

## CSARCH2-Simulation-Project---IEEE-754-Decimal-32-Floating-Point-Converter

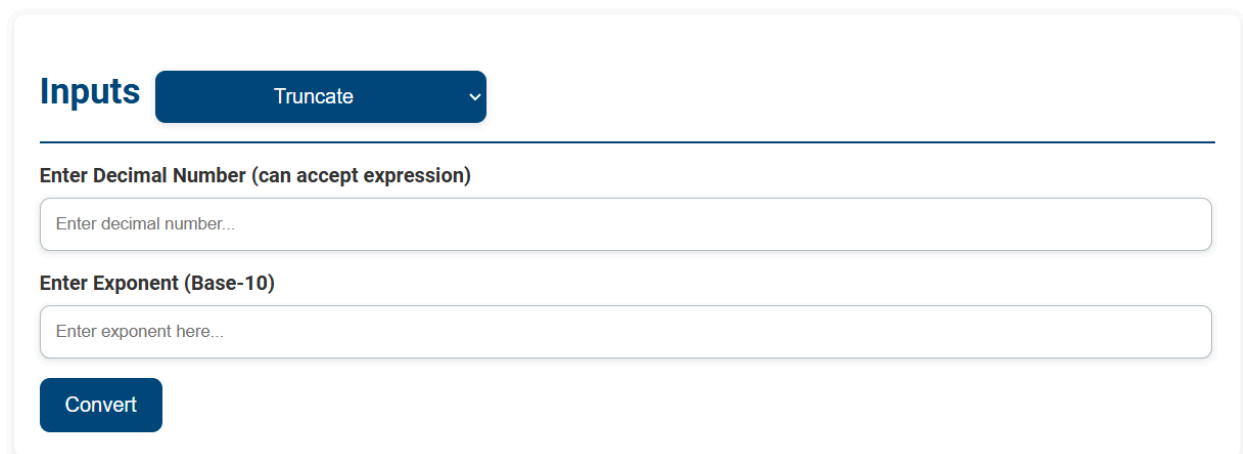
The IEEE-754-Decimal-32-Floating-Point-Converter is a website designed to convert any decimal input with exponent (Base-10) to an IEEE 754 Single Precision decimal (Decimal-32) format. This will also show processing done while converting the input to its respective output. Below is a manual on how to use the application with some sample inputs from the webpage.

### Developers

- Arca, Althea Denisse
- Del Castillo, Jose Mari
- Fulo, Rulet
- Lim, Alyanna

### How to Use

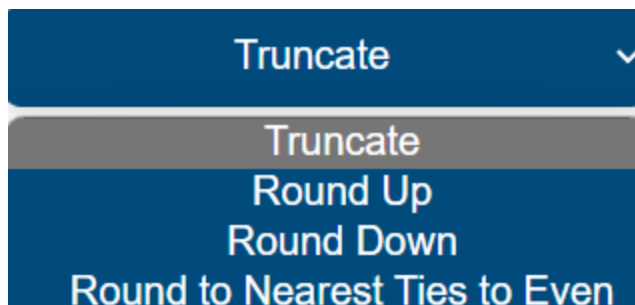
1. Input any decimal number (can accept basic expressions such as  $1/0$ ,  $\sqrt{4}$ ,  $x+10$ ) and the exponent which must be a number only.
2. Click Convert.



The screenshot shows the input interface of the converter. It features a header with the word "Inputs" in blue and a dropdown menu currently set to "Truncate". Below this, there are two input fields: "Enter Decimal Number (can accept expression)" and "Enter Exponent (Base-10)". Each field has a placeholder text "Enter decimal number..." and "Enter exponent here..." respectively. At the bottom left of the form is a blue "Convert" button.

If the decimal number input contains more than 7 digits, the user will have the option to truncate, round up, round down, and round to the nearest ties to even. To do this, do the following:

1. After clicking the convert button, click on the dropdown menu beside the input and choose any options.
2. Click the Convert button again to get the result.



Exporting Results:

1. Click the Export button below the results after converting.
2. The result of the conversion (in a text file named 'IEEE-754 Decimal-32 Output') will automatically download.

## TEST CASES (Select few only - The rest of the test cases will be in README.MD)

### Analysis and Brief Explanations

1. Input with 3-digit number

Input: 457

Exponent: 2

### Processing

Normalized Decimal

0000457

Final Exponent

2

E-Prime

103 (01100111)

### Output

Sign Bit

0

Combination Bits

01000

Exponent Continuation Bits

100111

Mantissa Continuation Bits

000000000100101011

Final Output (Binary)

0 01000 100111 000000000100101011

Final Output (Hexadecimal)

0x22700257

Export

2. Input with negative 5-digit number

Input: -12345

Exponent: -5

### Processing

Normalized Decimal

-0012345

Final Exponent

-5

E-Prime

96 (01100000)

### Output

Sign Bit

1

Combination Bits

01000

Exponent Continuation Bits

100000

Mantissa Continuation Bits

00000100100111000101

Final Output (Binary)

1 01000 100000 0000010010 0111000101

Final Output (Hexadecimal)

0xA20049C5

Export

This basic test case shows that the input decimal and exponent can be read by the program with digits less than or equal to 7 as the sign bits, combination bits, exponent continuation bits, mantissa continuation bits are correct as they parsed properly. The DPD or the mantissa continuation bits are also converted properly. Lastly, the final output binary and hexadecimal are correct as the program is just appending the respective outputs (bits) properly to the final output with spaces in between in the binary output to check if the outputs are correct and lastly they are converted into the hexadecimal in the hexadecimal output. The normalization for the decimal is seamless as it will just append zeros at the start of the normalized decimal field

3. Input with a decimal point with less than 7 digits

Input: 468.23

Exponent: -20

### Processing

Normalized Decimal

0046823

Final Exponent

-22

E-Prime

79 (01001111)

### Output

Sign Bit

0

Combination Bits

01000

Exponent Continuation Bits

001111

Mantissa Continuation Bits

00010001100100101101

Final Output (Binary)

0 01000 001111 0001000110 0100101101

Final Output (Hexadecimal)

0x20F1192D

Export

4. Input with a decimal point with less than 7 digits (negative decimal)

Input: -3012.43

Exponent: 4

### Processing

Normalized Decimal

-0301243

Final Exponent

2

E-Prime

103 (01100111)

### Output

Sign Bit

1

Combination Bits

01000

Exponent Continuation Bits

100111

Mantissa Continuation Bits

01100000010101000011

Final Output (Binary)

1 01000 100111 0110000001 0101000011

Final Output (Hexadecimal)

0xA2760543

Export

This shows the same correct outputs but they include the decimal point.

**Different Rounding Methods [with mixed inputs (e.g. without or without decimal point, negative or positive)] that are only applicable with inputs exceeding 7 digits and brief explanation of the processing:**

- Input with a decimal point with Truncate option

Input: 8934.95409 (Truncate)

Exponent: -30

### Processing

Normalized Decimal

8934954

Final Exponent

-33

E-Prime

68 (01000100)

### Output

Sign Bit

0

Combination Bits

11010

Exponent Continuation Bits

000100

Mantissa Continuation Bits

10101111001011011100

Final Output (Binary)

0 11010 000100 1010111100 1011011100

Final Output (Hexadecimal)

0x684AF2DC

Export

This shows that it was able to properly normalize the decimal to 7 digits by moving the

decimal point 3 places to the left which would subtract the exponent input per place ( $30 - 3$  [where 3 is the amount of decimal places moved]) and it would be added to the final exponent field. Note that value is then truncated as the rounding method is Truncate. E-Prime will be computed by the final exponent added by 101 and converted into an 8-bit binary value. The sign bit will depend on the sign of the decimal input, if negative the sign bit is 1 and if positive the sign bit is 0. For the combination bits, since the value first digit in the normalized decimal is a major number (8 or 9), the combination bits would start with 11 with 01 coming from the first 2 bits of the binary value of the E-Prime and the 0 would be coming from the first digit's 4-bit binary value with the rightmost bit ( $8 = 1000$ ). The exponent continuation bits would be from the last 6 bits of the binary value of the E-Prime. The mantissa continuation bits are the densely packed decimal values of the first 3 digits after the first digit in the normalized decimal, then another densely packed decimal value coming from the last 3 digits of the normalized decimal. With every amount of bits equating to 32 bits. This process will be repeated with other inputs and rounding methods if there are more than 7 digits in the decimal input.

6. Input without decimal point with Round Up option

Input: 4674320131 (Round Up)

Exponent: 9

Processing	Output
Normalized Decimal 4674321	Sign Bit 0
Final Exponent 12	Combination Bits 01100
E-Prime 113 (01110001)	Exponent Continuation Bits 110001
	Mantissa Continuation Bits 11011101000110100001
	Final Output (Binary) 0 01100 110001 1101110100 0110100001
	Final Output (Hexadecimal) 0x331DD1A1
	<a href="#">Export</a>

At this point, the focus will be mostly on the rounding process. As seen in the figure above, the decimal is first normalized to 7 digits since but the movement of decimals will be to the left so it will add the amount of times the decimal point moved to the exponent input which would be placed in the final exponent field ( $9+3$ ). The normalized decimal will be 4674320.131 but there would be a rounding method to be considered which in this case is rounding up so the value of 4674320 will be added by 1 regardless the decimal value which is why the final normalized decimal value in the processing part is

4674321 but for example when the decimal input is negative (e.g -434523.45 equating to -4345234.5 in the normalization steps) since the rounding method is upwards, the value of the decimal will then be truncated as the to round up means a value that is closer to 0 which is why the output is -4345234 for the normalized decimal.

7. Input with decimal point with Round Down option

Input: -3.3222123123 (Round Down)

Exponent: -54

### Processing

Normalized Decimal

-3322213

Final Exponent

-60

E-Prime

41 (00101001)

### Output

Sign Bit

1

Combination Bits

00011

Exponent Continuation Bits

101001

Mantissa Continuation Bits

01101000100100010011

Final Output (Binary)

1 00011 101001 0110100010 0100010011

Final Output (Hexadecimal)

0x8E968913

Export

As seen in the rounding method, since the new decimal value will be -3322212.3123 due to the normalization process to round down means to rounding to value farther from 0 which in this case will be -3322213 as it adds 1 to the value when it is negative while retaining the negative sign. But with a positive value rounding down just means to truncate the value to 7 digits.

8. Input with Round to Nearest Ties to Even option

Input: 7884565.5 (RTNTTE)

Exponent: 23

### Processing

Normalized Decimal

7884566

Final Exponent

23

E-Prime

124 (01111100)

### Output

Sign Bit

0

Combination Bits

01111

Exponent Continuation Bits

111100

Mantissa Continuation Bits

10000011101011100110

Final Output (Binary)

0 01111 111100 1000001110 1011100110

Final Output (Hexadecimal)

0x3FC83AE6

Export

For the round to nearest ties to even rounding method, it would have 3 possible outputs depending on the decimal value from the initial decimal normalization process. For the output above, the decimal input 7884565.5 will be rounded to nearest even number because the fractional portion is equal to 0.5 which in this case is 7884566, this is because of the 7th digit from the 7 digits on the right hand side, if the value is even it is truncated to 7 digits (e.g. 7884560.5 would equate to 7884560 which is the nearest even value when the value is odd the value of the 7 digits would be added by 1 to make sure that it goes to closest even number as seen in the output above. However there are cases when the fractional part of the decimal input is less than 0.5 which would round down and round up if the fractional part of the decimal input is more than 0.5.

## Special Cases (Infinity, Denormalized and NaN):

9. Input equates to final exponent greater than 90 (Special Case: Infinity)

Input: 893495409 (Round Up)

Exponent: 89

Processing	Output
<b>Normalized Decimal</b> 8934955	<b>Sign Bit</b> 0
<b>Final Exponent</b> 91 (Special Case: Infinity)	<b>Combination Bits</b> 11110
<b>E-Prime</b> Infinity (Exponent > 90)	<b>Exponent Continuation Bits</b> 000000
	<b>Mantissa Continuation Bits</b> 00000000000000000000
	<b>Final Output (Binary)</b> 0 11110 000000 0000000000 0000000000
	<b>Final Output (Hexadecimal)</b> 0x78000000
	<a href="#">Export</a>

10. Input equates to final exponent less than 101 (Special Case: Denormalized)

Input: -6423.34523 (RTNTTE)

Exponent: -99

Processing	Output
<b>Normalized Decimal</b> -6423345	<b>Sign Bit</b> 1
<b>Final Exponent</b> -102 (Special Case: Denormalized)	<b>Combination Bits</b> 01000
<b>E-Prime</b> 101 (01100101)	<b>Exponent Continuation Bits</b> 100101
	<b>Mantissa Continuation Bits</b> 00000000000000000000
	<b>Final Output (Binary)</b> 1 01000 100101 0000000000 0000000000
	<b>Final Output (Hexadecimal)</b> 0xA2500000
	<a href="#">Export</a>



11. Input equates to normalized decimal being NaN (Special Case: NaN)

Input:  $\sqrt{-1}$

Exponent: -5

Processing	Output
Normalized Decimal NaN	Sign Bit 0
Final Exponent (Special Case: NaN)	Combination Bits 11111
E-Prime NaN (Not-a-Number)	Exponent Continuation Bits 000000
	Mantissa Continuation Bits 00000000000000000000
	Final Output (Binary) 0 11111 000000 0000000000 0000000000
	Final Output (Hexadecimal) 0x7C000000
	Export

12. Input equates to normalized decimal being NaN (Special Case: NaN)

Input:  $5/0$

Exponent: 45

Processing	Output
Normalized Decimal NaN	Sign Bit 0
Final Exponent (Special Case: NaN)	Combination Bits 11111
E-Prime NaN (Not-a-Number)	Exponent Continuation Bits 000000
	Mantissa Continuation Bits 00000000000000000000
	Final Output (Binary) 0 11111 000000 0000000000 0000000000
	Final Output (Hexadecimal) 0x7C000000
	Export

Based on the outputs above, when the final exponent is above 90 it would indicate a special case of infinity where the output would be the combination bits will always be 11110, while the sign bit logic will be the same. And the rest of the bits are zeros. For the

denormalized special case, it would happen the final exponent is less than -101, the sign bit logic will be the same, E-prime will always be 101, the combination bits will always be 01000, as the first two bits of the binary equivalent of E-prime is 01, then for the last three bits will be based on 0 as a denormalized value would equate to 0 to a 4 bit binary value of (0000) where the last 3 will be appended to the right of 01 this logic will also apply for the rest of the normal cases when the first digit of the normalized decimal is a minor value (value of 0 to 7). The exponent continuation bits would be from the last 6 bits of the binary equivalent of E-prime, then the rest of the bits will be zeros. For the NaN special case, this will happen when the decimal input (or expression) equates to a Math error (e.g.  $\sqrt{-1}$ ,  $1/0$ , or a letter instead of a number). The combination bits will always be 11111 while the rest of the bits will be zeros.

### **Learnings and Conclusion:**

In conclusion, the students were able to make a functional and simple IEEE-754-Decimal-32-Floating-Point-Converter that can be useful in performing fast conversions. A feature to extract the respective inputs, processing, and output in a text file was also implemented. In the process of making the web application, the students were able to improve their skills in rounding floating point values while furthering their confidence in the decimal normalization process which is crucial in getting the correct output. The processing portion for the special cases are somewhat brute forced but it is done due to the consistent outputs of special cases. The students were also able to implement the decimal input to accept basic mathematical expressions such as square root and division. It can also accept letter or string input but would result in a NaN special case. Simple error handling is also implemented as the application will not output anything in the fields whenever one of the input fields are blank. Rounding methods are only applicable for decimal inputs of length greater than 7 digits. Lastly, exporting the output via a text file can only be done if a conversion was done correctly (correct inputs with outputs).