De La Salle University - Manila
Term 2 AY. 2023-2024

In partial fulfillment of the course
Introudction to Computer Organization

IEEE-754 Decimal-32 Floating Point Converter

Documentation

Submitted by:
Gonzaga, Eros Aneelouv
Hallar, Francine Marie
Hong, Letty
Lim, Justin Nathaniel
Milan, John Lloyd

Submitted to:
Sir Roger Uy

Introduction

This repository contains a Decimal-32 Floating Point converter implemented in JavaScript. The converter includes functions of converting the input to binary and hexadecimal output and an option to choose a preferred rounding method, such as truncate, round up, round down, and round to nearest ties to even. It also includes a graphical user interface for interacting with the converter, such as a dropdown round method option, convert and clear button, and an export to text file button.

Project Specifications

IEEE-754 Decimal-32 Floating Point Converter including infinity and NaN special cases. It accepts a decimal and base-10 value that can handle more than 7 digits, and a rounding method option. The output handles binary and hexadecimal output with an option to output in a text file.

Implementation

In order to implement the Decimal-32 Floating Point converter, the floating point number first needs to be parsed and converted into values that an algorithm can work with. The goal of this process is to turn the two inputs (a fixed point base **b** and an exponent **e**, in the format **b** * 10^**e**) into a sign bit, 7 normalized digits (from **b**) and the corresponding normalized exponent. Shown below is a rough outline of the conversion process:

Given inputs **b**, a string, and **e**, an integer,

1. Check if **b** is “nan”. If it is, display the decimal-32 value for NaN and exit. (This special case has to be checked now since the next line checks to see if **b** follows the format for a floating point number)
2. Check to make sure the input **b** is in the correct fixed point format. Fixed point data can start with a +, -, or no sign. Afterwards, any number of digits can be present (including zero, as .23 is still considered a valid floating point number for conversion), followed by an optional decimal point and a required sequence of digits of length 1 or more. This is accomplished using a regular expression.
3. Trim leading and trailing whitespace from **b**.
4. If **b** starts with -, set the sign bit to 1. Otherwise, set it to 0. Remove all instances of + or - from the start of **b**.
5. Remove the decimal point from **b**. This operation is effectively the same as moving the decimal point to the right of **b**, increasing the value of **b**. To maintain equality, decrease **e** by the number of digits to the right of the decimal point.
6. Remove all leading and trailing zeros from **b**. Removing trailing zeros decreases **b**, so increase **e** by the number of trailing zeros removed.
7. If **b** is less than 7 digits, pad the value with zeros from the left so it is seven digits, then end the routine by returning the sign bit, **b** as the normalized digits**,** and **e** as the exponent.

8. Otherwise, let **whole** be the first 7 digits of **b**, and **frac** be the remaining digits. Prepend the digits **frac** with "0.", so that the digits represent a fraction between 0 and 1. Convert **frac** to a float and **whole** to an integer, then adjust the value of **whole** based on the rounding mode.
    a. Truncate: Do nothing.
    b. Round down: Add 1 to **whole** if the sign bit is 1.
    c. Round up: Add 1 to **whole** if the sign bit is 0.
    d. Nearest ties-to-even: If **frac** is more than 0.5, or **frac** is 0.5 and **whole** is odd, add 1 to **whole**.
9. Remove all trailing zeros from **whole**. Add the number of zeros removed to **e**.
10. End the routine by returning the sign bit, **whole** as the normalized digits, and **e** as the exponent.

Let **digits** be the value of the 7 normalized digits returned by this routine.
To convert the values to decimal-32, the remaining special cases were handled. If **e** is over 90, a check was made to see if shifting **digits** to the left would allow **e** to equal 90 or less. For example, consider the case where **digits** = 0000001 and **e** = 91. This would not normally be representable in decimal-32, but by adjusting **digits** to be 0000010 and **e** to 90, it becomes possible to represent this number. These cases are handled in this manner, if possible.

If **e** is still too large, the value for infinity is returned.

If **e** is instead less than -101, return the decimal-32 value for 0.0 instead.

From here, we can assume that **digits** and **e** represent a normal case. Each part of the decimal-32 output is computed individually. First, to assist in the calculation, **e'** is set as equal to **e** + 101. Then, if the first digit of **digits** is 8 or 9, the combination field is set as "11" + the first two bits of **e'** + the last bit of the first digit of **digits**. Otherwise, it is set as the first two bits of **e'** + the last three bits of the first digit of **digits**.

The exponent continuation field is assigned to the remaining bits of **e'**. The last six **digits** are split into two groups of three and converted into Densely Packed Binary Coded Decimal through the following routine:

1. Copy the last bit of each digit to the third, sixth, and last bits of the answer, respectively.
2. If none of the digits are greater than 7, bit 7 is set to 0. Otherwise, set it to 1.
3. Set the remaining bits as follows:
    a. Let **bc** be the second and third bits of the first digit.
    b. Let **fg** be the second and third bits of the second digit.
    c. Let **jk** be the second and third bits of the third digit.

d. Assign the remaining bits according to this table:

Is this digit 8 or 9?			Assign the following bits:		
First digit	Second Digit	Third Digit	Bits 1 and 2	Bits 4 and 5	Bits 8 and 9
No	No	No	bc	fg	jk
No	No	Yes	bc	fg	00
No	Yes	No	bc	jk	01
No	Yes	Yes	bc	10	11
Yes	No	No	jk	fg	10
Yes	No	Yes	fg	01	11
Yes	Yes	No	bc	00	11
Yes	Yes	Yes	00	11	11

The converted digits are then placed in the coefficient continuation field. Now that all the parts of the converted decimal-32 number are complete, the website is able to show the results of the calculation on the site.

Development Process and Considerations

Some technical limitations were encountered during the development process that influenced the design of the final conversion algorithm. One large issue encountered was the conversion process for the floating point input, since built-in algorithms for parsing strings representing floating point numbers converted them into IEEE 754 binary floats. As a result, parsing the user's input had to be done manually.

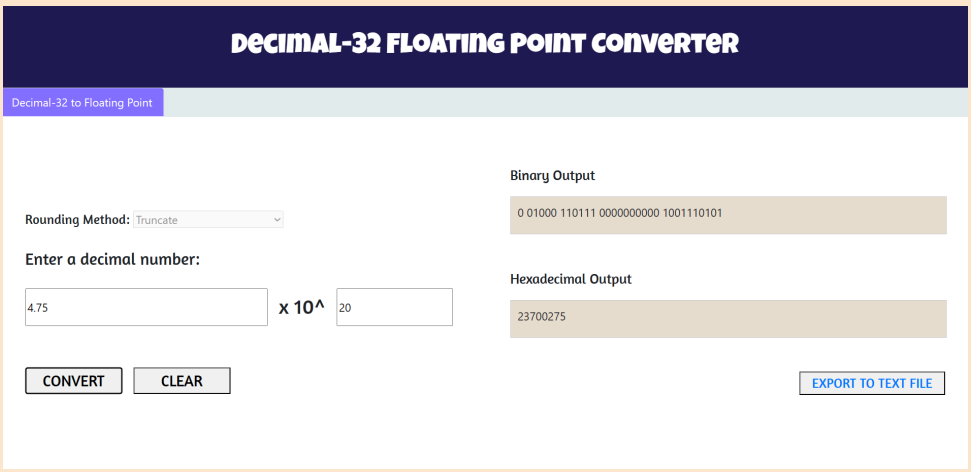
Another issue to be addressed was the enforcement of only having leading zeros for the value of **digits**. While values like **7 * 10^4** and **700 * 10^2** are the same value, converting these directly into decimal-32 results in different answers. To enforce a single canonical conversion value, the version of **digits** with no trailing zeros was preferred (if possible). Due to the potential of trailing zeros occurring both before and after rounding, they had to be removed multiple times in the input parsing procedure.

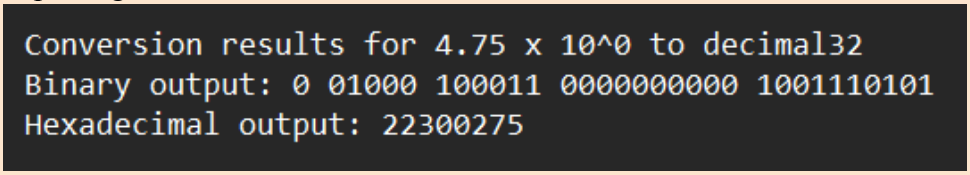
In some cases, it is impossible to use the version of **digits** with no leading zeros, such as in the case where **e** is over the maximum possible value representable by decimal-32. **7 * 10^92** has a value of **e** that is too high to be represented, so **700 * 10^90** must be used instead. The representation with the minimum number of trailing zeros is used.

Test Cases

<table>
<tr><th>Case Description</th><th>Input</th><th>Output</th><th>Expected Output</th><th>Pass/Fail</th></tr>
<tr><td rowspan="3">Normal positive
less than 7</td><td>4.75 x 10^0</td><td>Binary Output:
0
01000
100011
0000000000
1001110101

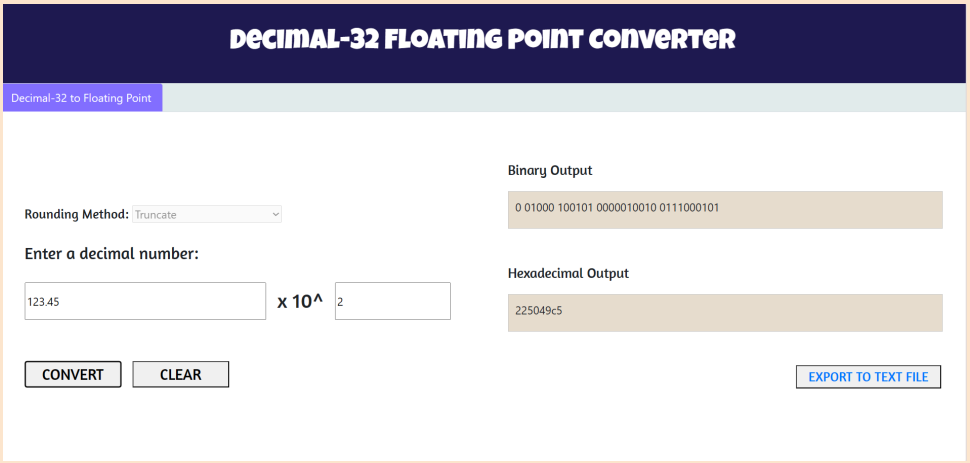
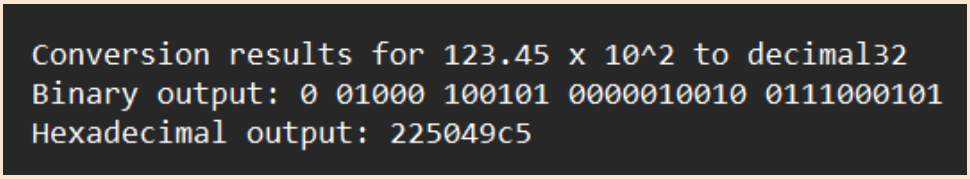
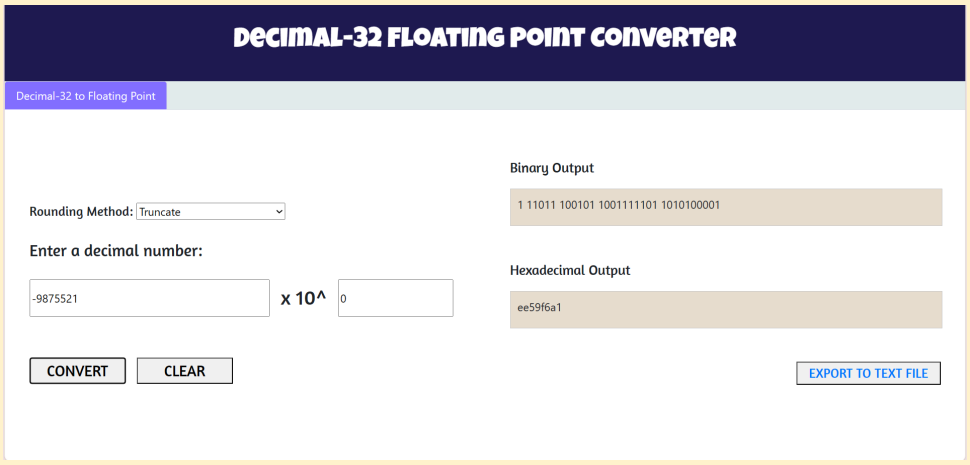
Hexadecimal Output:
22300275</td><td>Binary Output:
0
01000
100011
0000000000
1001110101

Hexadecimal Output:
22300275</td><td>PASS</td></tr>
<tr><td colspan="4"> 

Exporting to text file:

</td></tr>
<tr><td>123.45 x 10^2</td><td>Binary Output:
0
01000
100101
0000010010
0111000101

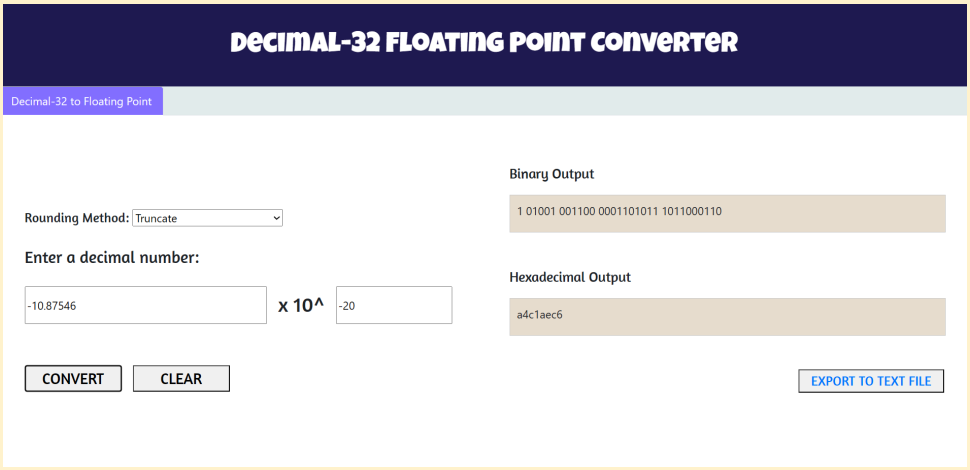
Hexadecimal Output:</td><td>Binary Output:
0
01000
100101
0000010010
0111000101

Hexadecimal Output:</td><td>PASS</td></tr>
</table>

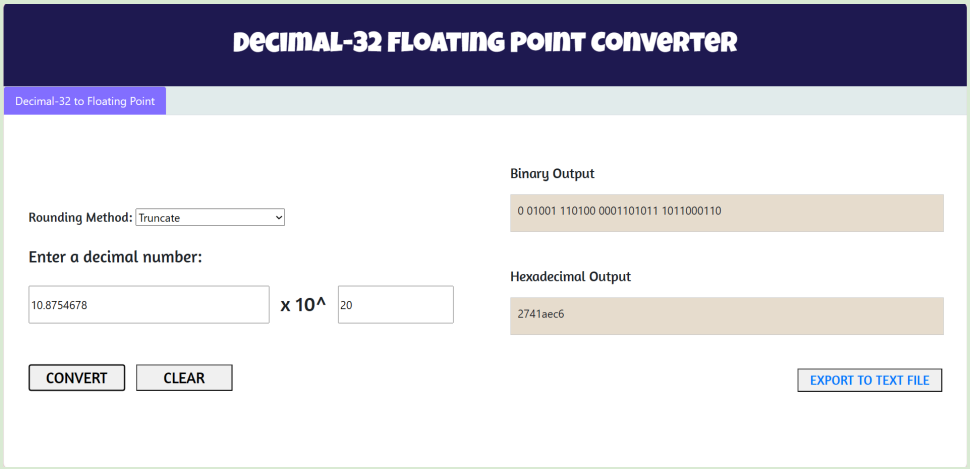
		225049C5	225049c5	
	 Exporting to text file: 		
Normal negative less than 7	-9875521 x 10^0	Binary Output: 1 11011 100101 1001111101 1010100001 Hexadecimal Output: EE59F6A1	Binary Output: 1 11011 100101 1001111101 1010100001 Hexadecimal Output: ee9f6a1	PASS
				

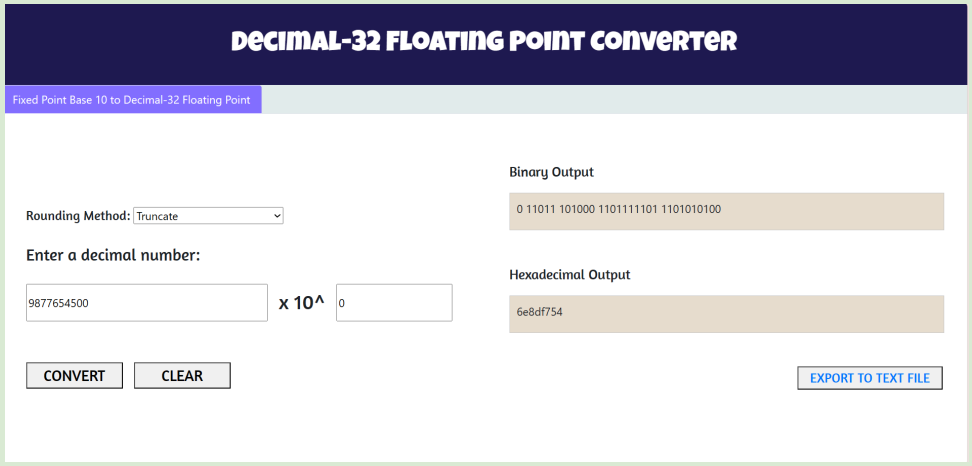
<table>
<tr><td></td><td colspan="4">Exporting to text file:
<pre>Conversion results for -9875521 x 10^0 to decimal32
Binary output: 1 11011 100101 1001111101 1010100001
Hexadecimal output: ee59f6a1</pre></td></tr>
<tr><td></td><td>-10.87546 x 10^ -20</td><td>Binary Output:
1
01001
001100
0001101011
1011000110

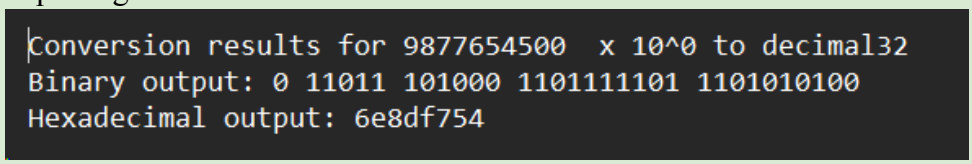
Hexadecimal Output:
A4C1AEC6</td><td>Binary Output:
1
01001
001100
0001101011
1011000110

Hexadecimal Output:
a4c1aec6</td><td>PASS</td></tr>
<tr><td></td><td colspan="4"> 

Exporting to text file:
<pre>Conversion results for -10.87546 x 10^-20 to decimal32
Binary output: 1 01001 001100 0001101011 1011000110
Hexadecimal output: a4c1aec6</pre></td></tr>
<tr><td>Normal positive more than 7 round by truncation</td><td>10.8754678 x 10^20</td><td>Binary Output:
0
01001
110110
0001101011
1011000111</td><td>Binary Output:
0
01001
110110
0001101011
1011000111</td><td>PASS</td></tr>
</table>

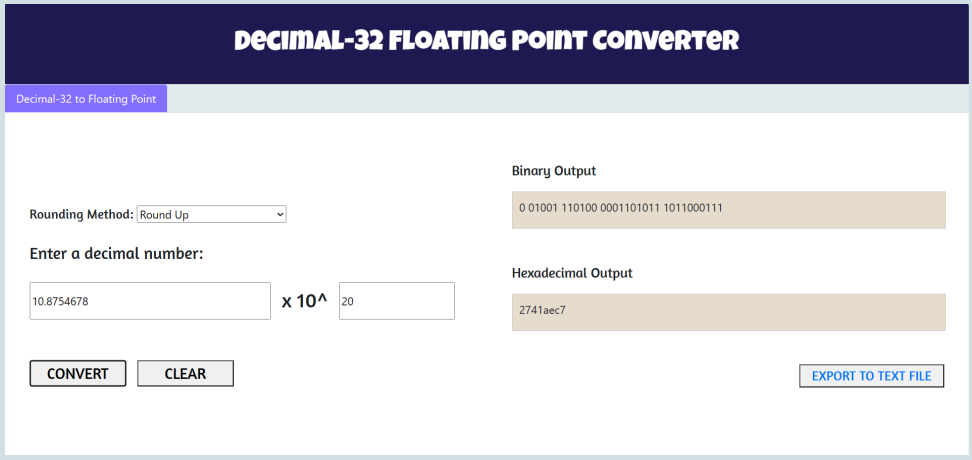
		Hexadecimal Output: 2741AEC6	Hexadecimal Output: 2741aec6	
	 Exporting to text file: Conversion results for 10.8754678 x 10^20 to decimal32 Binary output: 0 01001 110100 0001101011 1011000110 Hexadecimal output: 2741aec6		
	9877654500 x 10^0	Binary Output: 0 11011 111111 1101111101 1101010100 Hexadecimal Output: 6E8DF754	Binary Output: 0 11011 111111 1101111101 1101010100 Hexadecimal Output: 6e8df754	PASS

<table>
<tr>
<td></td>
<td colspan="4">


Exporting to text file:


</td>
</tr>
<tr>
<td rowspan="2">Normal positive more than 7 round up</td>
<td>10.8754678 x 10^20</td>
<td>Binary Output:
0
01001
110100
0001101011
1011000111

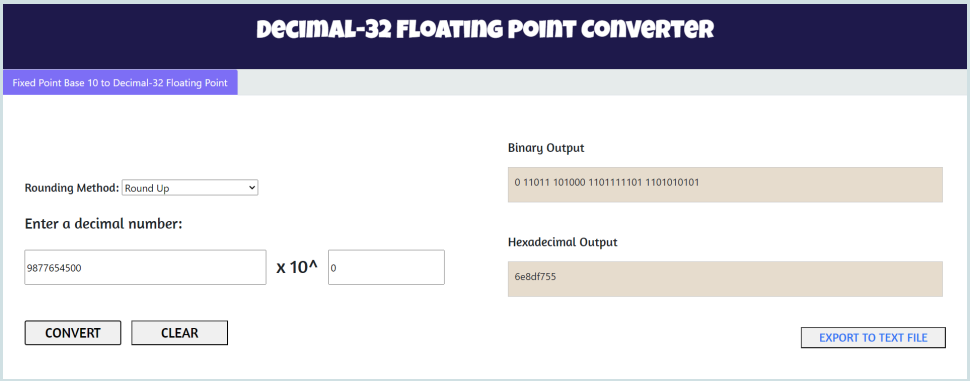
Hexadecimal Output:
2741AEC7</td>
<td>Binary Output:
0
01001
110100
0001101011
1011000111

Hexadecimal Output:
2741aec7</td>
<td>PASS</td>
</tr>
<tr>
<td colspan="4">


</td>
</tr>
</table>

<table>
<tr><td></td><td colspan="4">Exporting to text file:
<pre>Conversion results for 10.8754678 x 10^20 to decimal32
Binary output: 0 01001 110100 0001101011 1011000111
Hexadecimal output: 2741aec7</pre></td></tr>
<tr><td></td><td>9877654500 x 10^0</td><td>Binary Output:
0
11011
111111
1101111101
1101010101

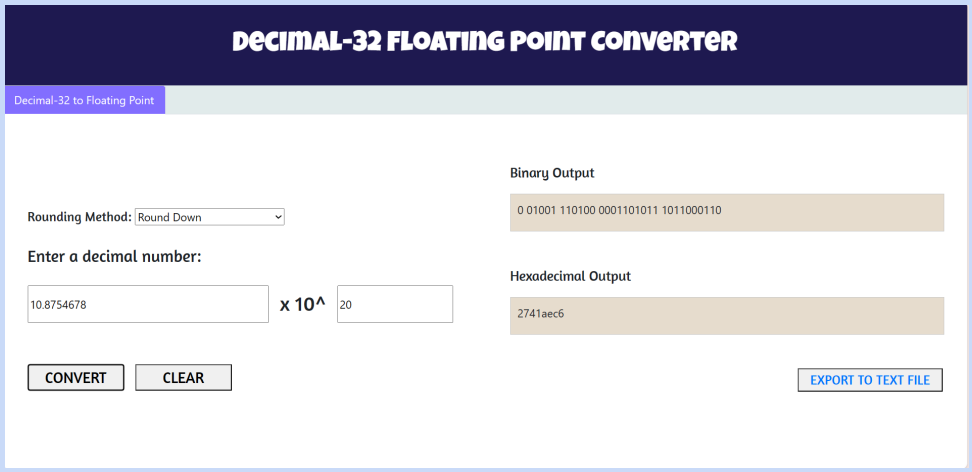
Hexadecimal Output:
6E8DF755</td><td>Binary Output:
0
11011
111111
1101111101
1101010101

Hexadecimal Output:
6e8df755</td><td>PASS</td></tr>
<tr><td></td><td colspan="4"> 

Exporting to text file:
<pre>Conversion results for 9877654500 x 10^0 to decimal32
Binary output: 0 11011 101000 1101111101 1101010101
Hexadecimal output: 6e8df755</pre></td></tr>
<tr><td>Normal positive more than 7 round down</td><td>10.8754678 x 10^20</td><td>Binary Output:
0
01001
110110
0001101011
1011000111

Hexadecimal Output:
2761AEC6</td><td>Binary Output:
0
01001
110110
0001101011
1011000111

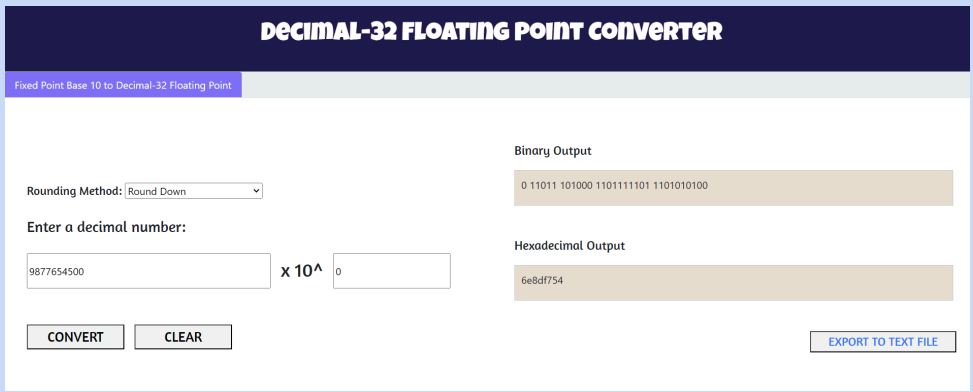
Hexadecimal Output:
2761aec6</td><td>PASS</td></tr>
</table>



Exporting to text file:

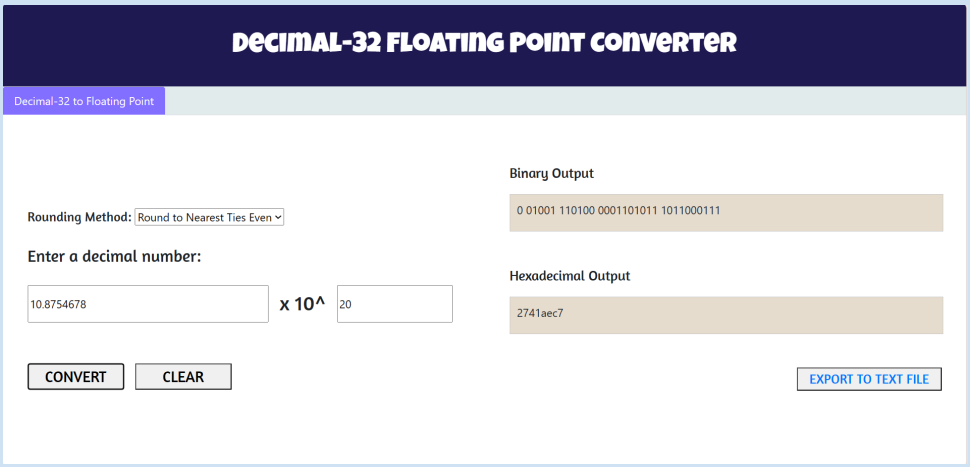
```
Conversion results for 10.8754678 x 10^20 to decimal32
Binary output: 0 01001 110100 0001101011 1011000110
Hexadecimal output: 2741aec6
```

9877654500 x 10^0	Binary Output: 0 11011 111111 1101111101 1101010100 Hexadecimal Output: 6E8DF754	Binary Output: 0 11011 111111 1101111101 1101010100 Hexadecimal Output: 6e8df754	PASS



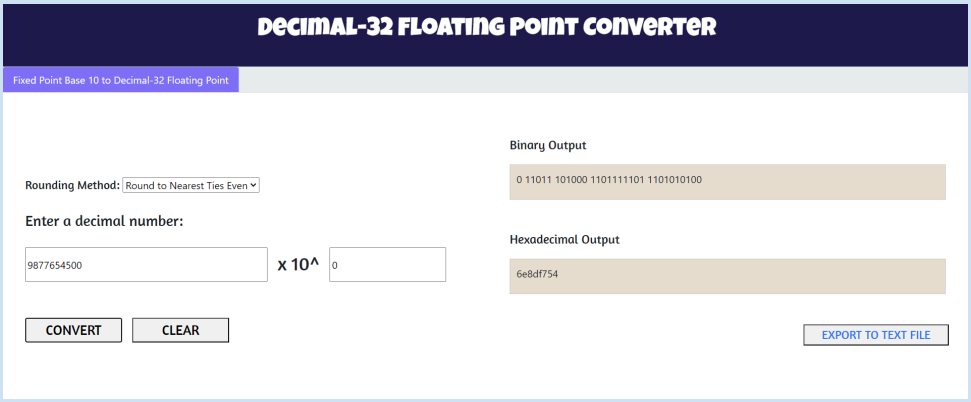
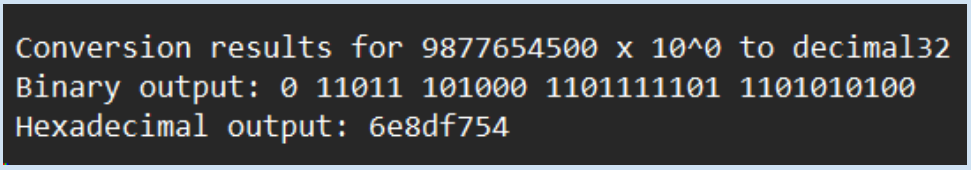
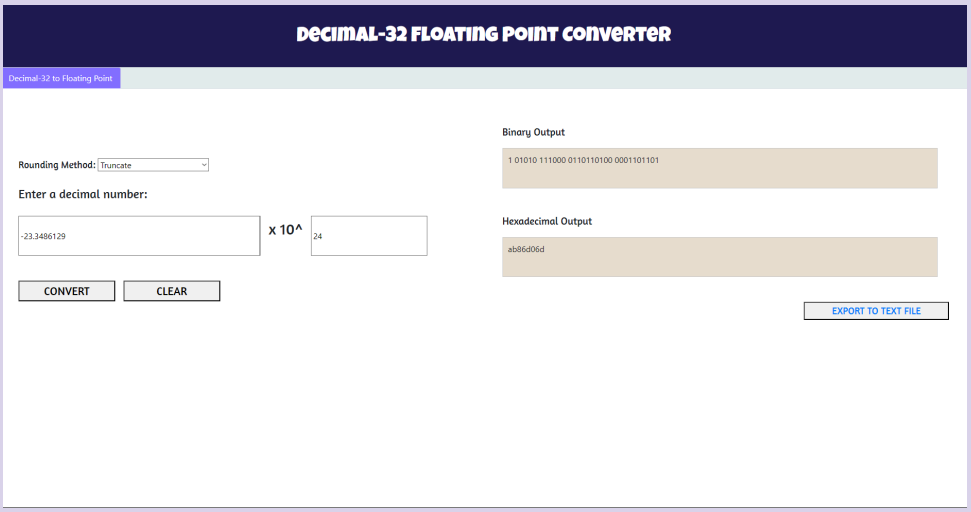
<table>
<tr><td></td><td colspan="4">Exporting to text file:
<pre>Conversion results for 9877654500 x 10^0 to decimal32
Binary output: 0 11011 101000 1101111101 1101010100
Hexadecimal output: 6e8df754</pre></td></tr>
<tr><td rowspan="4">Normal positive more than 7 round to nearest ties even</td><td>10.8754678 x 10^20</td><td>Binary Output:
0
01001
110100
0001101011
1011000111

Hexadecimal Output:
2741AEC7</td><td>Binary Output:
0
01001
110100
0001101011
1011000111

Hexadecimal Output:
2741aec7</td><td>PASS</td></tr>
<tr><td colspan="4"> 
</td></tr>
<tr><td colspan="4">Exporting to text file:
<pre>Conversion results for 10.8754678 x 10^20 to decimal32
Binary output: 0 01001 110100 0001101011 1011000111
Hexadecimal output: 2741aec7</pre></td></tr>
<tr><td>9877654500 x 10^0</td><td>Binary Output:
0
11011
111111
1101111101
1101010100

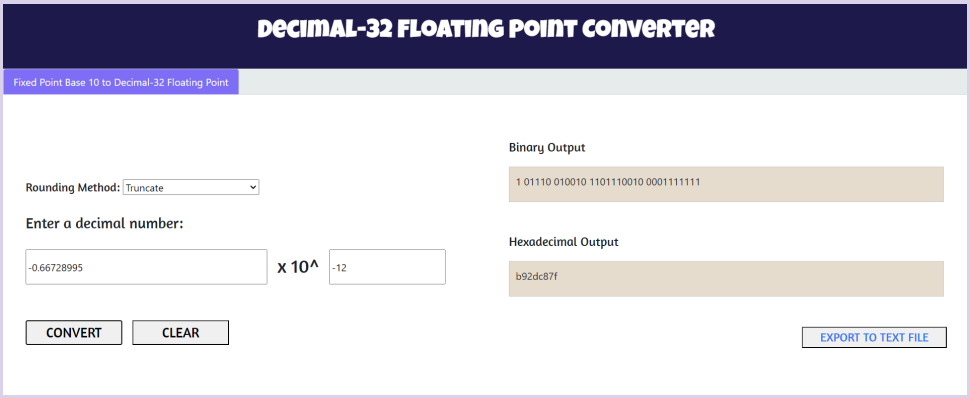
Hexadecimal Output:</td><td>Binary Output:
0
11011
111111
1101111101
1101010100

Hexadecimal Output:</td><td>PASS</td></tr>
</table>

		6FFDF754	6ffdf754	
	 Exporting to text file: 		
Normal negative more than 7 **round by truncation**	-23.3486129 x 10^24	Binary Output: 1 01010 111000 0110110100 0001101101 Hexadecimal Output: AB86D06D	Binary Output: 1 01010 111000 0110110100 0001101101 Hexadecimal Output: ab86d06d	PASS
				

<table>
<tr><td></td><td colspan="4">Exporting to text file:
<pre>Conversion results for -23.3486129 x 10^24 to decimal32
Binary output: 1 01010 111000 0110110100 0001101101
Hexadecimal output: ab86d06d</pre></td></tr>
<tr><td></td><td>-0.66728995 x 10^-12</td><td>Binary Output:
1
01110
010010
1101110010
0001111111

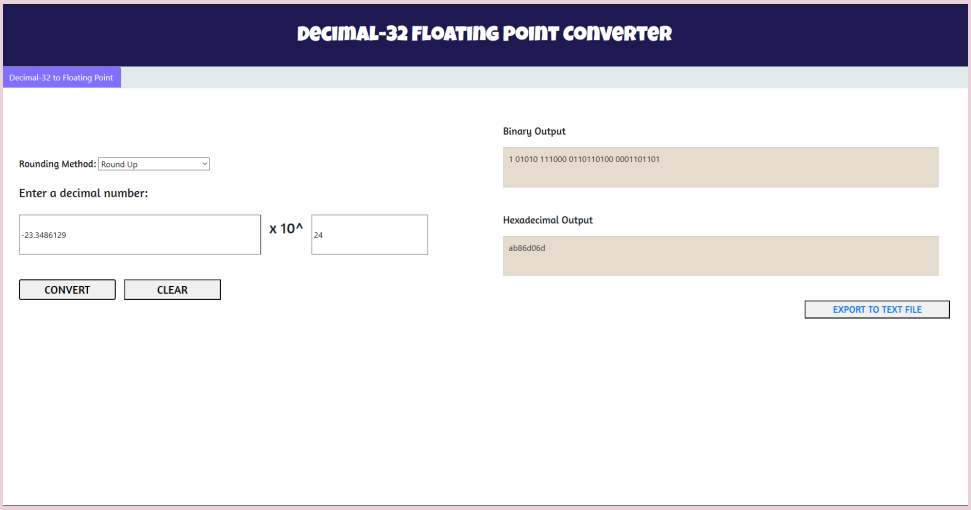
Hexadecimal Output:
B92DC87F</td><td>Binary Output:
1
01110
010010
1101110010
0001111111

Hexadecimal Output:
b92dc87f</td><td>PASS</td></tr>
<tr><td></td><td colspan="4"> 

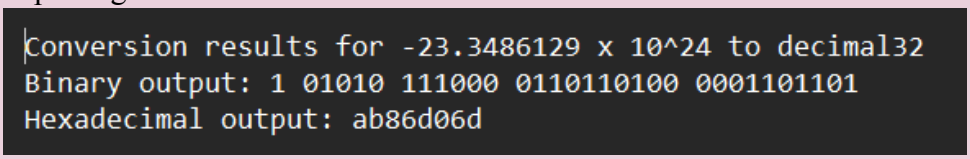
Exporting to text file:
<pre>Conversion results for -0.66728995 x 10^-12 to decimal32
Binary output: 1 01110 010010 1101110010 0001111111
Hexadecimal output: b92dc87f</pre></td></tr>
<tr><td>Normal negative more than 7 round up</td><td>-23.3486129 x 10^24</td><td>Binary Output:
1
01010
111000
0110110100
0001101101

Hexadecimal Output:
AB86D06D</td><td>Binary Output:
1
01010
111000
0110110100
0001101101

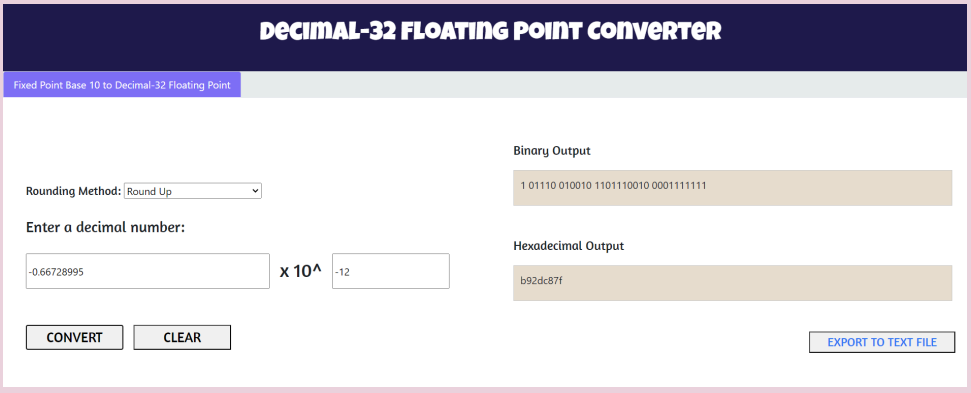
Hexadecimal Output:
ab86d06d</td><td>PASS</td></tr>
</table>



Exporting to text file:

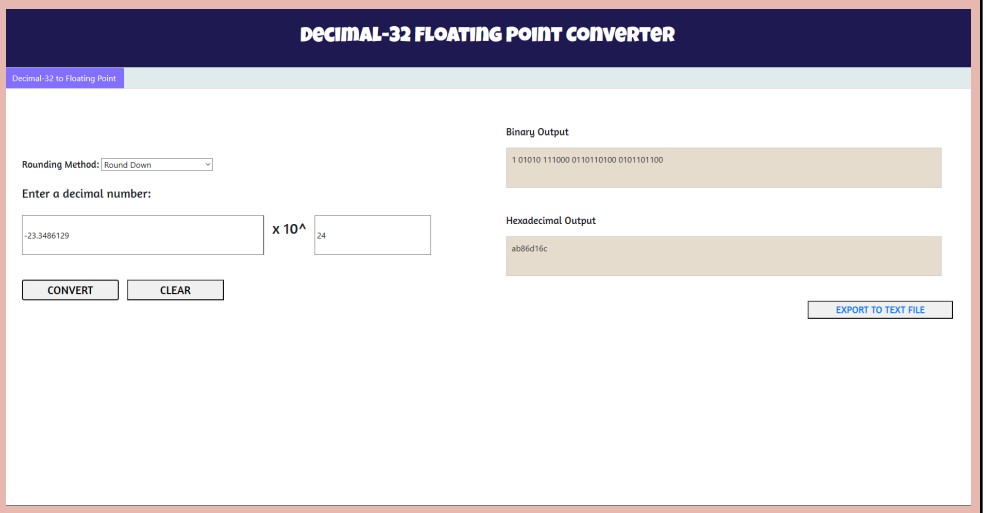


-0.66728995 x 10^-12	Binary Output: 1 01110 010010 1101110010 0001111111 Hexadecimal Output: B92DC87F	Binary Output: 1 01110 010010 1101110010 0001111111 Hexadecimal Output: b92dc87f	PASS

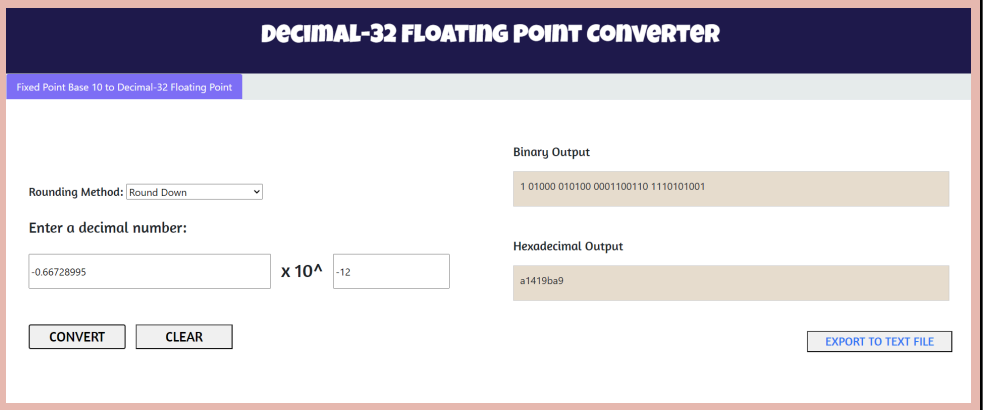
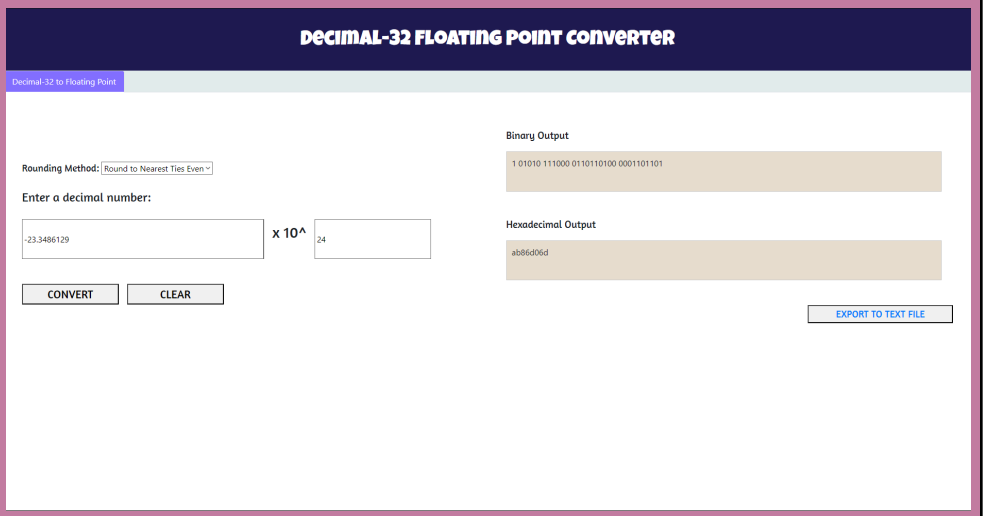


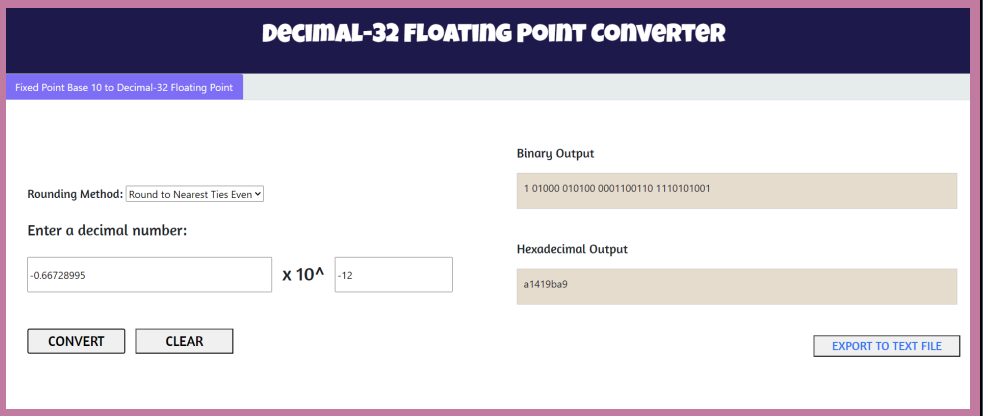
<table>
<tr><td></td><td colspan="4">Exporting to text file:
<pre>Conversion results for -0.66728995 x 10^-12 to decimal32
Binary output: 1 01110 010010 1101110010 0001111111
Hexadecimal output: b92dc87f</pre></td></tr>
<tr><td rowspan="3">Normal negative more than 7 round down</td><td>-23.3486129 x 10^24</td><td>Binary Output:
1
01010
111000
0110110100
0101101100

Hexadecimal Output:
AB86D16C</td><td>Binary Output:
1
01010
111000
0110110100
0101101100

Hexadecimal Output:
ab86d16c</td><td>PASS</td></tr>
<tr><td colspan="4"> 

Exporting to text file:
<pre>Conversion results for -23.3486129 x 10^24 to decimal32
Binary output: 1 01010 111000 0110110100 0101101100
Hexadecimal output: ab86d16c</pre></td></tr>
<tr><td>-0.66728995 x 10^-12</td><td>Binary Output:
1
01000
010100
0001100110
1110101001
Hexadecimal Output:</td><td>Binary Output:
1
01000
010100
0001100110
1110101001
Hexadecimal Output:</td><td>PASS</td></tr>
</table>

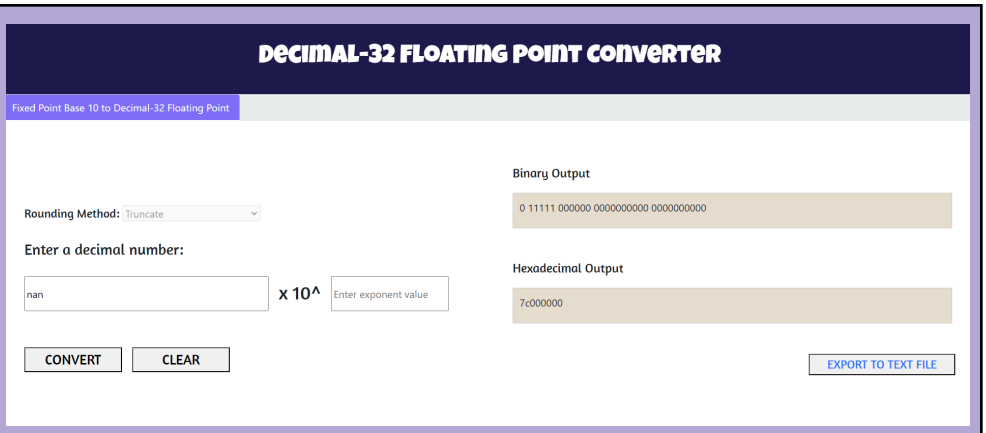
		A1419BA9	a1419ba9	
	 Exporting to text file: Conversion results for -0.66728995 x 10^-12 to decimal32 Binary output: 1 01000 010100 0001100110 1110101001 Hexadecimal output: a1419ba9		
Normal negative more than 7 **round to nearest ties to even**	-23.3486129 x 10^24	Binary Output: 1 01010 111000 0110110100 0001101101 Hexadecimal Output: AB86D06D	Binary Output: 1 01010 111000 0110110100 0001101101 Hexadecimal Output: ab86d06d	PASS
				

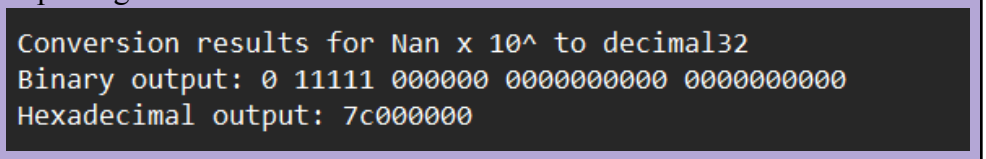
<table>
<tr><td></td><td colspan="4">Exporting to text file:
<pre>Conversion results for -23.3486129 x 10^24 to decimal32
Binary output: 1 01010 111000 0110110100 0001101101
Hexadecimal output: ab86d06d</pre></td></tr>
<tr><td></td><td>-0.66728995 x 10^-12</td><td>Binary Output:
1
01000
010100
0001100110
1110101001
Hexadecimal Output:
A1419BA9</td><td>Binary Output:
1
01000
010100
0001100110
1110101001
Hexadecimal Output:
a1419ba9</td><td>PASS</td></tr>
<tr><td></td><td colspan="4"> 

Exporting to text file:
<pre>Conversion results for -0.66728995 x 10^-12 to decimal32
Binary output: 1 01000 010100 0001100110 1110101001
Hexadecimal output: a1419ba9</pre></td></tr>
<tr><td>NaN</td><td>Nan
NaN
nan
nAn
nAN
NAN</td><td>Binary Output:
0
11111 000000
0000000000
0000000000

Hexadecimal Output:
7C000000</td><td>Binary Output:
0
11111 000000
0000000000
0000000000

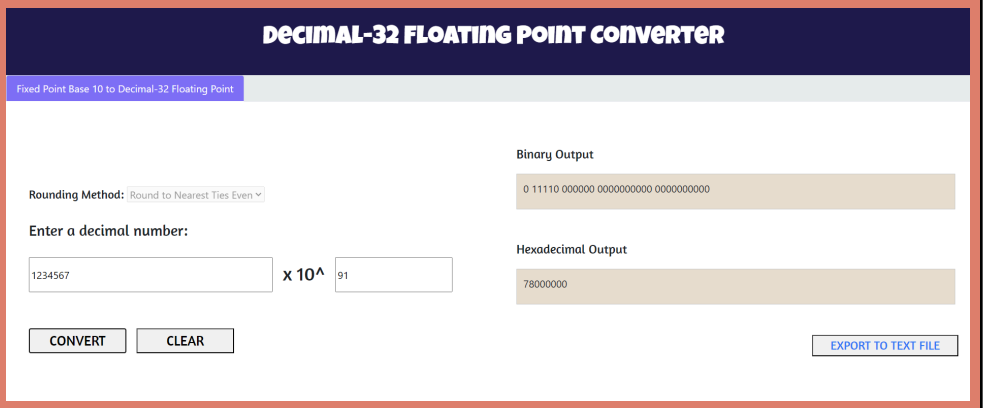
Hexadecimal Output:
7c000000</td><td>PASS</td></tr>
</table>

<table>
<tr>
<td></td>
<td colspan="4">


Exporting to text file:


</td>
</tr>
<tr>
<td>Positive Infinity</td>
<td>1234567 x 10^91</td>
<td>Binary Output:
0
11110
000000
0000000000
0000000000

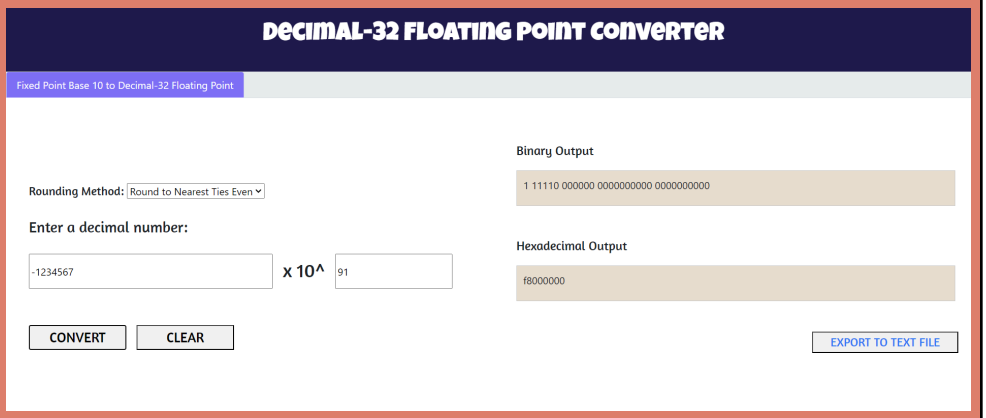
Hexadecimal Output:
78000000</td>
<td>Binary Output:
0
11110
000000
0000000000
0000000000

Hexadecimal Output:
78000000</td>
<td>PASS</td>
</tr>
<tr>
<td></td>
<td colspan="4">


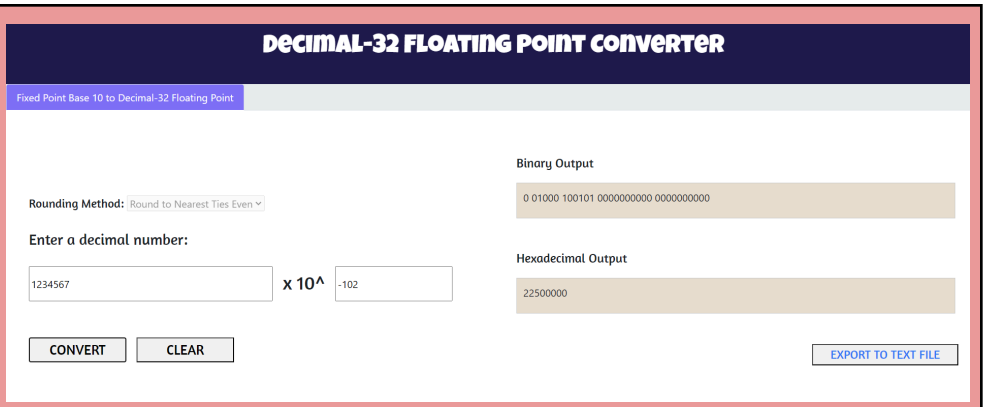
</td>
</tr>
</table>

<table>
<tr><td></td><td colspan="4">Exporting to text file:
<pre>Conversion results for 1234567 x 10^91 to decimal32
Binary output: 0 11110 000000 0000000000 0000000000
Hexadecimal output: 78000000</pre></td></tr>
<tr><td></td><td>-1234567 x 10^111</td><td>Binary Output:
1
11110
000000
0000000000
0000000000

Hexadecimal Output:
F8000000</td><td>Binary Output:
1
11110
000000
0000000000
0000000000

Hexadecimal Output:
f8000000</td><td>PASS</td></tr>
<tr><td></td><td colspan="4"> 

Exporting to text file:
<pre>Conversion results for -1234567 x 10^111 to decimal32
Binary output: 1 11110 000000 0000000000 0000000000
Hexadecimal output: f8000000</pre></td></tr>
<tr><td>Negative Infinity</td><td>1234567 x 10^-102</td><td>Binary Output:
0
01000
100101
0000000000
0000000000
Hexadecimal Output:
2500000</td><td>Binary Output:
0
01000
100101
0000000000
0000000000
Hexadecimal Output:
2500000</td><td>PASS</td></tr>
</table>



Exporting to text file:

```
Conversion results for 1234567 x 10^-102 to decimal32
Binary output: 0 01000 100101 0000000000 0000000000
Hexadecimal output: 22500000
```