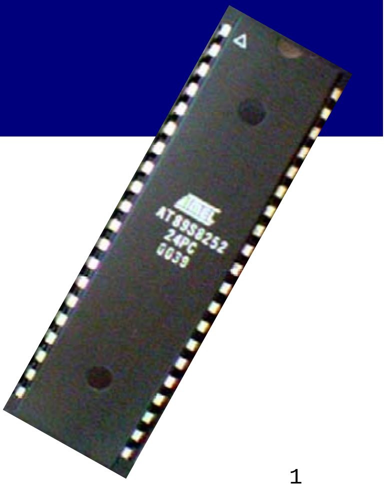
Digital Integrated Circuits & Microcontrollers



Chapter 2. Binary representation

Numeration Systems

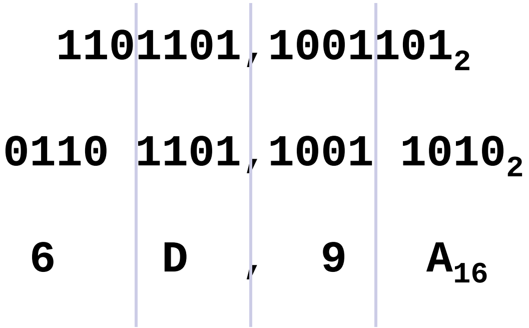
- **Decimal**
  - □ the set of symbols is:
    {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  - □ The integer number 493 in base 10:
    $493_{10} = 4\bullet10^2 + 9\bullet10^1 + 3\bullet10^0 =$
    $= 400 \quad + \ 90 \quad + 3$
  - □ The *rational* number 35,54 in base 10:
    $35.64_{10} = 3\bullet10^1 + 5\bullet10^0 + 6\bullet10^{-1} + 4\bullet10^{-2} =$
    $= 30 \quad + \quad 5 \quad + \quad 0.6 \quad + \ 0.04$

Numeration Systems

- **Binary**
  - □ the set of symbols is: {0, 1};
  - □ $11001_2 = 1\bullet2^4 + 1\bullet2^3 + 0\bullet2^2 + 0\bullet2^1 + 1\bullet2^0 =$
    $= 16 + 8 + 1 = 25$
  - □ $110.01_2 = 1\bullet2^2 + 1\bullet2^1 + 0\bullet2^0 + 0\bullet2^{-1} + 1\bullet2^{-2}=$
    $= 4 + 2 + 0.25 = 6.25$

Conversion between numeration systems

- **Conversion from binary into hexadecimal**
  - □ Group into 4 bits (nibbles)
  - □ Each nibble corresponds to a hexadecimal symbol:



$1101101{,}1001101_2$

$0110\ 1101{,}1001\ 1010_2$

$6\ \ D\ ,\ 9\ \ A_{16}$

Conversion between numeration systems

- **Conversion from decimal into base *b***
  - □ For the **integer part** we can write:

$$i = \left(\left(\left(\left(\mathrm{d}_k \mathrm{b} + \mathrm{d}_{k\text{-}1}\right)\mathrm{b} + \ldots + \mathrm{d}_3\right)\mathrm{b} + \mathrm{d}_2\right)\mathrm{b} + \mathrm{d}_1\right)\mathrm{b} + \mathrm{d}_0$$

- $\mathbf{d_0}$ is the remainder of division of *i* to ***b***
- The quotient (another integer) is divided to ***b***
- Repeat until reach 0.
- The remainder obtained after each division is the symbol $\mathbf{d}_k$ of representing into the base ***b***.

$$\boxed{i = \mathrm{d}_k \mathrm{b}^k + \mathrm{d}_{k\text{-}1}\mathrm{b}^{k\text{-}1} + \ldots + \mathrm{d}_3\mathrm{b}^3 + \mathrm{d}_2\mathrm{b}^2 + \mathrm{d}_1\mathrm{b}^1 + \mathrm{d}_0\mathrm{b}^0}$$

Conversion between numeration systems

- **Conversion from decimal into base *b***
  - □ For the **fractional part**:

$$f = \mathrm{d}_{-1}\mathrm{b}^{-1} + \mathrm{d}_{-2}\mathrm{b}^{-2} + \mathrm{d}_{-3}\mathrm{b}^{-3} + \ldots + \mathrm{d}_{-k}\mathrm{b}^{-k} + \ldots$$

- Multiply *f* with b

$$\mathrm{b} \cdot f = \mathrm{d}_{-1} + \mathrm{d}_{-2}\mathrm{b}^{-1} + \mathrm{d}_{-3}\mathrm{b}^{-2} + \ldots + \mathrm{d}_{-k}\mathrm{b}^{-k+1} + \ldots$$

- Keep the integer part from the right part, $\mathbf{d}^{-1}$, which is subtracted from the left part.
- Continue by multiplying the remaining fractionary part to b until reach 0.

$$\mathrm{b}\left(\mathrm{b} \cdot f - \mathrm{d}_{-1}\right) = \mathrm{d}_{-2} + \mathrm{d}_{-3}\mathrm{b}^{-1} + \ldots + \mathrm{d}_{-k}\mathrm{b}^{-k+2} + \ldots$$

Conversion between numeration systems

- **EXAMPLE:**
  - □ Represent the number 23.65 into base 2:
    - The integer part:

23 : 2 = 11	1	LSB
11 : 2 = 5	1	
5 : 2 = 2	1	
2 : 2 = 1	0	
1 : 2 = 0	1	MSB

$23_{10} = \mathbf{10111}_2$

Conversion between numeration systems

- The fractional part:

0.65 × 2 = 1.3	1	MSB
0.3 × 2 = 0.6	0	
0.6 × 2 = 1.2	1	
0.2 × 2 = 0.4	0	
0.4 × 2 = 0.8	0	
0.8 × 2 = 1.6	1	
0.6 × 2 = 1.2	1	
0.2 × 2 = 0.4	0	…

$0.65_{10}=\mathbf{0.10(1001)}_2$

- Results that number 0.65 cannot be exactly represented on a finite number of bits.

Negative numbers representation

- MSB – sign bit.
  - □ 0 for positive numbers (+);
  - □ 1 for negative numbers (–).
- The rest of *N*–1 bits are for value representation.

	2^{N-2} … 2^0
s	**m**
1 bit	← *N*−1 bits →

Negative numbers representation

- The representation: sign bit, magnitude
  - □ **EXAMPLE:**

    9 = 0 01001

    –9 = 1 01001
  - □ The range of representation:
    - 2^{N-1} positive values between 0 and 2^{N-1}–1.
    - 2^{N-1} negative values between –(2^{N-1}–1) and 0.

	2^{N-2} … 2^0
s	**m**
1 bit	← *N*−1 bits →

Negative numbers representation

- **Two’s complement** representation
  - □ The negative numbers representation is obtained by addition of 2^N.
  - □ **EXAMPLE:**
    - For N=6 bits (2^N = 64).

      13 = 001101_2

      –13 corresponds to 64 + (–13) = 51 = 110011_2

Two’s complement representation

- For obtaining negative numbers:
    - Each bit is complemented;
    - Add 1.
  - □ **EXAMPLE:**
    - for 13 = 001101_2

      complement each bit: 110010_2

      add 1: 110011_2 = –13
    - for –13 = 110011_2

      complement each bit: 001100_2

      add 1: 001101_2 = +13

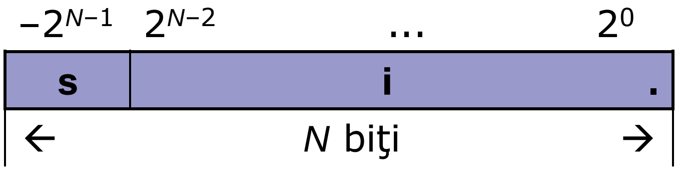
Two’s complement representation

- The range of two’s complement representation
  - 2^{N-1} positive values between 0 and $2^{N-1}-1$.
  - 2^{N-1} negative values between -2^{N-1} and -1.
- The result of adding a number with its two’s complement is 0:

13+	001101_2
−13	110011_2
= 0	1000000_2

Integer numbers representation

- Binary representation is considered **right aligned** (decimal point is right to LSB).
- MSB represents the sign bit.



- Integer two’s complement range:

$$-2^{N-1},\dots,-1,0,\dots,2^{N-1}-1$$

Integer numbers representation

- **EXAMPLE:** (for *N*=4 bits)

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal	Binary
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111

Integer numbers representation

- ***Sign integers addition***

$$\begin{array}{r|r} & \mathit{\cancel{1}111} \\ +3 & 0011 \\ -2 & 1110 \\ \hline 1 & 0001 \end{array} \qquad \begin{array}{r|r} & \mathit{011} \\ -5 & 1011 \\ +3 & 0011 \\ \hline -2 & 1110 \end{array} \qquad \begin{array}{r|r} & \mathit{\cancel{1}1} \\ -3 & 1101 \\ -4 & 1100 \\ \hline -7 & 1001 \end{array}$$

  - □ An overflow occurs if the result is outside of the *N* bits representation range:

$$\begin{array}{r|l} +3 & 0011 \\ +6 & 0110 \\ \hline 9 & 1001{=}{-7} \end{array} \qquad \begin{array}{r|l} -3 & 1101 \\ -6 & 1010 \\ \hline -9 & 0111\ {=}7 \end{array}$$

Integer numbers representation

- ***Sign bit extension***
  - □ Needed when **increasing** the number of bits for the integer part.
  - □ Sign bit is copied to the left toward MSB.

	N=4 biţi
+3	**0**011
−3	**1**101

	N'=8 biţi
+3	**0000 0**011
−3	**1111 1**101

Integer numbers representation

- ***Multiplying by a power of 2***
  - □ **Multiplying** by 2^k is equivalent with **shifting left *k* bits** and **filling with 0** toward LSB.

	N=8 biţi
3	0000 0011
$3\cdot2^2$	0000 11**00**

	N=8 biţi
−3	1111 1101
$-3\cdot2^2$	1111 01**00**

Integer numbers representation

- ***Dividing by a power of 2***
  - **Dividing** by 2^k is equivalent with **shifting right *k* bits** and **sign bit extension**.

	N=8 biţi
24	0001 1000
$24/2^3$	**000**0 0011

	N=8 biţi
−24	1110 1000
$-24/2^3$	**111**1 1101

Integer numbers representation

- ***Integers multiplication***
  - **unsigned integers**
  - The result in double precision representation
  - Binary multiplication with 0 and 1

6	0110
×5	×0101
	0110
	0000
	0110
	0000 +
30	0011110

Integer numbers representation

- Multiplication is equivalent with consecutive **shift** and **add** operations.
  - □ For example 5 can be expressed:

$$5 = 2^0 + 2^2$$

  - □ Multiplication can be computed:

$$6 \times 5 = 6 \times \left(2^0 + 2^2\right) = 6 \times 1 + 6 \times 2^2$$

$$\begin{array}{r|l} 6\times1 & 0110 \\ 6\times2^2 & \underline{0110\quad +} \\ & 0011110 \end{array}$$

Integer numbers representation

- ***Integers division***
  - □ Successive subtractions of the divisor from the dividend.

$$15|\underline{\ 3}$$
$$\ 5$$

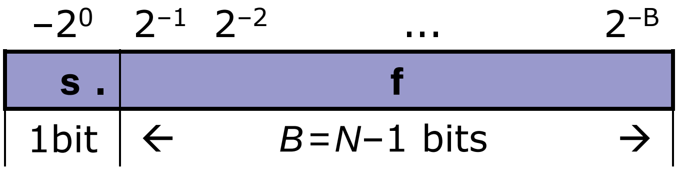
$$\begin{array}{r|l} 1111 & 11 \\ \hline -\underline{11} & 101 \\ 0011 & \\ -\underline{11} & \\ 00 & \end{array}$$

Fractional numbers representation

- A fractional part *f* is any number who's modulus satisfies the inequality:

$$0.0 \leq | f | < 1.0$$

  - □ Left aligned: binary point is at the right of MSB



  - □ Fixed point fractional representation range:

$$-1,...,-2^{-B},0,2^{-B}...,1-2^{-B}$$

Fractional numbers representation

- **EXAMPLE:**
  - □ for *N*=4 bits

Decimal	Binary
0	0000
0.125	0001
0.250	0010
0.375	0011
0.500	0100
0.625	0101
0.750	0110
0.875	0111

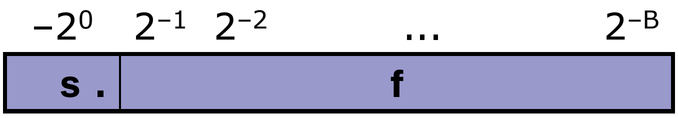
Decimal	Binary
-1	1000
-0.875	1001
-0.750	1010
-0.625	1011
-0.500	1100
-0.375	1101
-0.250	1110
-0.125	1111

Fractional numbers representation

- **Q*m.n* format**
  - □ *n* bits for fractional part;
  - □ (optional) specify the number of bits *m* for the integer part, excluding the sign bit (MSB);
  - □ The complete binary representation has *1+m+n* bits.
- **EXAMPLE** (for *N*=16 bits):
  - □ Q15 means 15 bits for the fractional part (16 bits with the sign bit)
  - □ Q1.14 has 1 bit for the integer part, 14 bits for the fractional part and the sign bit.

Fractional numbers representation

- ***Quick conversion of fractional numbers into binary***
  - □ *f* represented on *N*=*B*+1 bits is an **integer multiple of 2^{-B}**

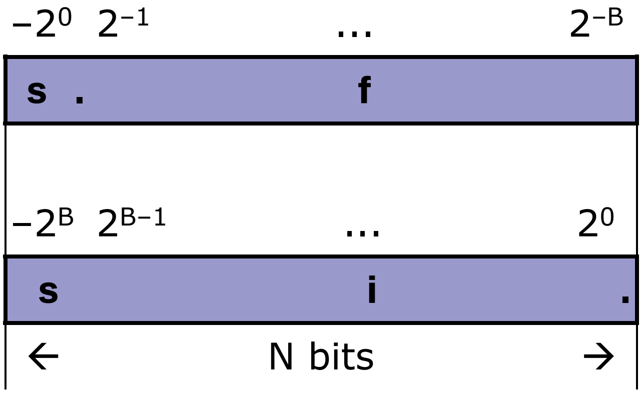


  - □ Let *i* the corresponding integer multiple:

$$i = f \cdot 2^B$$

    - Equivalent with a left shifting of *f* with B bits.

Fractional numbers representation



-2^0 2^{-1} … 2^{-B}

s . f

-2^B 2^{B-1} … 2^0

s i .

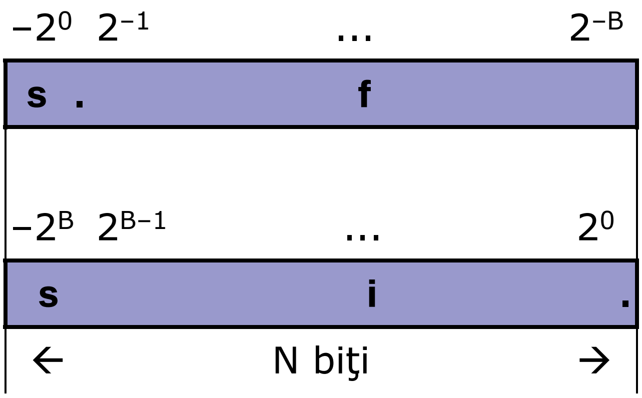
← N bits →

- □ Converting *f* from binary to decimal
  - consider the binary representation for the corresponding signed integer *i* and then, divide by 2^B

Fractional numbers representation

- **EXAMPLE:**
  - □ Binary to decimal (N=8 bits, B=7),
    - For: 0.010 0110
decimal integer: 0010 0110 = 38_{10}
divide by 2^7=128: 38/128 = 0.296875
    - For: 1.110 1100
two's complement: –0001 0100 = $–20_{10}$
divide by 2^7=128: –20/128 = –0.15625

Fractional numbers representation



-2^0 2^{-1} … 2^{-B}

s . f

-2^B 2^{B-1} … 2^0

s i .

← N biţi →

- □ Converting *f* from decimal to binary
  - multiply *f* by 2^B (shift left B bits)
  - Represent the integer part *i* as a signed integer on *N* bits.

Fractional numbers representation

- **EXAMPLE:**
  - □ Decimal to binary (*N*=8 biţi, *B*=7)
    - For: 0.875
multiply by 2^7: $0.875 \cdot 128 = 112_{10}$
represented binary: $112_{10} = 01110000_2$
    - For: –0.625
multiply by 2^7: $-0.625 \cdot 128 = -80_{10}$
represented in two's complement: $80_{10} = 01010000_2$
$-80_{10} = 10110000_2$

Fractional numbers representation

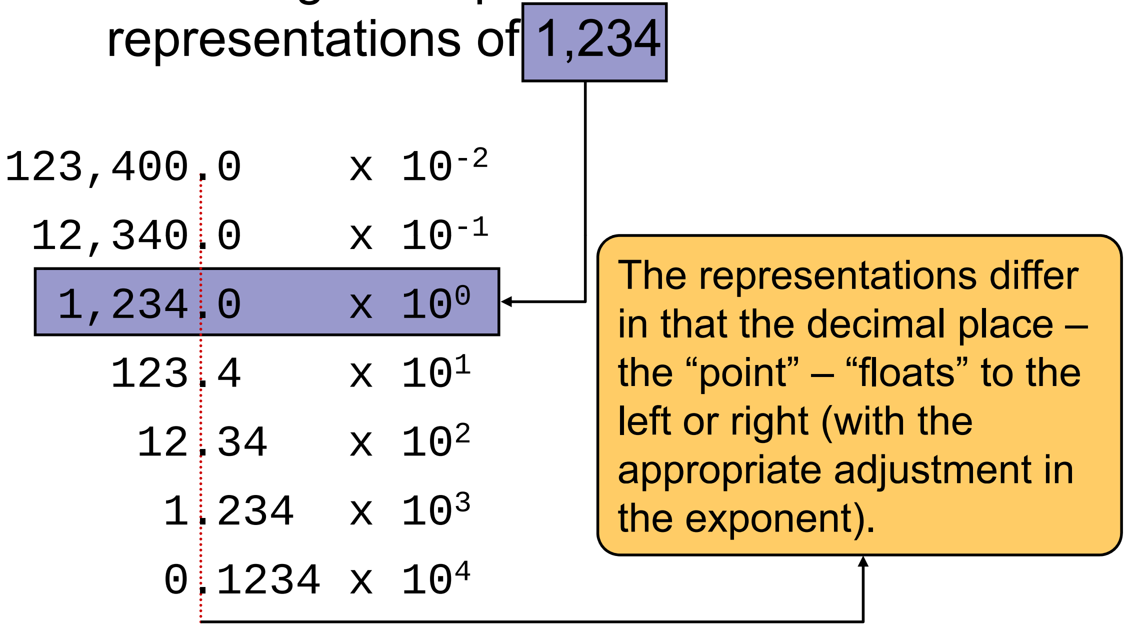
- For 0.65 (N=8 bits, B=7)
  multiply by 2^7: $0.65 \cdot 128 = 83.2_{10}$
  take the integer part: $83_{10} = 01010011_2$

- In the last example the **quantization error** (by truncation) appears since the fractional 0.65 cannot be exactly represented on 8 bits.
  - The result 0.1010011_2 is equal to 0.6484375.
  - The quantization error is:

$$\begin{array}{r} 0.65 \\ \underline{-\ 0.6484375} \\ = 0.0015625 \end{array}$$

Floating point representation

- The following are equivalent
  representations of 1,234



```
123,400.0    x 10-2
 12,340.0    x 10-1
  1,234.0    x 100
    123.4    x 101
     12.34   x 102
      1.234  x 103
      0.1234 x 104
```

$123{,}400.0 \times 10^{-2}$

$12{,}340.0 \times 10^{-1}$

$1{,}234.0 \times 10^{0}$

123.4×10^{1}

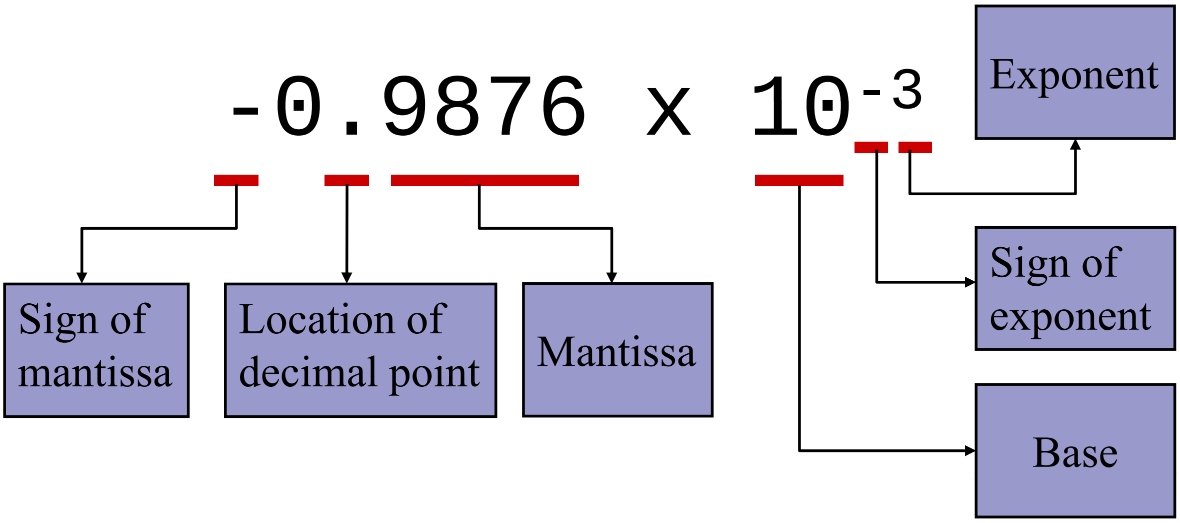
12.34×10^{2}

1.234×10^{3}

0.1234×10^{4}

The representations differ in that the decimal place – the “point” – “floats” to the left or right (with the appropriate adjustment in the exponent).

Parts of a Floating Point Number



-0.9876×10^{-3}

Exponent

Sign of mantissa

Location of decimal point

Mantissa

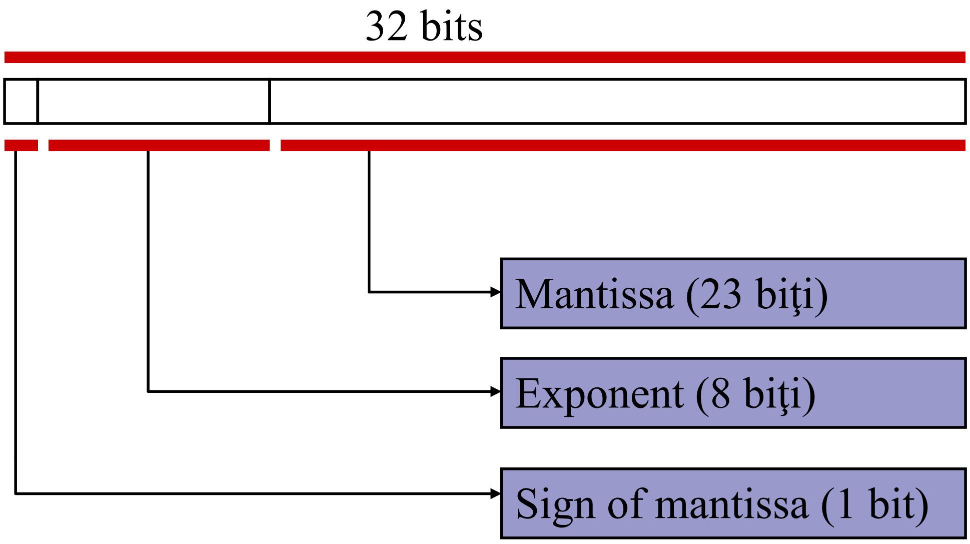
Sign of exponent

Base

IEEE 754 Standard

- Single precision: 32 bits, consisting of...
  - Sign bit (1 bit)
  - Exponent (8 bits)
  - Mantissa (23 bits)
- Double precision: 64 bits, consisting of…
  - Sign bit (1 bit)
  - Exponent (11 bits)
  - Mantissa (52 bits)

Single Precision Format



Normalization

- The mantissa is *normalized*
  - □ Has an implied decimal place on left
  - □ Has an implied “1” on left of the decimal place
- E.g.,
  - □ Mantissa → `10100000000000000000000`
  - □ Represents… $1.101_2 = 1.625_{10}$

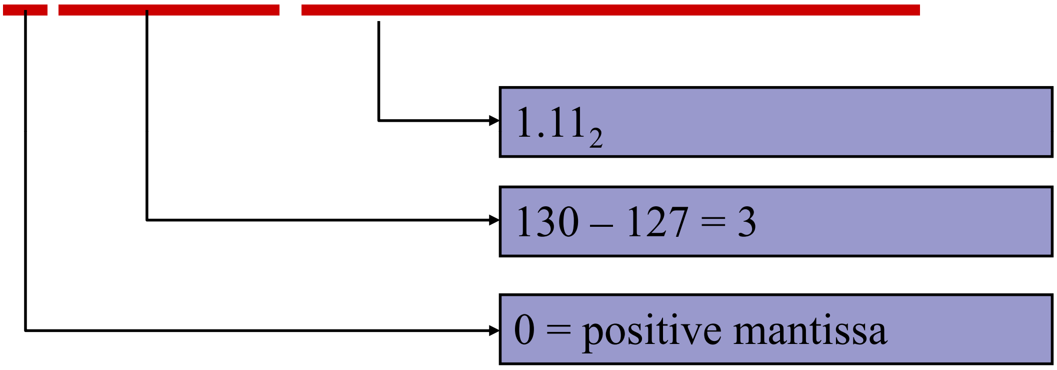
Excess Notation

- To include +ve and –ve exponents, “excess” notation is used
  - □ Single precision: excess 127
  - □ Double precision: excess 1023
- The value of the exponent stored is larger than the actual exponent
- E.g., excess 127,
  - □ Exponent → `10000111`
  - □ Represents… `135 - 127 = 8`

Example

- Single precision

`0 10000010 11000000000000000000000`



1.11_2

130 – 127 = 3

0 = positive mantissa

$$+1.11_2 \text{ x } 2^3 = 1110.0_2 = 14.0_{10}$$

Hexadecimal

- It is convenient and common to represent the original floating point number in hexadecimal
- The preceding example…

```
0 10000010 11000000000000000000000
  4    1    6    0    0    0    0    0
```

Converting from Floating Point

- What decimal value is represented by the following 32-bit floating point number?

$C17B0000_{16}$

Converting from Floating Point

- Step 1
  - □ Express in binary and find S, E, and M

C17B0000$_{16}$ =

$$\underbrace{1}_{S}\ \underbrace{10000010}_{E}\ \underbrace{11110110000000000000000}_{M}{}_2$$

1 = negative
0 = positive

Converting from Floating Point

- Step 2
  - □ Find “real” exponent, *n*
  - □ n = E – 127
    = 10000010_2 – 127
    = 130 – 127
    = 3

Converting from Floating Point

- Step 3
  - □ Put S, M, and *n* together to form binary result
  - □ (Don't forget the implied "1." on the left of the mantissa.)

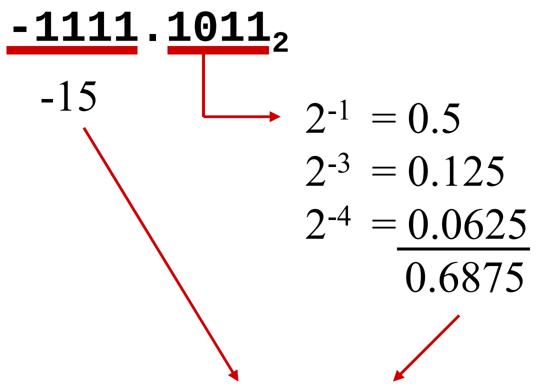
$-1.1111011_2 \times 2^n =$

$-1.1111011_2 \times 2^3 =$

-1111.1011_2

Converting from Floating Point

- Step 4
  - □ Express result in decimal



-1111.1011_2

-15

$2^{-1} = 0.5$

$2^{-3} = 0.125$

$2^{-4} = 0.0625$

0.6875

Answer: -15.6875

Converting <u>to</u> Floating Point

- Express 36.5625_{10} as a 32-bit floating point number (in hexadecimal)

Converting <u>to</u> Floating Point

- Step 1
  - □ Express original value in binary

36.5625_{10} =

100100.1001_2

Converting to Floating Point

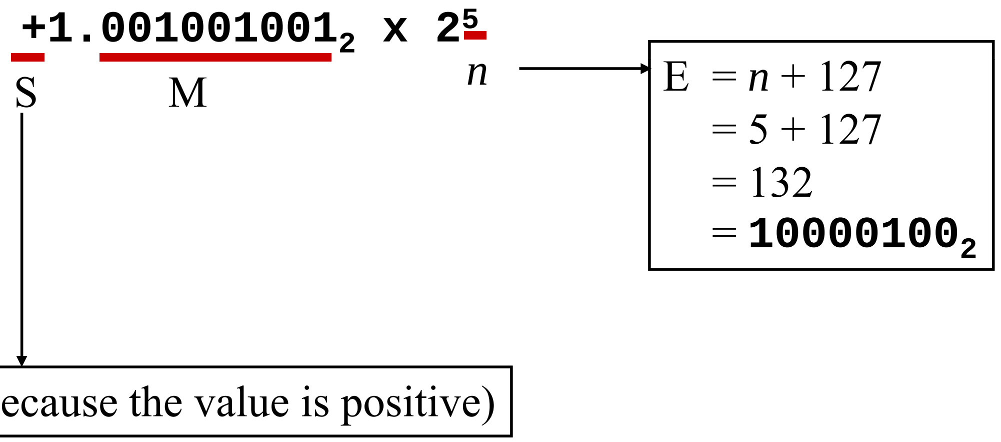
- Step 2
  - □ Normalize

$100100.1001_2 =$

$1.001001001_2 \times 2^5$

Converting to Floating Point

- Step 3
  - □ Determine S, E, and M



$\underbrace{+}_{S}1.\underbrace{001001001}_{M}{}_2 \times 2^{\underbrace{5}_{n}}$

$n \rightarrow$ E $= n + 127$
$= 5 + 127$
$= 132$
$= \mathbf{10000100}_2$

S = 0 (because the value is positive)

Converting to Floating Point

- Step 4
  - □ Put S, E, and M together to form 32-bit binary result

$$\underbrace{0}_{\text{S}}\ \underbrace{10000100}_{\text{E}}\ \underbrace{00100100100000000000000}_{\text{M}}{}_2$$

Converting to Floating Point

- Step 5
  - □ Express in hexadecimal

$$0\ 10000100\ 00100100100000000000000_2 =$$

$$0100\ 0010\ 0001\ 0010\ 0100\ 0000\ 0000\ 0000_2 =$$

$$4\quad 2\quad 1\quad 2\quad 4\quad 0\quad 0\quad 0_{16}$$

Answer: 42124000_{16}