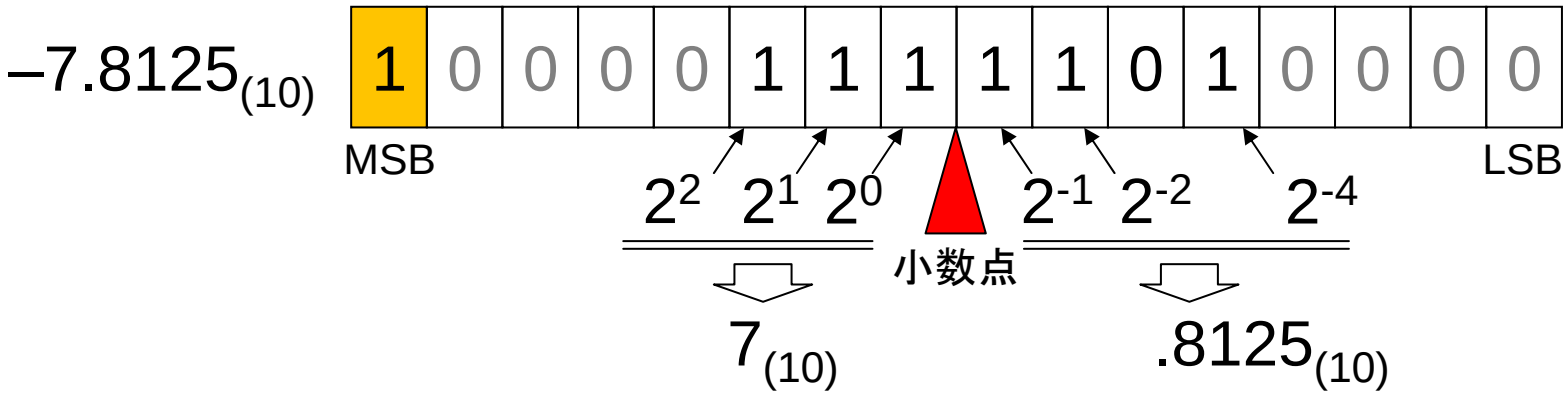
実数の表現

実数の表現

- nビットの実数表現では 2^n 種類の実数しか表現できない.
  - 実数は不可算無限である.
  - コンピュータ上では限られた範囲の実数しか扱えない.
  - コンピュータ上では限られた精度でしか実数を表せない.
- 実数の表現では誤差の問題が常に生じる.
  - 特定のビット幅に収まり切らなかった数は**丸め誤差**となる.
- 実数の表現の例:
  - **固定小数点表現**
  - **浮動小数点表現**

固定小数点表現

- 決められたビット幅 n に，実数 m を2進数に表現したものをそのまま入れる.
- このとき上位，ならびに下位の余ったビットには 0 を補填する.
- 小数点は特定のビットの右にあるものと考える.
- 最上位ビットは符号を表す符号ビットとして使用してもよい．m が正ならば符号ビットは 0，負ならば符号ビットは 1 にする.
- 符号なし整数表現や符号つき絶対値表現は固定小数点表現の特別な形式である.（小数点が右端にある.）



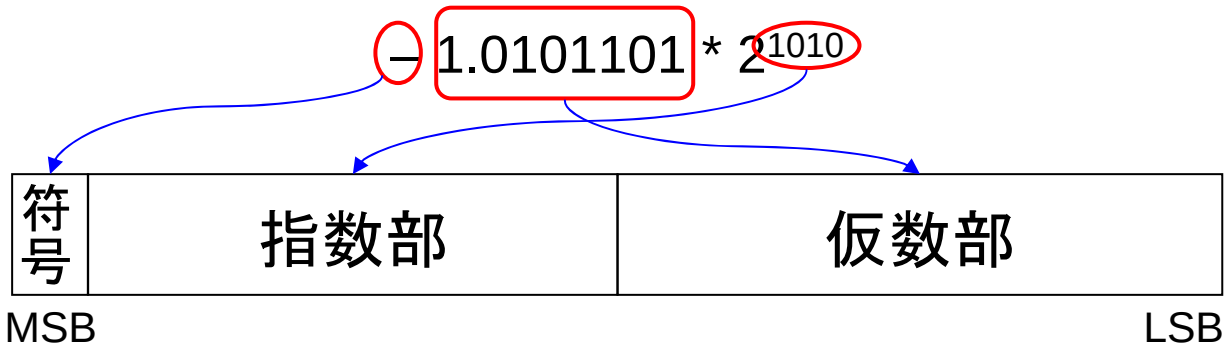
浮動小数点表現(1)

- **科学記数法**: 科学技術計算における実数の表記法.
  - 通常の記数法: 3.141592…, 2.718281828… *etc*.
  - 科学記数法: $6.02 * 10^{23}$, $1.60*10^{-19}$ *etc*.
  - 6.02や1.60にあたる部分を**仮数**と呼ぶ.
  - 23, –19にあたる部分を**指数**, 10を**基数**と呼ぶ.
- 科学記数法では**正規化**が行われる.
  - 仮数の整数部は常に1桁とする.
  - 仮数の整数部は常に 0 以外の数とする.
- 浮動小数点表現は科学記数法の2進数版ある.
  - 基数は10ではなく2とする.
  - 正規化の結果, 浮動小数点表現では仮数の整数部は常に1となる.
- C言語の float/double 型は「浮動小数点表現」.

浮動小数点表現(2)

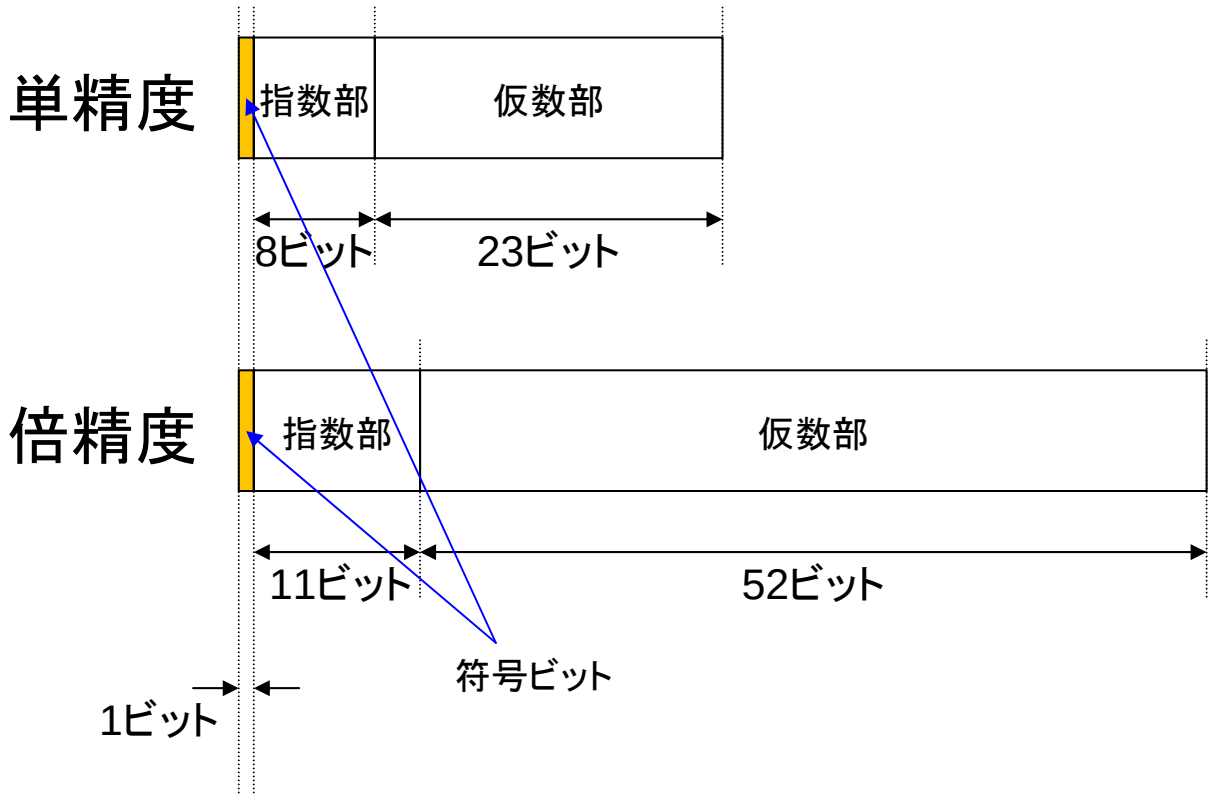
基本的な発想:

- 特定の幅のビット列を上から順に下記の3つのフィールドに分割する.
  - 最上位ビットを**符号ビット**(常に1ビット)として.
  - あるビット幅を**指数部**として.
  - 残りの部分を**仮数部**として.
- 実数の符号を符号ビットで表現する.(非負のとき0, 負のとき1)
- 実数を正規化された2進科学記数法で表現し, 指数を指数部に, 仮数を仮数部に格納する.



IEEE754（1）

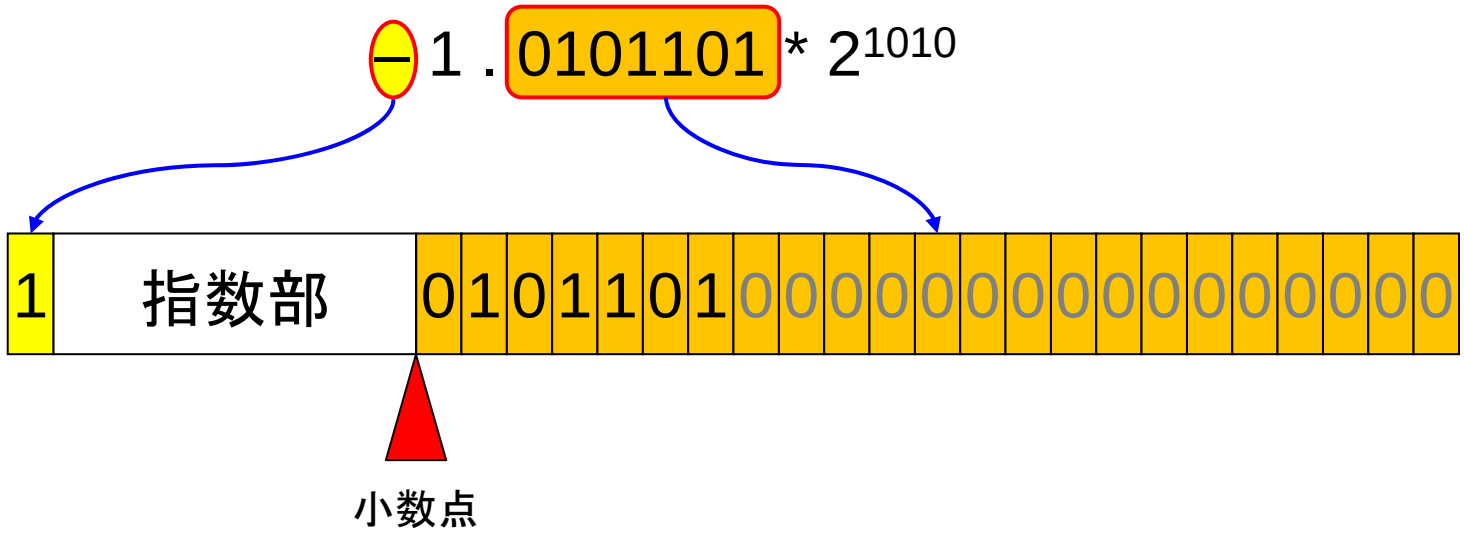
- **IEEE754** は浮動小数点表現の標準規格である.
- 単精度表現と倍精度表現の2種類の表現が定義されている.



IEEE754(2)

仮数部の表現:

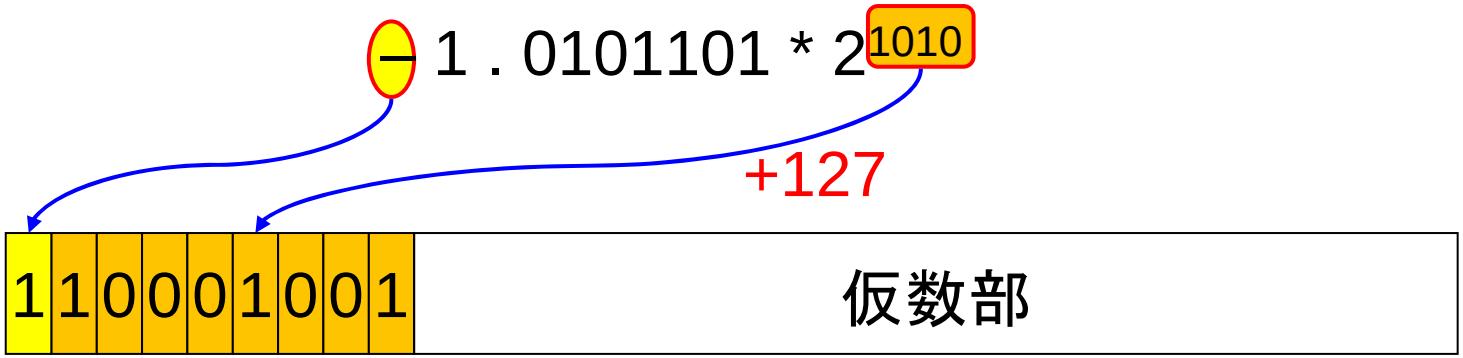
- 仮数部には，正規化後の仮数の小数部が固定小数点表現で格納される．小数点の位置は仮数部の最上位ビットの左とする．
- 正規化後，仮数の整数部は常に1(＝ **implicit MSB**)となるので，データ表現中に格納する必要がない．(**けち表現**)



IEEE754(3)

指数部の表現:

- n ビットの指数部は，実際の指数より，$2^{n-1}-1$ 大きい整数を符号なし整数表現で格納する．(**バイアス表現**，**ゲタ履き表現**)
  - 単精度(指数部=8ビット)では，実際の指数より127大きい整数を格納．したがって，可能な指数は –127 から 128 まで．(本当は–126から127まで．)
  - 倍精度(指数部=11ビット)では，実際の指数より1023大きい整数を格納．したがって，可能な指数は –1023 から 1024 まで．(本当は–1022から1023まで．)



IEEE754（4）

例外的な表現:

- 指数部が 000…0 のときは特別.
  - 仮数部が xxx…x のとき、単精度の場合は $0.xxx…x * 2^{-126}$、倍精度の場合は$0.xxx…x * 2^{-1023}$と解釈する（非正規化数）.
  - すなわち，0 は 000…0 と表現される（指数部と仮数部は全て0）.
- 指数部が 111…1 のときは特別.
  - 仮数部が 000…0 のときは，符号ビットの値に応じて +∞ または –∞ と解釈する.
  - 仮数部が 000…0 でないときは，非数（NaN）と解釈する.

IEEE754(5)

オーバーフロー：指数部で表現できないほど値が大きい

- 単精度の場合，絶対値が 2^{128} 以上の値は表現できない．
- 倍精度の場合，絶対値が 2^{1024} 以上の値は表現できない．

アンダーフロー：指数部で表現できないほど値が小さい

- 単精度の場合，絶対値が2^{-126}未満の非零値は表現できない．
- 倍精度の場合，絶対値が2^{-1022}未満の非零値は表現できない．

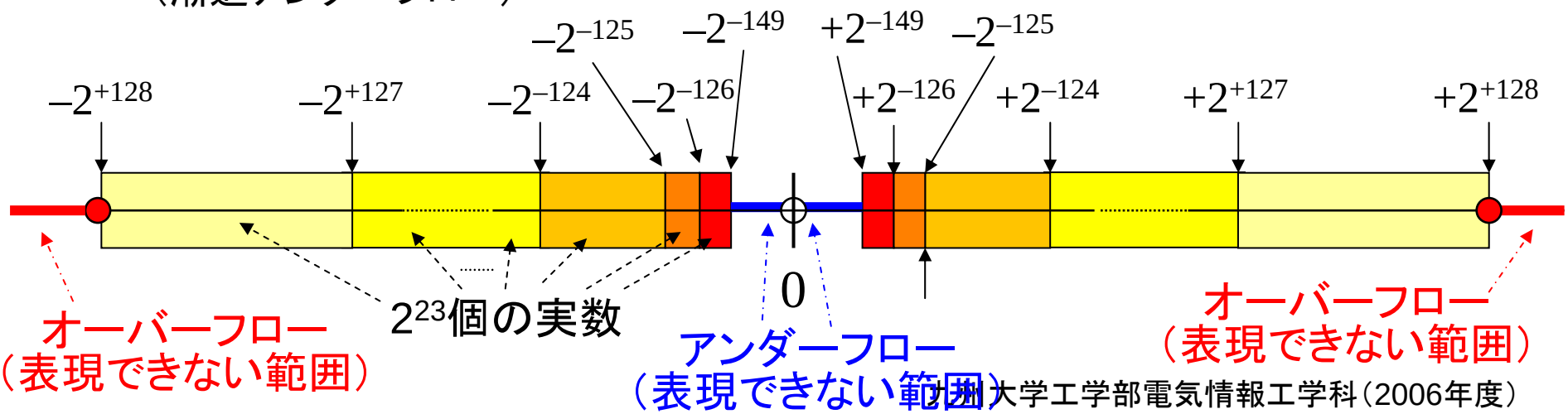
補足：
IEEE754規格では不正規化数をサポートしている。これにより，2^{Emin}（*Emin*は単精度で－126，倍精度で－1022）未満の値を表現できるようになる．値は0.仮数部×2^{Emin}となり，このような場合を漸進アンダーフローと呼ぶ．

IEEE754(6)

浮動小数点表現(単精度)	範囲	間隔
0 00000000 ****…********	$\pm 0.000..0 * 2^{-126} \sim \pm 0.111..1 * 2^{-126}$	2^{-149}
* 00000001 ****…********	$\pm 1.000..0 * 2^{-126} \sim \pm 1.111..1 * 2^{-126}$	2^{-149}
* 00000010 ****…********	$\pm 1.000..0 * 2^{-125} \sim \pm 1.111..1 * 2^{-125}$	2^{-148}
…	…	…
* 11111110 ****…********	$\pm 1.000..0 * 2^{+127} \sim \pm 1.111..1 * 2^{+127}$	2^{+104}

不正規化数として表現可能な範囲
(漸進アンダーフロー)

正規化数として表現可能な範囲



IEEE754（まとめ）

- 単精度表現（32ビット）と倍精度表現（64ビット）を定義

- 正規化数以外（非正規化数、無限大、非数など）の表現も可能

単精度の場合

	*E*と*F*	数値
正規表現	$0 < E < 255$	$(-1)^S \times (1+F) \times 2^{(E-127)}$
非正規表現	$E=0, F \neq 0$	$(-1)^S \times (0+F) \times 2^{(-126)}$
ゼロ	$E=0, F=0$	$(-1)^S \times 0$
非数（$\sqrt{-1}$等）	$E=255, F \neq 0$	NaN (Not a Number)
∞	$E=255, F=0$	$(-1)^S \times \infty$

- *E*と*F*の値によりどの表現に相当するか一意に定まる
- *E*=255の場合は指数部が全て‘1’

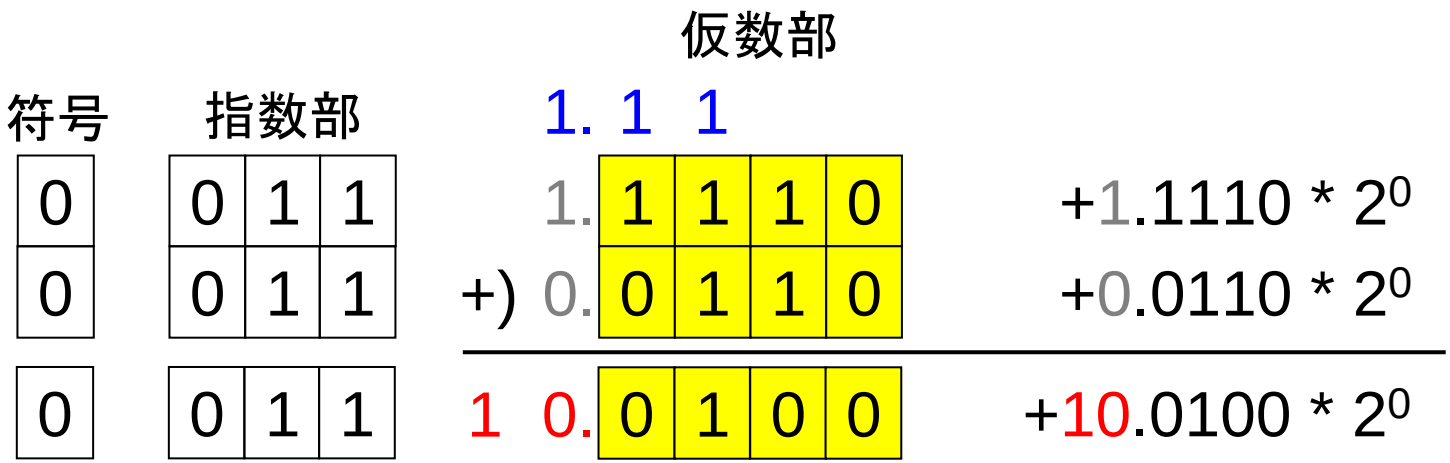
浮動小数点加算(1)

1. 指数部を比較し，指数部の小さい方の仮数部を，指数部が等しくなるまで右シフトする．(＝1ビット右シフトすると指数部を1増やす．)

符号	指数部：ゲタは$3_{(10)}$	仮数部	
0	0 1 1	1. 1 1 1 0	$+1.1110 * 2^{0}$
0	0 0 1	1. 1 0 0 1	$+1.1001 * 2^{-2}$
0	0 1 1	1. 1 1 1 0	$+1.1110 * 2^{0}$
0	0 1 0	0. 1 1 0 0	$+0.1100 * 2^{-1}$
0	0 1 1	1. 1 1 1 0	$+1.1110 * 2^{0}$
0	0 1 1	0. 0 1 1 0	$+0.0110 * 2^{0}$

浮動小数点加算(2)

2. 仮数部を加算する.



浮動小数点加算(3)

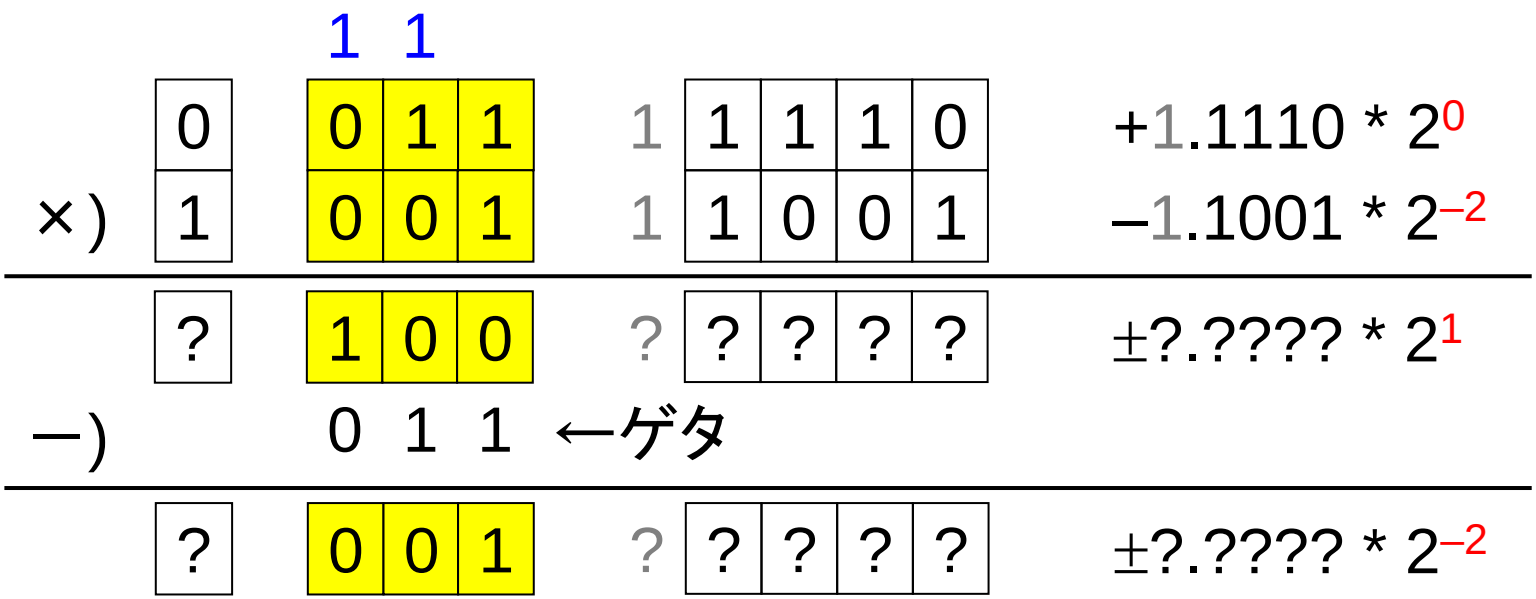
3. 結果を正規化する. 仮数部の整数部が 1 になるまで, 仮数部を右／左シフトする. (＝1ビット右／左シフトすると指数部を1増やす／減らす.)

0	0	1	1	1	0.	0	1	0	0	$+10.0100 * 2^0$
0	1	0	0		1.	0	0	1	0	$+1.0010 * 2^1$

- 正規化の結果, オーバーフローまたはアンダーフローが発生していないかチェックする(つまり、計算結果が表現できる実数の範囲を超えていないか確認する).
- 正規化のあと丸め処理を行う. 丸め処理の結果が正規形でない場合は, 再度正規化を行う.

浮動小数点乗算(1)

1. 指数部を加算する．加算した後，ゲタ(指数部が n ビットのとき $2^{n-1}-1$)を引く．



浮動小数点乗算(2)

2. 仮数部を乗算する.

	0		0	1	1		1.	1	1	1	0	$+1.1110 * 2^{0}$
×)	1		0	0	1		1.	1	0	0	1	$-1.1001 * 2^{-2}$

					1	1	1	1	0	
				0	0	0	0	0		
			0	0	0	0	0			
		1	1	1	1	0				
	1	1	1	1	0					

符号無し加算

1	0.	1	1	1	0	1	1	1	0	$\pm 10.11101110 * 2^{-2}$

?		0	0	1	1	0.	1	1	1	0

浮動小数点乗算(3)

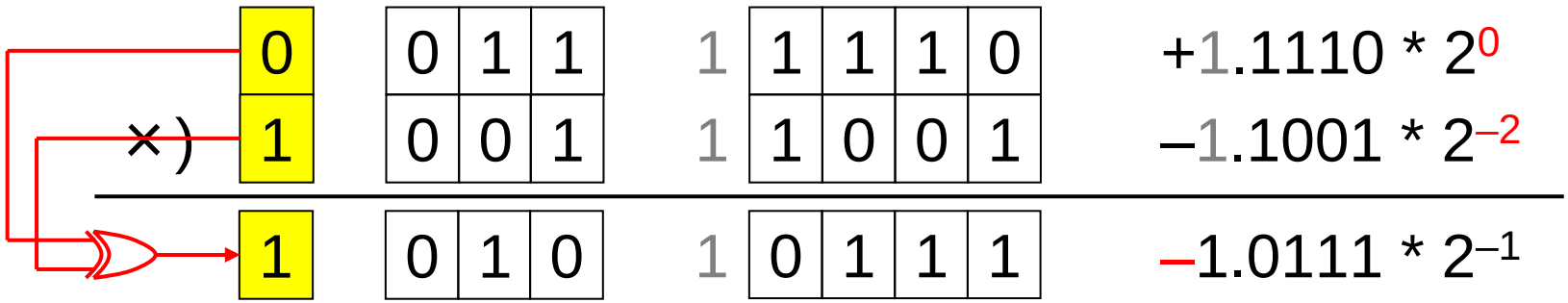
3. 結果を正規化する．仮数部の整数部が 1 になるまで，仮数部を右シフトする．（＝1ビット右シフトすると指数部を1増やす．）

?	0	0	1	1	0.	1	1	1	0	$\pm 10.1110 * 2^{-2}$
?	0	1	0		1.	0	1	1	1	$\pm 1.0111 * 2^{-1}$

- 正規化の結果，オーバーフローまたはアンダーフローが発生していないかチェックする(つまり、計算結果が表現できる実数の範囲を超えていないか確認する)．
- 正規化のあと丸め処理を行う．丸め処理の結果が正規形でない場合は，再度正規化を行う．

浮動小数点乗算(4)

4. 符号を確定する．結果の符号ビットは，掛けられる数と掛ける数の符号ビットが等しければ 0，等しくないとき 1 とする．



浮動小数点乗算(精度に関する考察)

- 先に示した浮動小数点乗算で得られた結果

$$-1.875 * 2^{0} \times 1.5625*2^{-2} = -1.4375*2^{-1}$$

$(+1.1110_{(2)} * 2^{0})$　$(-1.1001_{(2)} * 2^{-2})$　$(-1.0111_{(2)} * 2^{-1})$

- 先に示した浮動小数点乗算での正しい演算結果

$$-1.875 * 2^{0} \times 1.5625*2^{-2} = -1.46484375*2^{-1}$$

$(+1.1110_{(2)} * 2^{0})$　$(-1.1001_{(2)} * 2^{-2})$　$(-1.011101110_{(2)} * 2^{-1})$

なぜ、値が異なるのだろう？

仮数部の乗算を行った際に下位4ビットを切り捨てた！
(つまり、演算結果は丸められている！)→丸め誤差