Arbitrary Sized Integers

Sam Moore, David Gow

July 7, 2014

Abstract

We have implemented arbitrary sized integers which sometimes don't segfault, using a combination of the C++ standard library and stand alone x86-64 assembly. Performance tests reveal that we would have been better off just using the GNU Multiprecision Library (GMP).

1 Integer Representation

A positive integer (natural number) can be written as the sum of smaller integers "digits" multiplied by powers of a base.

$$z = \sum_{i=0}^{\infty} d_i \beta^i \tag{1}$$

Where each digit $d_i < \beta$ the base. A set of β unique symbols are used to represent values of d_i. A fixed size representation truncates the sum at some $i = N$, which can represent all values $0 \leq z \leq \beta^{n+1} - 1$.

A seperate sign symbol (eg: '-') can be used to represent negative integers using the same digit sum.

Example in base 10 (decimal):

$$5682_{10} = 5 \times 10^3 + 6 \times 10^2 + 8 \times 10^1 + 2 \times 10^0 \tag{2}$$

In base 2 (binary) the same integer is:

$$1011000110010_2 = 1 \times 2^{12} + 0 \times 2^{11} + \ ... + 0 \times 2^0 \tag{3}$$

1.1 Representation on computer hardware

Computer hardware implements operations for fixed size integers. The base is $\beta = 2$ and the digits are $\{0, 1\}$. The most significant bit can be reserved for the sign instead of a digit.

We can construct larger size integers by considering some sequence of fixed size integers to be individual digits. In practice we will still be limited by the memory and processing time required

for "big" integers.

For example, we can represent 5682_{10} as a single 16 bit digit or as the sum of two 8 bit digits. Each digit is being written in base 2 or 10 because there is not a universal base with $\geq 2^8$ unique symbols.

$$5682_{10} = 0001011000110010_2 = 10110_2 \times 2^8 + 110010_2 \times 2^0 \tag{4}$$
$$= 22_{10} \times 2^8 + 50_{10} \times 2^0 \tag{5}$$

2 Addition Algorithms

3 Subtraction Algorithms

4 Multiplication Algorithms

5 Division Algorithms

5.1 Naive Algorithm

5.2 Shifting Algorithm

6 Base conversion

Since humans are not very good at understanding binary, it is convenient to convert integer representations from one base to another.

7 IPDF Integer Representations