USING A RECIPROCAL BASE IN NUMBER SYSTEMS

MOOSA NASIR
SEPTEMBER 12, 2022

Abstract. All current number systems, whether rational or irrational, use a base greater than 1. We propose a fractional base, specifically a base which is the reciprocal of common bases such as binary and decimal. **Keywords.** Base, Number Systems, Reciprocal

In number systems, the base represents the number of different digits a number can use when representing a quantity. Each place in a number can be representing as $b^p \times d$, where b is the base, p is the number place with $p = 0$ to the right of the decimal point, and d is the digit. For example, as shown in figure 1, 2743.54_{10} can be represented as $10^3 \times 2 + 10^2 \times 7 + 10^1 \times 4 + 10^0 \times 3 + 10^{-1} \times 5 + 10^{-2} \times 4$.

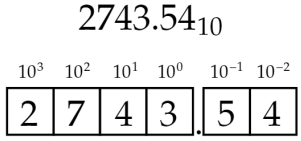

Figure 1. Example of a base 10 number

With this rule in mind, nothing stops us from using any value for b. Using the example in figure 1, we can change b to $\frac{1}{10}$. Following the same rule for p, 2743.54_{10} now becomes $453.472_{\frac{1}{10}}$ as shown in figure 2.

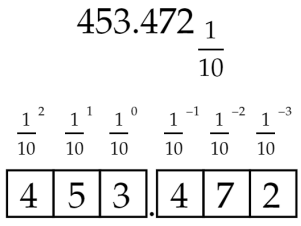

Figure 2. Example of a base $\frac{1}{10}$ number

The example shown in figure 1 has a terminating decimal. If we were to have a non-terminating decimal number such as $1.\bar{3}_{10}$, In base $\frac{1}{10}$, it would look like $\bar{3}1_{\frac{1}{10}}$ as shown in figure 3. This observation tells us that in reciprocal bases, non-terminating decimals will have a bar to the right of the decimal point after the ones place. Else the reciprocal base number would be have an infinite number of digits

After looking at a few examples, we can devise an algorithm to convert from base b to its corresponding base $\frac{1}{b}$. The simplest algorithm is to 1) reverse the digits of the number and

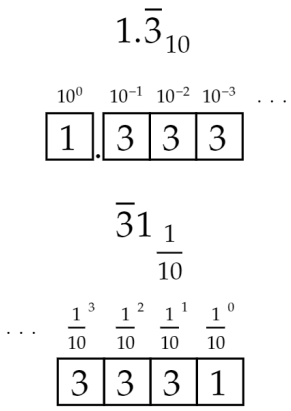

FIGURE 3. Example of non-terminating decimals in base 10 and base $\frac{1}{10}$

2) move the the decimal point down one place. This can be seen in figure 4. To prove that this algorithm works, $n_2 = n_{\frac{1}{2}} = 2.75$ where $n_2 = 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$ and $n_{\frac{1}{2}} = 1 \times \frac{1}{2}^2 + 1 \times \frac{1}{2}^1 + 0 \times \frac{1}{2}^0 + 1 \times \frac{1}{2}^{-1}$.

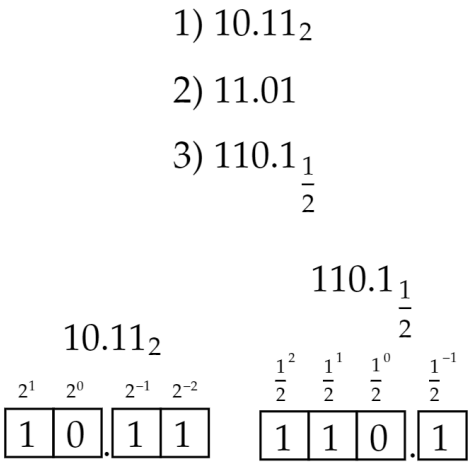

FIGURE 4. Example of using the algorithm for a base 2 number

Arithmetic done for numbers in reciprocal bases is very similar to arithmetic for whole base numbers, with the only difference being that the direction of arithmetic procedures are reversed, ex. left becomes right and vice versa.

Addition is by far the simplest arithmetic operation done in reciprocal bases. While in whole base arithmetic carrying is done to the left, carrying in reciprocal base arithmetic is done to the right, as shown in figure 5. In the example, we first align the numbers to their one to one number places, then we start from the left and sum the digits at each column, carrying one to the right if the sum is greater than the largest possible digit d, and moving the sum subtract d down to the final sum.

$$\begin{array}{r} {\scriptstyle 1} \\ 7.2_{\frac{1}{10}} \\ +\ 9.5_{\frac{1}{10}} \\ \hline 6.8_{\frac{1}{10}} \end{array}$$

FIGURE 5. Example of adding two base $\frac{1}{10}$ numbers

Subtraction is also a fairly easy operation in reciprocal bases. The steps will be done to the right instead of to the left like whole base arithmetic, as shown in figure 6. In the example we first try to subtract 9 from 7. Since this would give us a negative difference, we will carry 1 from the tenths place, and subtract 1 from it as well. Now we subtract 9 from 17, which gives us 8. The next digit is the difference of $7 - 1$ and 5, which gives us 1.

$$\begin{array}{r} {\scriptstyle 1}\ {\scriptstyle -1} \\ 7.7_{\frac{1}{10}} \\ -\ 9.5_{\frac{1}{10}} \\ \hline 8.1_{\frac{1}{10}} \end{array}$$

FIGURE 6. Example of subtracting two base $\frac{1}{10}$ numbers

Multiplication is a little tricky understand but after a while, it will become easy. It is very similar to whole base arithmetic, as shown in figure 7. In the example we can see that instead of moving our 2 to the left, we move to the right, past the decimal point. This makes sense because $1 \times \frac{1}{10}^{0} + 2 \times \frac{1}{10}^{-1}$ is equal to 21, which is the base 10 result of 7×3.

$$\begin{array}{r} 7_{\frac{1}{10}} \\ \times\ 3_{\frac{1}{10}} \\ \hline 1.2_{\frac{1}{10}} \end{array} \qquad \begin{array}{r} {\scriptstyle 2} \\ 7_{\frac{1}{10}} \\ 7_{\frac{1}{10}} \\ +\ 7_{\frac{1}{10}} \\ \hline 1.2_{\frac{1}{10}} \end{array}$$

FIGURE 7. Example of multiplying two base $\frac{1}{10}$ numbers

Division is not much different from multiplication in terms of difficulty. We can apply the same rules and thought process. As long as you know division in base b it will be trivial to do it in base $\frac{1}{b}$. As shown in figure 8, the process is extremely similar to division in base b.

$$7_{\frac{1}{10}} \div 3_{\frac{1}{10}} = 7_{\frac{1}{10}} \times \overline{3}0_{\frac{1}{10}}$$

$$\begin{array}{r} 7_{\frac{1}{10}} \\ \times\ \overline{3}0_{\frac{1}{10}} \\ \hline \overline{3}2_{\frac{1}{10}} \end{array}$$

FIGURE 8. Example of dividing two base $\frac{1}{10}$ numbers

Scientific Notation is also possible in base $\frac{1}{b}$. For example, in base 10, the scientific notation of 743 is 7.43×10^2. In base $\frac{1}{10}$, $3.47_{\frac{1}{10}}$ (which is 743 in base $\frac{1}{10}$) becomes $347_{\frac{1}{10}} \times 0.1^2_{\frac{1}{10}}$ as shown in figure 9. In base $\frac{1}{b}$, scientific notation allows for never needing the decimal point in a number.

$$34.7_{\frac{1}{10}} = 347_{\frac{1}{10}} \times 0.1^2_{\frac{1}{10}}$$

FIGURE 9. Example of Scientific Notation in a base $\frac{1}{10}$ number

Reciprocal bases might have some **Application**. They could be used to represent extremely small number without dealing with decimal points. They can also be used the define/display decimal numbers as whole numbers.

A simple implementation is shown below which shows addition and subtraction in base $\frac{1}{b}$. It should help someone get started in implementing their own reciprical base system.

```
#include <iostream>
#include <string>

using namespace std;

#define PRECISION 100

struct RecipricalB
{
    int base;
    int precision;
    int * buffer;
};

string ToString(RecipricalB n);

RecipricalB Create(int base, string str, short precision = PRECISION)
{
    RecipricalB n;
    n.base = base;
    n.precision = precision;
    n.buffer = new int[precision];

    for(int i = str.length() - 1; i >= 0; i--)
    {
        n.buffer[str.length() - i - 1] = str[i] - '0';
    }

    return n;
```

```
}

bool Add(RecipricalB n1, RecipricalB n2, RecipricalB &n3)
{
    if (n1.base != n2.base) { cout << "numbers don't have same base"; return false; }
    if (n1.precision != n2.precision) { cout << "numbers don't have same precision";
        return false; }

    n3 = Create(n1.base, "");

    for (int i = n1.precision; i >= 0; i--)
    {
        n3.buffer[i] = n1.buffer[i] + n2.buffer[i];
    }

    for (int i = n1.precision; i >= 0; i--)
    {
        if (n3.buffer[i] >= n3.base)
        {
            n3.buffer[i] -= n3.base;
            n3.buffer[i-1]++;
        }
    }

    return true;
}

bool Subtract(RecipricalB n1, RecipricalB n2, RecipricalB &n3)
{
    if (n1.base != n2.base) { cout << "numbers don't have same base"; return false; }
    if (n1.precision != n2.precision) { cout << "numbers don't have same precision";
        return false; }

    n3 = Create(n1.base, "");

    for (int i = n1.precision; i >= 0; i--)
    {
        n3.buffer[i] = n1.buffer[i] - n2.buffer[i];
    }

    for (int i = n1.precision; i >= 0; i--)
    {
        if (n3.buffer[i] < 0)
        {
            n3.buffer[i-1]--;
            n3.buffer[i] += n3.base;
```

```
        }
    }

    return true;
}

string ToString(RecipricalB n)
{
    string str = "";
    bool start = false;
    for (int i = n.precision - 1; i >= 0; i--)
    {
        if (start == false)
        {
            if (n.buffer[i] != 0)
            {
                str += to_string(n.buffer[i]);
                start = true;
            }
        }
        else
        {
            str += to_string(n.buffer[i]);
        }
    }
    return str;
}

int main()
{
    RecipricalB n1 = Create(10,"12");
    RecipricalB n2 = Create(10,"10");
    RecipricalB n3;
    Add(n1, n2, n3);
    cout << ToString(n1) << " + " << ToString(n2) << " = " << ToString(n3) << endl;


    return 1;
}
```