// Online C compiler to run C program online

#include <stdio.h>

void print\_float\_bits(float f) {

union {

float f;

unsigned int i;

} u;

u.f = f;

printf("float=%f\r\n",f);

for (int i = 31; i >= 0; i--) {

printf("%d", (u.i >> i) & 1);

}

printf("\n");

}

int main() {

// Write C code here

printf("Try programiz.pro\r\n");

float number = -13.25;

number = -13.25;

print\_float\_bits(number);

number = 2.0;

print\_float\_bits(number);

number = 0.5;

print\_float\_bits(number);

number = 8.5;

print\_float\_bits(number);

number = -8.5;

print\_float\_bits(number);

return 0;

}

Results

Try programiz.pro

float=-13.250000

11000001010101000000000000000000

float=2.000000

01000000000000000000000000000000

float=0.500000

00111111000000000000000000000000

float=8.500000

01000001000010000000000000000000

float=-8.500000

11000001000010000000000000000000

**Question: How to convert 8.5 floating to binary**

(8.5)10=(01000001000010000000000000000000)2

| **Field** | **Bits** | **Value** |
| --- | --- | --- |
| **Sign** | 0 | Positive |
| **Exponent** | 10000010 | 130 (decimal) |
| **Mantissa** | 00010000000000000000000 | Fractional part |

**✅ Step 2: Decode exponent**

* **Exponent bits**: 10000010 = **130**
* **Bias** for single-precision float = 127
* **Actual exponent** = 130 − 127 = 3

**✅ Step 3: Decode mantissa (aka fraction)**

* IEEE 754 uses **normalized form**, so the real mantissa is:
* 1.0001 (binary)
* Now convert 1.0001₂ to decimal:

**✅ Step 4: Final value**

* Use the formula:

**✅ Purpose of the Bias or (Recalculate the Actual Exponent):**

* To make **sorting by bit pattern** match **sorting by numeric value**
* To avoid using a separate sign bit for the exponent
* To represent both **positive and negative exponents** using only **positive binary numbers**

| **Binary Exponent Field** | **Decimal Value** | **Real Exponent = Field − 127** |
| --- | --- | --- |
| 00000000 (0) | 0 | Special case (denormal or 0) |
| 01111111 (127) | 127 | 0 |
| 10000000 (128) | 128 | +1 |
| 01111110 (126) | 126 | -1 |
| 11111111 (255) | 255 | Special (NaN or Infinity) |

C-Language Notes

-------------------------------------- Code Starts ------------------------

union {

int i;

unsigned int ui;

float f;

} int2float;

int intNum = 1091043328;

int2float.i = intNum;

printf("int=%d => float=%f\r\n",int2float.f,intNum);

float floatNum = 9.25;

int2float.f = floatNum;

printf("float=%f => int=%d\r\n",floatNum,int2float.i);

unsigned int uintNum = 1091043328;

int2float.ui = uintNum;

printf("unsigned int=%d => float=%f\r\n",int2float.f,uintNum);

int intNegNum = 3238526976;

int2float.i = intNegNum;

printf("int=%d => float=%f\r\n",int2float.f,intNegNum);

-------------------------------------- Code Ends ------------------------

Results

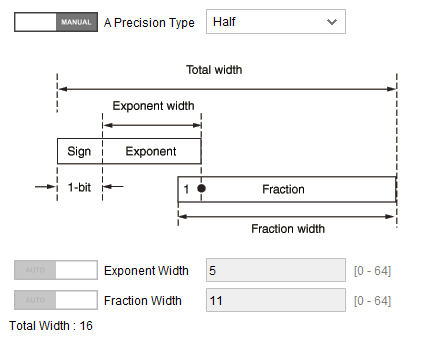
int=1091043328 => float=8.500000

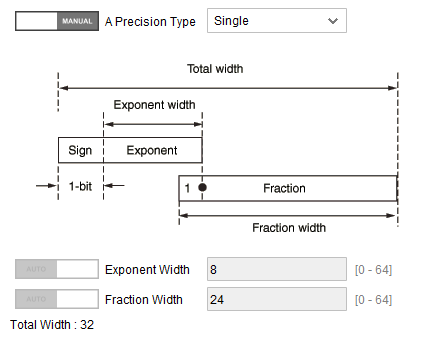
float=9.250000 => int=1091829760

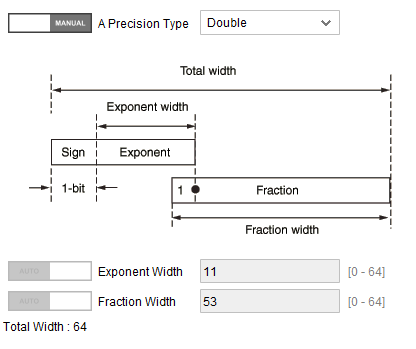
unsigned int=1091043328 => float=8.500000

int=-1056440320 => float=-8.500000

Learn Types







|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Float Type** | **Sign bits** | **Exponential bits** | **Fractional bits** | **Total bits** |
| Half | 1 | 5 | 10 | 16 |
| Single | 1 | 8 | 23 | 32 |
| Double | 1 | 11 | 52 | 64 |

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity first\_one\_detector is

port (

data\_in : in std\_logic\_vector(15 downto 0);

first\_one\_pos : out std\_logic\_vector(3 downto 0);

found : out std\_logic

);

end entity;

architecture Behavioral of first\_one\_detector is

begin

process(data\_in)

begin

found <= '1'; -- Assume found

if data\_in(15) = '1' then first\_one\_pos <= "1111";

elsif data\_in(14) = '1' then first\_one\_pos <= "1110";

elsif data\_in(13) = '1' then first\_one\_pos <= "1101";

elsif data\_in(12) = '1' then first\_one\_pos <= "1100";

elsif data\_in(11) = '1' then first\_one\_pos <= "1011";

elsif data\_in(10) = '1' then first\_one\_pos <= "1010";

elsif data\_in(9) = '1' then first\_one\_pos <= "1001";

elsif data\_in(8) = '1' then first\_one\_pos <= "1000";

elsif data\_in(7) = '1' then first\_one\_pos <= "0111";

elsif data\_in(6) = '1' then first\_one\_pos <= "0110";

elsif data\_in(5) = '1' then first\_one\_pos <= "0101";

elsif data\_in(4) = '1' then first\_one\_pos <= "0100";

elsif data\_in(3) = '1' then first\_one\_pos <= "0011";

elsif data\_in(2) = '1' then first\_one\_pos <= "0010";

elsif data\_in(1) = '1' then first\_one\_pos <= "0001";

elsif data\_in(0) = '1' then first\_one\_pos <= "0000";

else

first\_one\_pos <= "0000";

found <= '0';

end if;

end process;

end architecture;