前回:
授業の概要を説明
C言語を例としてプログラムの基本要素とその実行モデルを説明

今回:
計算機のなかでのデータの表現のしかた
とくに数値の表現について
  自然数(0以上の整数)
  整数(負の値も含む)
  実数(小数点のある数)

キーワード:
基数, MSB, LSB, 2の補数, 符号ビット, 符号拡張,
固定小数点, 浮動小数点, 指数部, 仮数部
---
2 数の表現

コンピュータの中のデータ
  変数に格納される
  (実際はメモリという記憶素子に格納)
いろいろなデータ(数値, 文字, 画像, 音声)
  基本的にはすべて数値で表現される
数値にもいろいろ
  0以上の整数(自然数)
  整数(負数も含めて)
  実数(固定小数点数,浮動小数点数)

コンピュータのメモリ
  0/1の値を記憶する素子がたくさん
  ビット (bit, binary digit)
「数をビットの並びでどのように表現するか」

2.1 自然数

2.1.1 基数表記法
m 個の数字の並びで自然数を表記する
  m: 桁数
  B: 基数
    一桁の数字は0から(B-1)までB種類
  N_i: (下から) i 桁目の数字 (i >= 0)
  B^i: (下から) i 桁目の重み

一番下の桁が 1の位, その左隣が Bの位,
さらにその左隣が Bの二乗(B^2と書く)の位, ...

数式で書くと
\sum_{i=0}^{m-1} {N_i B^i}
(この表記法について - Latex風味)
  _ 下につく添字
  ^ 上につく添字
  { } 記法の上でのカッコ
  \sum Σ

最上位桁 N_{m-1} MSD (Most Significant Digit)
最下位桁 N_0   LSD (Least Significant Digit)

B進法 - 基数Bを用いた表記法
B進数 - B進法で表現した数

表現可能範囲 (m桁で)
  0 - (B^{m}-1)

何進の表記かわからないと困る
  末尾に基数Bを小さく付加
    カッコ書きで付加する流儀もあり
  先頭に記号をつける流儀もあり (たとえばC言語)
    0x - 16進
    0 - 8進

2.1.2 操作
ある数値 v を B進法で表記するとき

i桁目 N_i の取り出し (i >= 0)
  (v / (B^i)) % B
  除算は整数除算 - 小数点以下切捨て
  % は剰余をもとめる演算

加算
  桁ごとに加算
  B 以上になった桁は上の桁に 1 を加える (繰り上がり)
  (普通の筆算と同じ)

2.1.2.1 オーバーフロー
桁数が固定であるので、加算の結果があふれる場合がある
たとえば 2進4桁 の場合
  1010 + 0111 = 10001
10 + 7 = 17 が2進4桁に収まらないため
単純に4桁だけ取り出すと 0001 で誤った結果に

検出の方法
- 結果が被加数のいずれもより小さくなった
- 最上位桁からの繰り上がりが発生

2.1.3 性質
最下位が 0 である数は, B で割り切れる

最下位に 0 を付け加えると, 値は B 倍になる
最下位の数字をひとつ取り除くと, 値は 1/B になる (1未満切捨て)
(桁を右や左にずらすことに相当 - shiftする という)

ある値vを B進で表記すると何桁になるか
  floor((\log_B v) + 1)
  floor関数: 小数点以下切捨て

2.1.4 変換
数値への変換
  定義式どおり (2.1.1)
数値からの変換
  最下位桁からやるのが簡単
  くりかえし{
    v % B が最下位桁の値
    v = v / B
  }

2.1.5 計算機でよく使われる基数
10進 (decimal)
  おなじみ
2進 (binary)
  2進の一桁を bit という
    (binary digit)
  2進数の MSD/LSD を MSB/LSB ともいう
8進 (octal)
  2進への変換
    2^3 = 8 だから, 2進3桁は8進1桁
    桁毎に3bitの2進数に変換すればよい
  相互変換は簡単 - 2.1.4の割り算不要
  例: 8進 17 -> 001(2進での1) 111(2進での7)
                  = 2進 001111
16進 (hexadecimal)
  10-15を表す数字が必要
    a,b,c,d,e,f (大文字でも可)を使うのが普通
  2進への変換
    2^4 = 16 だから, 2進4桁は16進1桁
  相互変換は簡単 - 割り算不要
  例: 16進 1a -> 0001(2進での1) 1010(2進でのa)
                    = 2進 00011010

8進, 16進は, 2進と簡単に変換できる上,
2進に比べて桁数も少ないのでよく利用される

2.2 整数
負数の表現が入ってくる
いろいろな方式がある
  符号付絶対値
    符号(+/-)と自然数の組み合わせ
    MSBを符号, 残りを自然数
  1の補数
  2の補数

2.2.1 補数
桁数をmとする

m桁のB進数 X, Y

真補数 (Bの補数) (radix complement)
  YがXの真補数, とは
  X+Y=B^{m}

擬補数 ((B-1)の補数) (diminished radix complement)
  YがXの擬補数, とは
  X+Y=B^{m}-1

2進の場合(B=2)
真補数を 2の補数 two's complement
偽補数を 1の補数 ones' complement
  (one が複数になっている; つまり 1111..11 というココロ)

n >= 0 として、
負数 -n を nの真補数であらわす

計算機は内部は2進数で表現するので
  負数は2の補数であらわす

例: 3ビットの2進数が表現する値

	符号なし	符号付絶対値	1の補数	2の補数
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-3	-4
101	5	-1	-2	-3
110	6	-2	-1	-2
111	7	-3	-0	-1

2の補数表現の特長
  正+正, 正+負, 負+正, 負+負が
  すべて同じ方法で計算できる
  (符号なしの場合と同じ扱い)

例: 2+(-1) をいろいろな表記法であらわし、
それらを符号なしとして加算してみる
  符号付絶対値 010+101 = 111 (-3)
  1の補数   010+110 = 000 (0)
  2の補数   010+111 = 001 (1) - これだけ正解

さらに:
  減算 = 負数の加算 だから
  負数が求まれば減算は加算回路でできる

mビットの2の補数表現での表示可能範囲
-(2^{m-1}) から {2^{m-1}}-1 まで
対称でないのは 0 が正数側に入っているから

8ビット -128 ... 127
16ビット -32768 ... 32767

32ビット -20億 ... 20億
64ビット -9×10^18 ... 9×10^18

最上位ビットMSBが符号ビット
  0 なら 正か0
  1 なら 負

2.2.2 2の補数の求め方
Aの2の補数を B とすると
定義から A+B=2^{m}
  だから B = 2^{m}-A なのだが
  引き算不要でできる
Aのビット反転を\overline{A}とする
  (\overline は上線の表記)
  ビットごとに0と1を入れ替えたもの
    これは 1の補数 になっている
  A + \overline{A} = 11...1
                   = 2^{m}-1
  \overline{A} + 1 = 2^{m} - A
                   = B
例: 0101 (2進での5) の2の補数は
  ビット反転 1010
  +1して   1011

2.2.3 減算(負数の加算)の例
基数2, 4bit として
0111 - 0101 (7 - 5) を求めるには
7   + (-5)
0111 + 1011=10010
5bit目を無視すると
0010 = 2(10進) - 正解

2.2.4 符号拡張 (sign extension)
mビットの2の補数表現を, m+1以上のビットで表す
上のほうのビットに何をつめるか
  無条件に0をつめてはダメ
    1011 (-5) -> 00001011 (11)
  正と0なら, 0 をつめる
  負なら, 1 をつめる
つまり, それまでの符号ビットを左にコピーすればよい
例: 4ビット → 8ビット
0101 → 00000101
1011 → 11111011

2.2.5 オーバーフロー
加算の結果が指定桁数に収まらない
正+正、負+負 の場合だけ発生

例: 2進4ビットのとき
  0111 + 0010 = 1001 (7 + 2 = -7)
  1010 + 1100 = 10110 (-6 + -4 = 6)

検出
- 正+正の結果が負になった、負+負の結果が...
- 実際の回路では...

2.3 実数
  小数点以下を持つ数
  数学的な「実数」とは別

2.3.1 固定小数点方式
数値表現の中の小数点の位置を決めてしまう
小数点の右隣の桁は(1/B)の位, その右は(1/(B^2))の位...
整数と同様の表現
  小数点以下4桁の場合
  \sum_{i=-4}^{m-1-4} {N_i B^i}
  小数点以下は桁の重みを B^{-1}, B^{-2}, とする
数値への変換

  定義どおり
数値からの変換
  1以上の部分は整数と同じ
  v < 1.0 として
  // 小数点の右から順次下位へ桁の値を決めていく
  くりかえし{
    v = v * B
    v の整数部分がその桁の値
    v = v - (整数部分)
  }

例:
0011.1100 (2進)
  2+1+(0.5)+(0.25) = 3.75 (10進)
0.1(10進)
  0.0001100110011....(2進)

2.3.2 浮動小数点方式 (floating point number)
固定小数点方式の問題点
  表現できる範囲が狭い
  大きい値は範囲外
  小さい値は有効数字が減少
そこで、有効数字部分と桁位置部分を分離する
  0.602×10^{24} みたいに

仮数(mantissa) 0.602
指数(exponent) 24
仮数と指数のペアで数値を表現
  値 = 仮数×B^{指数}
ほかに符号ビットSを別につける

指数にしたがって小数点の位置が変化
  「浮動」小数点 floating point number

仮数のビット数で有効桁数がきまる
指数のビット数で表現可能範囲がきまる

IEEE標準規格の2進浮動小数点数表記
二通りの長さ(ビット数)
単精度(float - C言語での型名) 32ビット
倍精度(double) 64ビット

2.3.3.1 単精度（float）
  32ビット S:1 E:8 M:23 (プリント(1)図3.1)
  S,E,Mは符号、指数、仮数のビット数

値 = (-1)^{S} × 1.M × 2^{E-127}

仮数M 先頭は二進数で必ず 1 だから
  その1はMに入れない(けち表現)
  これが「1.M」のココロ

指数E 127のゲタ の意味
  浮動小数点数のビット列を(符号付き)整数とみなした時,
  その大小が実数値の大小と一致するようになる
    (値そのものは当然異なるけど)
  整数比較命令で比較可能になる

表現可能範囲
  E は 8bitだから、
  255 >= E >= 0 (符号なしと見ると)
指数(E-127)として使うのは、そのうち
  127 >= 指数 >= -126
  指数 -127, 128は特別な表現に使用 (表3.3)

有効数字
  23+1 = 24ビット (けち表現で1ビット稼いだ)
  (log2/log10)*24 = 7.22

10進で約7桁

正の最大数
S=0, E=254, M=111(中略)111
= 1.111(中略)111 × 2^{127}
~ 2 × 2^{127}
= 3.40 × 10^38

正の最小数
S=0, E=1, M=0
= 1.0 × 2^{-126}
= 1.17 × 10^-38
(これは正規化数の場合; じつはE=0の不正規化数もあって, 以下略)

2.3.3.2 倍精度（double）
64ビット S:1 E:11 M:52
S,E,Mは符号、指数、仮数のビット数

値 = (-1)^{S} × 1.M × 2^{E-1023}

有効数字
52+1 = 53ビット
(log2/log10)*53 = 15.95
10進で約16桁

正の最大数
S=0, E=2046, M=111(中略)111
= 1.111(中略)111 × 2^{1023}
~ 2 × 2^{1023}
= 1.80 × 10^308

正の最小数
S=0, E=1, M=0
= 1.0 × 2^{-1022}
= 2.22 × 10^-308