コンピュータ概論　第6回

数値データと数値計算

2進数

- 数を0と1だけで表現した数
- 実数もすべて2進数で表わす
- コンピュータの数値表現でも利用
  - ただし、以下の点で注意が必要
    - 決められた桁数（ビット長）で表現
    - 正負符号や小数点もバイナリ表現

cf. 16進数

- 数を0～9とA～Fで表現した数
- 2進数4桁分を1桁で表現できる

  0000→0、0101→5、1010→A、1111→F
- 2進（バイナリ）表現の代わりに
  16進（HEX）表現が用いられることが多い
- 8進数というものもある

  10進数と桁が大体同じになるようにするため
- 例

  $61_{(10)} \Rightarrow 3D_{(16)}$

Cでは　0x3D　のように書くと16進数と見なされる

整数のバイナリ表現

- 各ビットが2のn乗を表す（8ビット長）

m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0
0	0	1	1	1	1	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

負の整数

- 最上位ビットに符号情報

m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0
0	0	1	1	1	1	0	1
$\pm$	2^6	2^5	2^4	2^3	2^2	2^1	2^0

- 3種類の表現方法
  - 符号絶対値
  - 下駄ばき
  - 2の補数

符号絶対値

- 最上位ビットを符号専用とする。
  - ＋　→　０
  - －　→　１
- 簡単に作れる
- 正数の表現は符号無しの場合と同じ
- 例　8ビット長

  ３　⇒　　００００００１１

  -３ ⇒　　１０００００１１

下駄履き(Offset)

- 0を2進数のちょうど真ん中に合わせる
  - 011…11 を －1
  - 100…00 を 0 (最上位ビットだけ1)
  - 100…01 を 1
- 数の大きさの順番がそのまま
- 他の方式と違い，最上位ビットは1の時＋
- 正数も普通の符号無し表現と異なる
- 例 8ビット長

  3 ⇒ 10000011

  -3 ⇒ 01111101

cf. 1の補数

- すべてのビットを反転させる
  - 000…01 を 1
  - 111…10 を −1
- ＋0と−0が存在する
- 論理否定とも呼ぶ（数値表現には使わない）
- 正数は普通の符号無し表現と同じ
- 例 8ビット長

  3 ⇒ 00000011

  -3 ⇒ 11111100

2の補数

- 1の補数に000…01加える
  - 000…01　を　　1
  - 111…11　を　－1
- 1加えるので－0が存在しない
- オーバーフローを利用して，加減算が容易
  - 整数値を扱う場合に一般的に用いれる
- 正数は普通の符号無し表現と同じ
- 例　8ビット長

  3　⇒　　00000011

  -3　⇒　　11111101

オーバーフロー

- 扱える桁の範囲を越えてしまうこと
- 符号の反転などの計算エラーにつながる
- 16ビットの整数の場合，通常，
  -32768~32767を越えると発生する
  0111～1は32767だが、
  1000～0は-32768になる（符号の逆転）
- 2の補数はオーバーフローを利用することで、加減算に適した表現となっている

2の補数

$11_{(10)}$

↓2進数へ

$00001011_{(2)}$

↓反転

11110100

↓1加える

$11110101_{(2)}$

↓10進数へ

$-11_{(10)}$

↓完成

$-11_{(10)}$

加算

$$3+(-1)$$

2の補数

$$\begin{array}{rrr} 0 & 1 & 1 \\ + & 1 & 1 \\ \hline 1 & 0 & 1 \end{array}$$

2

符号絶対値

$$\begin{array}{rrr} 0 & 1 & 1 \\ + & 1 & 1 \\ \hline 1 & 0 & 0 \end{array}$$

0

下駄履き

$$\begin{array}{rrr} 1 & 1 & 1 \\ + & 0 & 1 \\ \hline 1 & 0 & 0 \end{array}$$

-2

実数のバイナリ表現

- 固定小数
  - 整数部と小数部の桁数が固定

m_4	m_3	m_2	m_1	m_0	.	m_{-1}	m_{-2}	m_{-3}
1	0	0	1	1	.	1	0	1
±	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}

- 浮動小数
  - 指数表現で小数点の位置が変化

$$M \times r^E$$

浮動小数とは

- r進数で$M \times r^E$という形で表現
- 10進数なら2.998×10^8のように

2進数浮動小数の場合

- $M \times 2^E$（ただし，$1 \leqq |M| < 2$)
- 10進数の0.15625ならば
  - 1.25×2^{-3}（正式にはMを2進数表現して、1.01×2^{-3} ）

IEEE32方式

- 米国電気電子学会により作られた
  標準規格（JIS規格のようなもの）
- 32ビットで1つの浮動小数点数を表わす
- パソコンやワークステーション用の
  OSで採用
- 64ビットのIEEE64もある

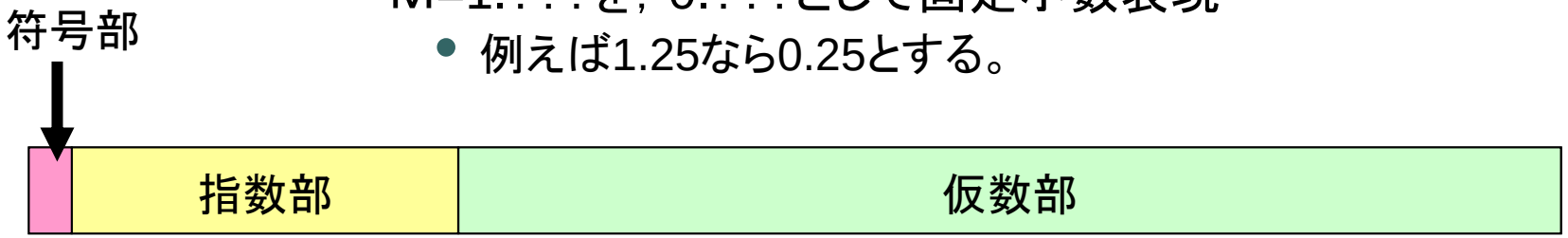
IBM方式

- IBMにより作られた方式
- 現在はスーパーコンピュータなどの一部の大型コンピュータ以外では使われない
- r＝16で表現する

$$M \times 16^{E} \ (1 \leq \left|\mathrm{M}\right| < 16)$$

32ビットの使い方（IEEE方式）

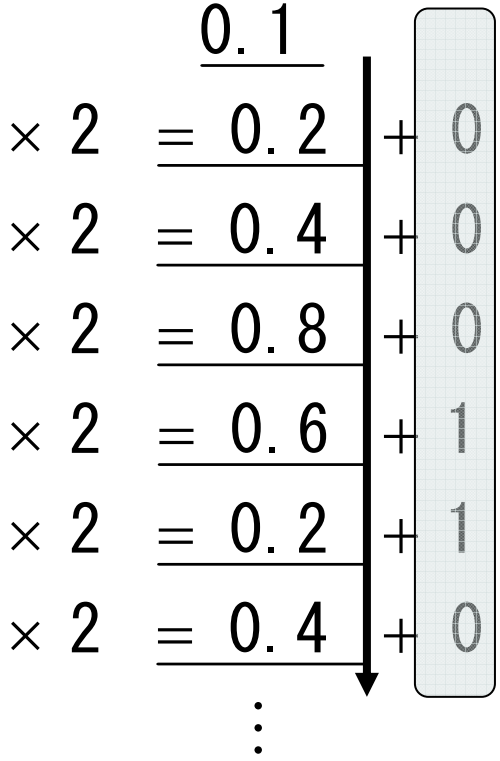
- 1ビット
  - 符号　正は0，負は1
- 2～9ビット
  - E（指数部）
  - 下駄履き（整数）8ビット
- 10～32ビット
  - M（仮数部）
  - M=1.???を，0.???として固定小数表現
    - 例えば1.25なら0.25とする。



丸め誤差

- 10進数の0.1を2進数で表現すると…
  - 0. 0001100110011･･･（循環小数）
  - 10進数の有限小数が全て
    2進数の有限小数で表現
    できるとは限らない
    例）1/3は3進数なら0.1だが
    10進数だと0.333･･･（無限小数)
- コンピュータでは桁数は有限
  - 途中で切り捨てをする

丸め誤差



数値演算における注意点

- 2進数を基本
  - 負数表現や浮動小数点の工夫
- コンピュータ固有の問題
  - オーバーフロー
  - 丸め誤差
- 世間の常識は専門分野の非常識
  - 『コンピュータは計算を間違えない』という大嘘
    - コンピュータに計算をさせるときには注意が必要
      - 表現桁数(ビット長)の把握
      - 表現方式の特徴の把握

例題1

5をIEEE32で表現する

- 1.25×2^2
- 符号は正→0
- E=2→1000 0010
- M=1.25だから0.25→
  0100 0000 0000 0000 0000 000
- **0100 0001 0010 0000 0000 0000 0000 0000**

例題2

-0.625をIEEE32で表現する

- -1.5×2^{-1}
- 符号は負→1
- E=-1→0111 1111
- M=1.5だから0.5→
  1000 0000 0000 0000 0000 000
- **1011 1111 1100 0000 0000 0000 0000 0000**

Cにおける数値表現

- 整数
  - 16ビットまたは32ビット（OSにより異なる）
  - 負数を2の補数で表現
- 実数
  - IEEE方式の浮動小数
  - 32ビット（単精度）
  - 64ビット（倍精度）

Cにおける数値の表示

- 実行中にデータを表示するための命令
  - `printf`（システム関数）
- 表示できるもの
  - 文字列
  - 数値
  - 書式にあわせた数値交じり文字列

printf("表示文字列");

- 表示文字列
  - 表示する文字列
  - センターのシステムでは、日本語も可能
  - 制御文字（cf.ASCIIコード表）も一部使える
    - ¥n　改行
    - ¥t　タブ
    - ¥¥　¥を表示する

プログラム例

- 文字列を表示するプログラム

```
#include<stdio.h>
main(void){
  printf("Hello, 自分の名前¥n");
}
```

文字列

改行

printf("表示形式",値…)

- 表示形式
  - 以下の形式指定文字を文字列中に入れる
    - その場所に値の内容を表示
  - 値に対応する形式を記入
    - %d　整数
    - %f　実数(単精度)
    - %lf　実数(倍精度)
    - %c　ASCII文字(半角)1文字
    - %s　文字列(1文字以上の文字)
- 値…(複数の場合は『,値,値,…』)
  - 形式指定文字の順番に並べる

プログラム例

- 数値を表示するプログラム

```
#include<stdio.h>
main(void){
  printf("Number is %d¥n",55);
}
```

表示形式

%dのところに挿入される値

プログラム例

- 複数の数値を表示するプログラム

```
#include<stdio.h>
main(void){
  printf("%d+%d=%d¥n",3,4,7);
}
```

表示形式　　順番に表示される

Cにおける数値演算

- 演算子（演算記号）

数学記号	演算子
＋	+
－	-
×	*
÷	/

- 数式
  - 数学における数式と同じ演算の優先度
  - 他にもC固有の演算子がある（テキスト参照）
  - カッコ記号は『（』『）』のみ
  - 代入は右辺から左辺に（左辺には変数しか書けない）

プログラム例

- 数式の値を表示するプログラム

```
#include<stdio.h>
main(void){
  printf("%d×%d=%d¥n",3,4,3*4);
}
```

表示形式

掛け算は*

プログラム例

- オーバーフローの発生

```
#include<stdio.h>
  main(void){
  printf("Integer A is % d¥n", 100000);
  printf("Integer B is % d¥n", 200000);
  printf("The answer of A * B is %d¥n", 100000 * 200000);
}
```

どちらもintの範囲内

積の結果は？

おさらい

- 整数の2進表現
  - 符号絶対値方式
  - ゲタ履き方式
  - 2の補数方式
  - オーバーフロー
- 浮動小数の2進表現
  - IEEE方式
  - IBM方式
  - 丸め誤差
- printfによる値の表示

プログラム課題

- 形式の異なる数値を表示する次のプログラムを作成し，実行せよ.

```
#include<stdio.h>
main(void){
  printf("Number is %d¥n",200/3);
  printf("Number is %lf¥n",200/3);
  printf("Number is %c¥n",200/3);
}
```