ビット、バイト、キロ、メガ、

- ビット(bit):2進数1桁(0か1か)
- バイト(byte):2進数8桁(8ビット) 英数字1文字

- キロ:一般に10^3を表すが、1キロバイトは1,024バイト(2^{10}バイト)
- メガ:一般に10^6を表すが、1メガバイトは1,048,576バイト(2^{20}バイト)
- ギガ: 10^9
- テラ: 10^{12}
- ペタ: 10^{15}

コンピュータでの数の表現

- 桁数を(32ビットなどに)固定
- 例:4ビットでの表現は2^4=16通り
- 0000, 0001, …, 1111
- これで正負の数をどう表すか？
  - 絶対値表現
    - 先頭の1ビットで正負の符号を、残りで絶対数を表す
    - ＋7から－7まで表現
  - ゲタばき表現
    - 一定数を足すことで負数も正として表現
    - 0～15で、-8～7を表す

2の補数表現

- 正の数、負の数の表現
- nビットで表現するとき、

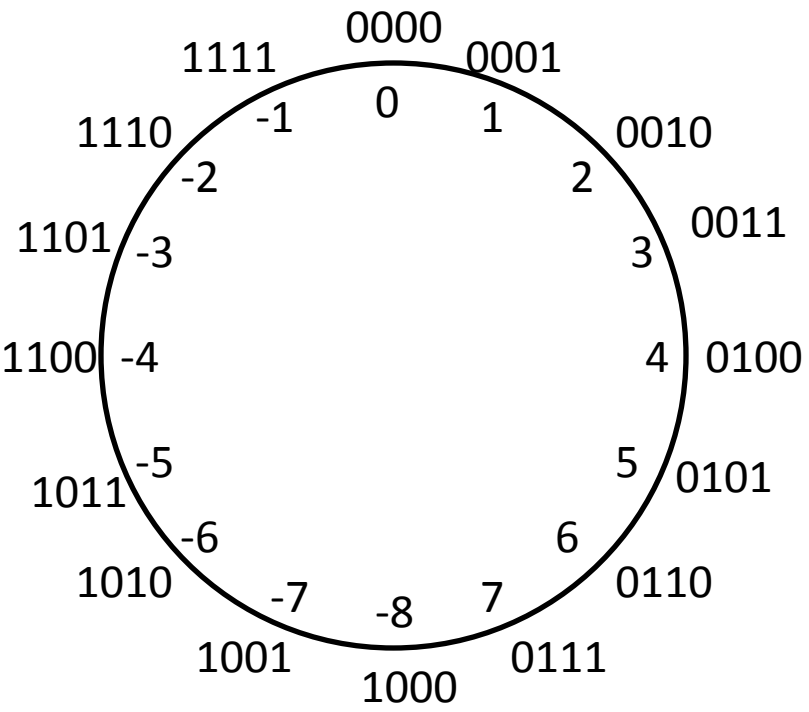
「Nの2の補数」を2^n-Nで定義

- 例: 3の2の補数(-3)は16-3

=13に対応する2進数(1101)

- 最上位のビットが1なら負
- nビットで2^{n-1}-1～-2^{n-1}を

表現



2の補数の性質

- 負数も正数と同様に計算できる
  - －19＋5＝－14
  - 1110 1101＋0000 0101＝1111 0010

  - ー19×5＝ー95
  - 1110 1101×0101＝100 1010 0001 下8桁はOK

教科書p.28のプログラム

```
#include <stdio.h>
int main(void) {
    int i;
    i = 1;
    while (i>0)
        i = i + 1;
    printf(“%d ¥n”,i);
}
```

- iを1ずつ増やしていくが、なぜか-2147483648を表示して終了

停止する理由

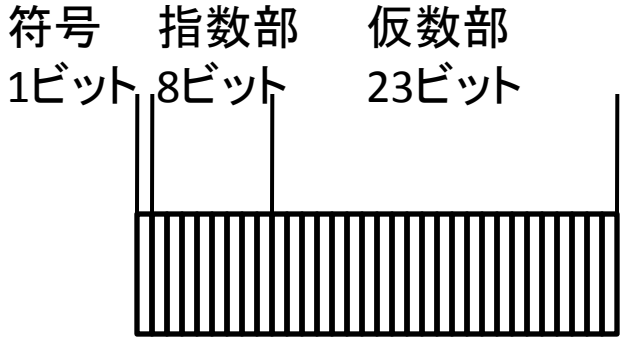
- 整数型の変数iを32ビットで表現
- 1 0000 0000 … 0000 0001
- 2 0000 0000 … 0000 0010
- …
- 2,147,483,647 $=2^{32-1}-1$ (32ビットで表せる正の最大数) 0111 1111 … 1111 1111
- ~~2,147,483,648~~ 1000 0000 … 0000 0000
  -2,147,483,648 $=-2^{32-1}$ (32ビットで表せる負の最小数)

実数の表現

- 固定小数点数
  - 整数部分に用いるビット数と小数部分に用いるビット数を固定
  - 3.14159265358979, 0.00000000006673
  - (同じ桁数で)表現できる実数の範囲が狭い
- 浮動小数点数
  - 指数を使って表現された実数
  - 6.673×10^{-11}, 6.0221415×10^{23}
  - (同じ桁数で)表現できる実数の範囲が広い

コンピュータ内部での浮動小数点数の表現

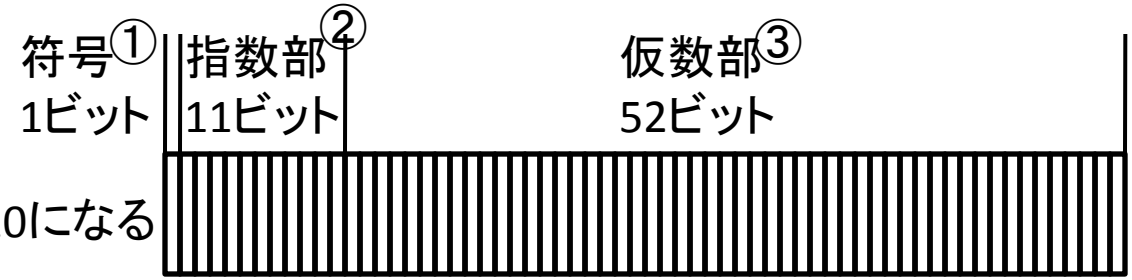
- 単精度浮動小数点数



- 倍精度浮動小数点数

浮動小数点数の具体例

- 符号(仮数部)×$2^{(指数部)}$の形で実数を表現
- 0.1の浮動小数点数
- $0.1_{10}=(1/10)_{10}=(1/1010)_2$
  $=0.0\ 0011\ 0011\ 0011\ 0011\ \ldots_2$
  $=1.1001\ 1001\ 1001\ 1001\ \ldots_2 \times 2^{-4}$
- 指数部に1023(10^{10}-1)を足す
  - 仮数部11ビット(10^{11}=2048)で1024～-1023を表す
  - 1023+(-4)=1019=011 1111 1011②
- 符号部は正なら0負なら1
  - この場合は0①
- 最終桁は四捨五入
  - 1001 1001 … 1010③



最終的に
0011 1111 1011 1001 1001 … 1010になる

丸め誤差

- $(0.1_{10}=)0.0\ 0011\ 0011\ 0011\ \ldots_2$は無限に続く
- 有限桁数の仮数部におさまるように丸める(四捨五入)際に生じる誤差

- 教科書p.34のプログラム
  - 0.1×10ー1はゼロだが、丸め誤差があるために5.55112e-17(5.55112×10^{-17})となってしまう

浮動小数点数の計算での問題

- 丸め誤差
  - 2進数で無限小数となる実数を、途中で切り捨てるために生じる誤差
- 情報落ち
  - 値が大きく異なる2数の加算の際、大きな値の指数に揃えると小さな値が仮数部の表現範囲に入らず欠落するために生じる誤差
- 桁落ち
  - 値がほぼ等しい数同士の減算の際に有効数字が減少することで生じる誤差

情報落ち

- 値が大きく異なる2数の足し算 123.4+0.1234

$$
\begin{array}{r}
123.4 \\
+\ \ 0.1234 \\
\hline
123.5234
\end{array}
$$

有効数字4桁とすると
0.0234は誤差

- 大きい数に小さい数をたくさん足す時、小さい数同士を足して値を近づけてから大きい数と足すと回避できる

情報落ちの回避

- 大きな数(a)と小さい数(b)を足す場合

  s = a + b; ←情報落ちが生じる

  r = (s – a) – b; ←情報落ちで発生する誤差

  - 本来は0になるはずの値

- 足し算を繰り返すとき、前回の誤差を修正

$$\underbrace{\underbrace{a1 + a2}_{\text{和と誤差を計算}} + a3}_{\text{誤差を修正し和と誤差を計算}} + \ldots$$

カハン加算公式

```
s = a[1]; c = 0;
for(i = 2; i <= n; i++){
    y = a[i] – c;   /*前のループで発生した誤差を修正*/
    t = s + y;        /* 総和を計算、一時的にtに格納 */
    c = (t – s) – y /* このループでの誤差を計算 */
    s = t;            /* sに総和の値を格納 */
}
```

- sは総和の値、cは前回の誤差

桁落ち

- 値がほぼ等しい2数の引き算

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

a=1,b=1000.001,c=1の時、b=1000.001

$$\sqrt{b^2 - 4ac} = \sqrt{999998} = 999.99899$$

bと$\sqrt{b^2 - 4ac}$の値が近く、引き算すると有効数字の桁数が減ってしまう

- 式変形してそのような引き算を回避