \ldots c_0 . c_{-1} \ldots)_\beta$ where the subscript β tells us the base of the expansion.

The most important cases are $\beta = 2, 8, 10$ and 16.

We consider a few examples:

Example. Rewrite in base 10.

$$x = (1001)_2, y = (0.11111\ldots)_2, \text{ and } z = (1 \ldots 1)_2$$

where z has $k + 1$ digits.

Solution. We have

$$x = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^1 + 1 \cdot 2^0 = (17)_1 0.$$

For y, we need to sum the infinite series,

$$y = \sum_{n=1}^{\infty} 2^{-n} = (0.5)_{10}$$

by the formula for a geometric series. To evaluate z, we must write the geometric sum in closed form,

$$1 + 2 + 2^2 + \ldots + 2^k = \frac{1 - 2^{k+1}}{1 - 2} = 2^{k+1} - 1.$$

∎

1.2 Converting to base 2.

If we evaluate the expression (1) using base 10 arithmetic, we can convert form base β to base 10. In this section, we consider how to convert from base 10 to base 2. It should be clear how to modify these procedures to other bases.

1.2.1 Conversion of whole numbers.

If we have a whole number x, we show how to write the binary expansion of x, $(c_N \dots c_0)_2$. We begin by dividing by 2, that is writing $x = 2 \cdot q + r$ where the remainder r is 0 or 1. The remainder r is units digit, c_0 and the quotient q has binary expansion $(c_N \dots c_1)_2$. Continuing in this manner, we can find additional digits by repeated division.

A simple example illustrates this procedure.

Example. Convert $(61)_{10}$ to base 2 by repeated division.

Solution. We write $61 = 2 \cdot 30 + 1$, then $30 = 2 \cdot 15 + 1$ and continuing gives

$$\begin{aligned} 61 &= 2 \cdot 30 + 1 \\ 30 &= 2 \cdot 15 + 0 \\ 15 &= 2 \cdot 7 + 1 \\ 7 &= 2 \cdot 3 + 1 \\ 3 &= 2 \cdot 1 + 1 \\ 1 &= 2 \cdot 0 + 1 \end{aligned}$$

we may read up the right-hand side and obtain the binary expansion is $(111101)_2$.

Another way to look at this is that we may insert each expression into its predecessor to obtain:

$$\begin{aligned} 61 &= 2 \cdot 30 + 1 \\ 61 &= 2 \cdot (2 \cdot 15 + 0) + 1 \\ 61 &= 2 \cdot (2 \cdot 15 + 0) + 1 \\ 61 &= 2 \cdot (2 \cdot (2 \cdot 7 + 1) + 0) + 1 \\ 61 &= 2 \cdot (2 \cdot (2 \cdot (2 \cdot 3 + 1) + 1) + 0) + 1 \\ 61 &= 2 \cdot (2 \cdot (2 \cdot (2 \cdot (2 \cdot 1 + 1) + 1) + 1) + 0) + 1 \\ 61 &= 2 \cdot (2 \cdot (2 \cdot (2 \cdot (2 \cdot (2 \cdot 0 + 1) + 1) + 1) + 1) + 0) + 1 \\ 61 &= 2^5 + 2^4 + 2^3 + 2^2 + 0 \cdot 2^1 + 1 \end{aligned}$$

We can read off the binary expansion from the next to the last expression and then the last expression is obtained by multiplying out the digits.

∎

For convenience, we summarize the procedure to write a decimal as a binary number. In this procedure, we let `x mod 2` denote the remainder when we divide by 2 and `x % 2` be the quotient.

```
input x
i = 0
while x > 0
 c_i = x mod 2
 x = x % 2
 i++
end while
```

Roughly speaking, each time we divide by 2, the remainder gives us one digit of the binary expansion and the remaining digits are shifted one place to the right to give us the expansion of the quotient.

1.2.2 Converting fractions to binary

If we have a number between 0 and 1 which is written in binary, then we can multiply by two to shift one digit to the left and then pick off one digit of the expansion. That is if $x = (0.c_{-1}c_{-2}\ldots)_2$, then $2x = (c_{-1}.c_{-2}\ldots)_2$. Now it is easy to pick off one binary digit as the integer part of 2x. We may repeat this process as often as necessary to get the full expansion.

We can summarize this procedure with the following which will return the (possibly infinite) binary expansion of a real number x between 0 and 1.

```
input x, a real number between 0 and 1
j=1

while x not equal 0
x = 2*x
c(j) = integer part of x
j = j+1
endwhile

return c(1)c(2)c(3)....
```

Again, we illustrate with an example.

Example. Convert 1/3 to binary.

We know that we have $1/3 = (0.3333\ldots)_{10}$. We multiply by 2, and then take the fractional part to give

$$\begin{aligned}
2 \cdot 0.333\ldots &= 0.6666\ldots \\
2 \cdot 0.666\ldots &= 1.333\ldots \\
2 \cdot 0.333\ldots &= 0.6666\ldots \\
2 \cdot 0.666\ldots &= 1.333\ldots \\
2 \cdot 0.333\ldots &= 0.6666\ldots \\
2 \cdot 0.666\ldots &= 1.333\ldots \\
2 \cdot 0.333\ldots &= 0.6666\ldots
\end{aligned}$$

By now, we should see a pattern and conclude that

$$1/3 = (0.\overline{01})_2$$

where we use the bar to indicating a repeating block of digits.

Example. Write the decimal fraction $1/10 = (0.1)_{10}$ in binary form.

Solution.

$$(0.1)_{10} = (0.00011001100\ldots)_2.$$

∎

If we round this expression to 23 binary digits, we obtain $y(0.00011001100110011001101)_2$ and the error $|1/10-y|$ is approximately 2^{-25}. One may read at `http://www.ima.umn.edu/ãrnold/disa` about one disaster related to this simple fact.