Concepts of Conversion of Base 10 Number to Base 2 Fixed Point Register Binary Number Representation

Sean Rodby, Luke Snyder, Autar Kaw

University of South Florida

United States of America

kaw@eng.usf.edu

Website: http://numericalmethods.eng.usf.edu

Version: Maple 12; Last Revised: August 28, 2008

Introduction

The following worksheet illustrates how to convert a positive decimal number to a fixed point register binary number using loops and various conditional statements. The user inputs a decimal number in the *Input* section of the program, and then an equivalent binary number is given as an output.

Initialization

```
> restart : with(ArrayTools) :
```

Section 1: Input

This is the only section where the user interacts with the program.

Enter positive real number to be converted to binary number

```
> decnum := 54.3
```

$$decnum := 54.3 \quad \textbf{(3.1)}$$

Enter number of bits to be used in *integer part*

```
> m := 8
```

$$m := 8 \quad \textbf{(3.2)}$$

Enter number of bits to be used in *fractional part*

```
> n := 6
```

$$n := 6 \quad \textbf{(3.3)}$$

This is the end of the user section. All information must be entered before proceeding to the next section. **RE-EXECUTE THE PROGRAM.**

Section 2: Procedure

Calculating the minimum and maximum possible base-10 values based on the number of bits specified by the user.

```
> sum1 := 0 :
```

```
for i from 0 to m − 1 do
sum1 := sum1 + 2^i
end do:
sum2 := 0 :
for i to n do
sum2 := sum2 + 2^(-i)
end do:
maxval := evalf(sum1 + sum2) :
minval := evalf(2^(-n)) :
printf("\nThe magnitude of the minimum number that can be represented in floating point
    representation given the number of bits is minval = %g.", minval);
    printf("\nThe magnitude of the maximum number that can be represented in floating
    point representation given the number of bits is maxval = %g.", maxval);
```

```
The magnitude of the minimum number that can be represented in floating
point representation given the number of bits is minval = 0.015625.
The magnitude of the maximum number that can be represented in floating
point representation given the number of bits is maxval = 255.984.
```

If the maximum value that can be produced given the user specified number of bits is smaller than the number to be represented, then more bits are needed. Similarly, if the minimum value that can be produced given the user specified number of bits is larger than the number to be represented, a change in bits is needed.

```
> if maxval < decnum then
printf("\nSince |%f| > %f, the number of bits specified is not sufficient,dec_num,maxval to
    represent this number in floating point representation.", decnum, maxval);
printf("Either specify fewer mantissa bits, or more total bits for the worksheet.");
elif minval > decnum then
printf("Since %g > |%g|, the number of bits specified is not sufficient to represent this
    number in floating point binary format. Either specify fewer mantissa bits, or more total
    bits for the worksheet.", minval, decnum);
else
    printf("\nSince %g < |%g| < %g the base−10 number can be represented in floating
    point binary format.", minval, decnum, maxval);
end if;
```

```
Since 0.015625 < |54.3| < 255.984 the base-10 number can be represented in
floating point binary format.
```

Using the *floor* command to isolate each section of the decimal number to begin binary number conversion

```
> before_decimal := floor(decnum);
after_decimal := decnum − floor(decnum);
```

$$before\_decimal := 54$$

$$after\_decimal := 0.3 \qquad \textbf{(4.1)}$$

Using a loop to perform the binary conversion repetitively until the necessary number of bits are used.
Using the *floor* command to calculate the quotient of the decimal number.
Using a combination of the *ceil* and *floor* commands to determine the remainder, and inputting this value into an array.
Then calculating the binary conversion of the value before the decimal. Here we use the variable

Bold to denote the value of the integer part of the decimal number.

```
> Bold := before_decimal :

  for i to m do
  Quotient := floor(Bold/2);
  Binary[i] := ceil(Bold/2) - floor(Bold/2);

  if Quotient = 0 and before_decimal ≠ 0 then
  Binary[i] := 1;
  break
  elif before_decimal = 0 then
  Binary[i] := 0;
  break
  end if;
  Bold := Quotient;
  end do:
```

Using this method, the placement of the binary numbers in the array *Binary* are backwards and so must be flipped around to form the actual binary number.

```
> lenBin := convert(Binary, list) :
  t := Size(lenBin, 2) :

  for j to t do
  Binary_Before_Dec[j] := Binary[t + 1 - j];
  end do:
```

Converting the *fractional part* of the decimal into a binary number. Here we initialize the sum using a new value, *BoldA*, which represents the value after the decimal point.

```
> BoldA := after_decimal :
  for i from 1 to n do
  Num := BoldA·2;
  if Num < 1 then
  Binary_After_Dec[i] := 0;
  elif Num ≥ 1 then
  Num := Num - floor(Num);
  Binary_After_Dec[i] := 1;
  end if;
  BoldA := Num;
  end do:
```

Combining the binary numbers and determining the number of terms in the resulting base 2 number. Then displaying the results.

```
> BBDconv := convert(Binary_Before_Dec, list) :
  BBDLen := Size(BBDconv, 2) :
  BADconv := convert(Binary_After_Dec, list) :
  BADLen := Size(BADconv, 2) :
  str1 := "" :
  for i to BBDLen do
  str1 := cat(str1, Binary_Before_Dec[i])
  end do:

  str2 := "" :
  for i to BADLen do
```

```
    str2 := cat(str2, Binary_After_Dec[i])
    end do:

    Final_Bin_Num := cat(str1, ".", str2);
```

$$Final\_Bin\_Num := "110110.010011" \quad \textbf{(4.2)}$$

Using a loop to sum values of the *integer* portion of the base-2 number. The loop variable *sumint* is used for summation and is initialized at 0.

```
> int_bin := floor(parse(Final_Bin_Num)) :
  str_int := convert(int_bin, string) :
  m := length(str_int) :
  n := length(Final_Bin_Num) :

  sumint := 0 :
  for i to m do
  bin_int[i] := parse(str_int_i);
  sumint := sumint + bin_int[i] 2^(m - i);
  end do:
```

Using a loop to sum values of the *fractional* portion of the base-2 number. The loop variable *sumfrac* is used for summation and is initialized at 0.

```
> sumfrac := 0 :
  j := 1 :
  for i from m + 2 to n do
  bin_frac[j] := parse(Final_Bin_Num[i]);
  sumfrac := sumfrac + bin_frac[j] * 2^(-j);
  j := j + 1;
  end do:
```

Adding the *fractional* portion of the base-2 number with the *integer* portion which yields the base-10 number.

```
> total_dec := evalf(sumint + sumfrac);
```

$$total\_dec := 54.29687500 \quad \textbf{(4.3)}$$

Finding the relative percentage difference between the representation and the number to be represented.

```
> tot_perc_diff := ((decnum - total_dec) / decnum) * 100;
```

$$tot\_perc\_diff := 0.005755064457 \quad \textbf{(4.4)}$$

Conclusion

This worksheet illustrates the use of Maple to convert a base-10 number to a base-2 binary number. It is important to understand the binary system as it has numerous applications. Critical to this understanding is being able to convert binary numbers to base-10 numbers, and vice-versa.

References

Binary Representation of Numbers
See: http://numericalmethods.eng.usf.edu/mws/gen/01aae/mws_gen_aae_txt_binaryrepresentation.pdf