コンピュータの仕組み
(8)

柴山　潔

コンピュータの仕組み

1　コンピュータシステム
2　ハードウェア
3　内部装置
4　プロセッサ(5)
5　メモリ
6　外部装置
7　論理回路
8　オペレーティングシステム

4 プロセッサ(5)

4.1 プロセッサの動作とその制御

4.2 プロセッサの機能 ―演算と制御―

4.3 演算器(2)

4.4 順序制御機構

負数の2進数表現の必要性

- コンピュータ内部での2進数表現（内部表現）における問題は「『符号』をどう表現するのか？」
  - 人間 → 符号"＋","－" を数の先頭に書くだけ
  - コンピュータ → “＋”も“－”も“0”か“1”で表現する必要
- 特に，「負数」
  - 負数の2進数での表現方法 → 演算器のハードウェア構成方式に大きく影響

負整数の2進数表現

- 負整数をコンピュータ内部で2進数表現するときに考慮しなければならない事項

➢"－"符号も"0"か"1"で表す

← (人間が使用する)符号－絶対値表現(例:"－7")は不可

➢正整数の加算(器)によって負整数の加算(減算)も可能

← (人間が使用する)符号判定による加算/減算の選択(例:"6－7"は"＋6"から"＋7"を引く減算)は遅い

負整数の補数表現

- 2進数表現のnビットの正整数Nについて，それを符号反転した負整数を2進数表現

■ 1の補数　$\overline{N}$

- 正整数Nの各ビットを反転("0"⇔"1")して得る2進数表現

■ 2の補数　$\overline{\overline{N}}$

- 正整数Nの1の補数に"＋1"を加算することによって補正

1の補数 $\overline{N}$

- 正整数Nのn個の各ビットを反転("0"⇔"1")して得る負整数の2進数表現
  - 符号は最上位ビット，"0"ならば正，"1"ならば負

  - N, $\overline{N}$を10進正整数と見なした式4.5の計算

$$\overline{N} = 2^n - 1 - N \quad (4.5)$$

例題4.5：
「符号も含めて4ビットの2進正整数$N=(0111)_2$の符号反転である」負整数を1の補数で表現

［解答］

（ただし，符号は最上位ビットで，“0”ならば正，“1”ならば負）

$$N=(0111)_2\left[=(+7)_{10}\right]$$

$$\Downarrow$$

$$\overline{N}=\left(\overline{0111}\right)_2=(\underline{1000})_2\left[=(-7)_{10}\right]$$

（参考）

1の補数の表現範囲

- k桁（ビット）2進数mの表現範囲（負数は1の補数表現）は，

  $-(2^{(k-1)}-1)$ ～ $2^{(k-1)}-1$ の (2^k-1) 個

  - ″0″と″−0″（←無意味，不要，無駄）が存在

（例） 3ビット（右図）：表現範囲は −3～+3
（$2^3-1=7$個）

3ビットの場合

10進数	2進数
+3	011
+2	010
+1	001
0	000
−0	111
−1	110
−2	101
−3	100

（再掲）

負整数の補数表現

- 2進数表現のnビットの正整数Mについて，それを符号反転した負整数を2進数表現

■ 1の補数　$\overline{N}$

- 正整数Nの各ビットを反転("0"⇔"1")して得る2進数表現

■ 2の補数　$\overline{\overline{N}}$

- 正整数Nの1の補数に"＋1"を加算することによって補正

2の補数 $\overline{\overline{N}}$

- 正整数Nのn個の各ビットを反転("0"⇔"1")することによって得る1の補数に"＋1"を加算することによって補正して得る負整数の2進数表現
  - 符号は最上位ビット，"0"ならば正，"1"ならば負

- N, $\overline{N}$, $\overline{\overline{N}}$を10進正整数と見なした式4.6の計算

$$\overline{\overline{N}} = 2^n - N = \overline{N} + 1 \quad (4.6)$$

例題4.6：
「符号も含めて4ビットの2進正整数$N=(0111)_2$の符号反転である」負整数を2の補数で表現

［解答］

（ただし，符号は最上位ビットで，“0”ならば正，“1”ならば負）

$$N=(0111)_2\left[=(+7)_{10}\right]$$

$$\Downarrow$$

$$\overline{\overline{N}}=\left(\overline{\overline{0111}}\right)_2=\overline{N}+(0001)_2=\left(\overline{0111}\right)_2+(0001)_2$$

$$=(1000)_2+(0001)_2=(\underline{1001})_2\left[=(-7)_{10}\right]$$

（参考）

2の補数の表現範囲

- k桁（ビット）2進数mの表現範囲（負数は2の補数表現）は，

$$-2^{(k-1)} \sim 2^{(k-1)}-1 \text{ の } 2^k \text{ 個}$$

（例） 3ビット（右図）：表現範囲は −4～+3（$2^3=8$個）

3ビットの場合

10進数	2進数
+3	011
+2	010
+1	001
0	000
−1	111
−2	110
−3	101
−4	100

補数表現の符号ビット

＝最上位ビット("0"→"＋", "1"→"－")

- 符号ビットも，下位ビット（列）と同様に，連結して，「（2進）数値」を表すビット → 加算可

補数による加算

- 負数を補数表現
  - ➢ 加算（器）だけで加減算を行うことが可能
  - ➢ 1または2の補数を使う加算（減算）機構：オーバフローの扱いや最上位ビットからの桁上げによる結果の補正の必要性の2点で相違
- ■ 補数器：符号反転を行う演算器
  - ◆ $A-B$を行う減算器
    ＝ 補数器（減数Bの補数をとる，符号反転演算）＋ 加算器（被減数Aと加算）

例題4.7(1):
“－7”を1の補数（4ビット）によって表現，
減算(3－7)を1の補数を使う加算で

［解答］

$$3-7=3+(-7)=-4$$

$$(+3)_{10}=(0011)_2 \quad (+7)_{10}=(0111)_2 \quad (+4)_{10}=(0100)_2$$

$$(-7)_{10}=(\overline{0111})_2=(1000)_2$$

$$3+(-7)=(0011)_2+(1000)_2=(1011)_2\left[=(\overline{0100})_2=(-4)_{10}\right]$$

乗算器と除算器

- 乗算も除算も加算あるいは減算をくり返すことによって可 →「演算に非常に時間がかかったり，その時間がまちまちであったりする」ことを回避

↓

- 現代のコンピュータでは，乗除算を専門に行う演算器をハードウェア機構として装備
- 乗算と除算とは計算の仕組みが根本的に異なるので，乗算器と除算器とは別々に装備

内部装置での実数の表現

- 限りなくある実数はどう扱う？
- 実数を表現する際に使う「小数点」は，コンピュータ（の演算器）では，どのようにして表現？

実数と浮動小数点数表現

- 有限ビットの実数Rは式(4.7)で2進数表現可

$$R = m \times 2^e \quad (4.7)$$

  - mは仮数(部)，eは指数(部)，どちらも2進整数
    - 仮数m：有効数字
    - 指数e：仮数 mに対して小数点の位置決め
  - m も e もいずれも2進整数，負数であれば(m の符号は R の符号)補数表現

浮動小数点数表現の正規化

- ある数の浮動小数点数表現は，mとeの設定の組み合わせで決まる小数点の位置（＝仮数m）の違いで，無数に存在

（例） $(9)_{10} = (1001)_2$

$$R = m \times 2^e \quad (4.7)$$

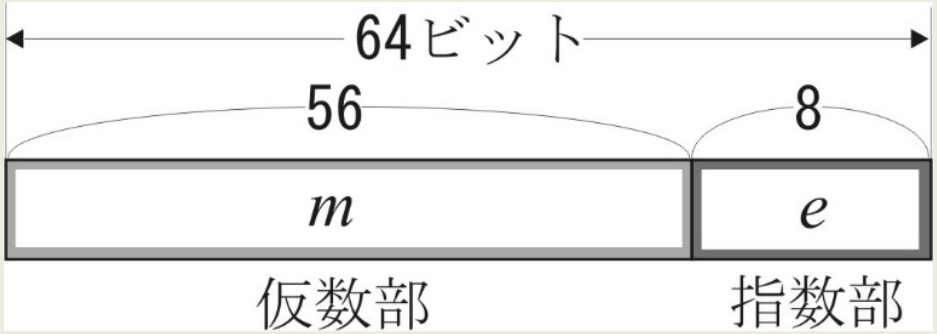
1001×2^0　100.1×2^1　10.01×2^2　0.001001×2^6 …

0.1001×2^4　10010×2^{-1}　1001000×2^{-3} …

- 正規化：浮動小数点数表現を，小数点の位置（＝仮数m）をあらかじめ固定（例：整数，純小数など）することによって，mとeの設定の組み合わせを1個（1表現形式）に統一

浮動小数点数表現

- コンピュータ内部で，実数を定長（例：64ビット）の浮動小数点数で表現
  - （例）仮数部：56ビット，指数部：8ビット
- 実数 R を整数 m と e とによって表現
  - m を整数にするように，整数 e によって調整 ＝ 正規化
  - 指数 e によって小数点位置を調整



例題4.8:
2進実数の$(10100)_2$ $(11.011)_2$ $(0.1111)_2$ を浮動小数点数表現

［解答］

◆この解答では，仮数と指数を2進数で，それらの負数は「絶対値に符号"ー"を付ける」表現 → 実際には，負の仮数や指数は補数表現

$$(10100)_2 = (101)_2 \times 2^{(10)_2}$$

$$(11.011)_2 = (11011)_2 \times 2^{(-11)_2}$$

$$(0.1111)_2 = (1111)_2 \times 2^{(-100)_2}$$

浮動小数点数演算器

- 指数部に対する演算器と仮数部に対する演算器とを別々に用意
  - ◆（例）浮動小数点数 $R_1 = m_1 \times 2^{e_1}$，　$R_2 = m_2 \times 2^{e_2}$ に対する乗算器
    (1) 仮数部の乗算器
    (2) 指数部の加算器
    (3) 正規化：演算結果の仮数部を有効ビットだけで表わすように，指数部によって調整

$$R_1 \times R_2 = \underline{(m_1 \times m_2)} \times 2^{\underline{(e_1+e_2)}} \quad (4.8)$$

論理演算器

- 論理演算：ビット独立の（ビットごとの，2進数1桁の）演算そのもの
  - コンピュータの論理演算器は算術演算器と比べるとずっと簡単に実現
- 2進数はハードウェアの言葉
  - 論理演算（否定，論理積，論理和の3種類）は各1個の論理素子（＝ハードウェアの基本単位，否定/論理積/論理和のどれか）で直接実現
    - （例）16ビットの論理積（アンド）を求める論理演算器：16個の論理素子（＝アンド素子）を連結して並べるだけ

4 プロセッサ(5)

4.1 プロセッサの動作とその制御

4.2 プロセッサの機能 ―演算と制御―

4.3 演算器

4.4 順序制御機構(1)

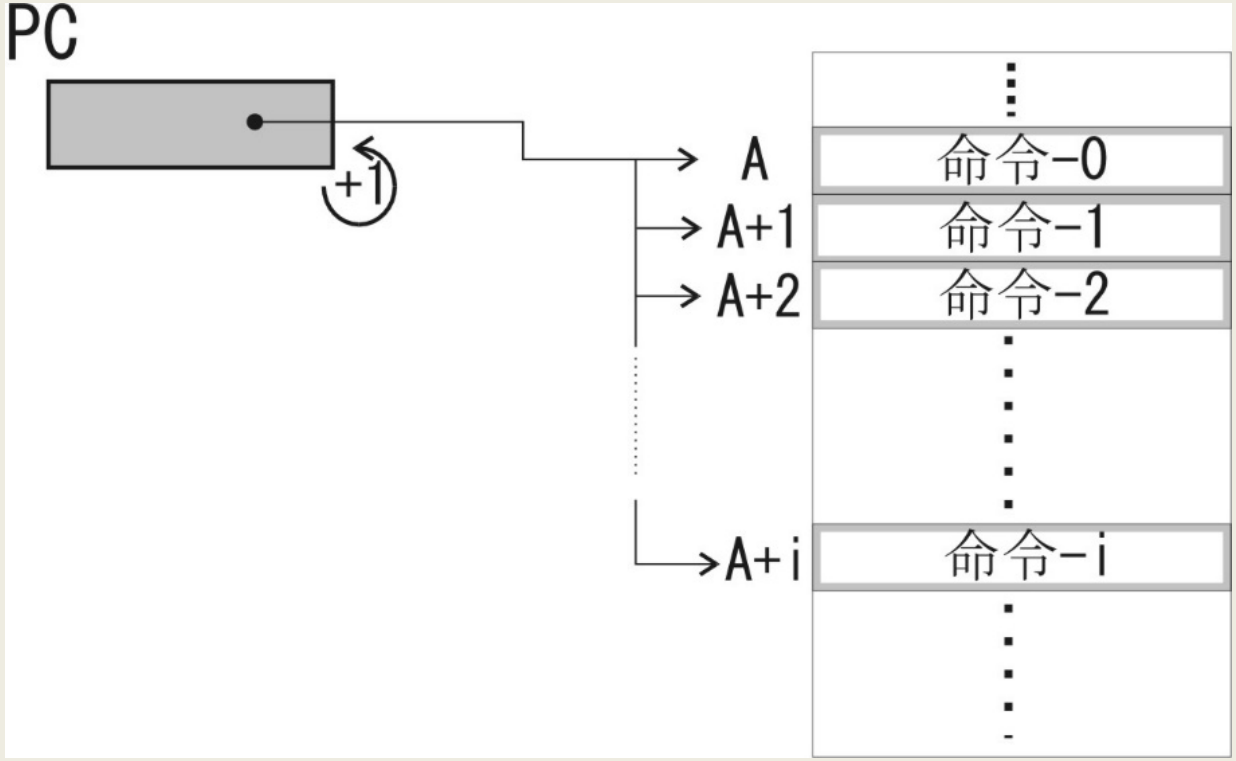
順序制御と順序制御機構

- 順序制御：実行するマシン命令アドレスを決定する操作，「マシン命令の実行順序を決める」こと
- 順序制御機構：プロセッサの中で順序制御を行うハードウェア機構

プログラムカウンタによる順序制御

- ひとまとまりのあるいは一連のマシン命令列（マシン語プログラム）は，原則として，並び（格納）順で実行
  - プログラムカウンタ(PC)を"1"だけカウントアップ
  - 命令取り出しステージで，自然に並び順でマシン命令がメインメモリからプロセッサ内に読み出し
  - 次に実行するマシン命令アドレスの設定のために，実行中の命令実行サイクル中のどこかのステージでPCをカウントアップ

プログラムカウンタによる順序制御 ―並び順―（図）



順序制御機構による順序制御

- 順序制御：PCによってマシン命令の実行順序の管理と制御
- 順序制御機構：PCをはじめとする，順序制御を担当するハードウェア機構

➢ 順序制御機構はプロセッサの代表的な制御機構

（参考）

プログラミングにおける具体的な順序制御例(1)

1. プログラムの書き（実行）順（→メインメモリでの並び/格納順）にしたがって実行（＝順序制御なし）
2. メインメモリのあちらこちらに格納してあるプログラムを連結（連続）して実行
3. 「どのプログラムを実行するか？」を選択
4. プログラムの一部分/全体をやり直す，あるプログラムの開始点に戻る

（参考）

プログラミングにおける具体的な順序制御例(2)

5. 何回も同じプログラムをくり返し実行（くり返し）
6. プログラムの実行途中でいったん別のプログラムを呼ぶ，そこから戻る
7. あるプログラムを複数のコンピュータや人間によって共用
8. あらかじめ予測できないこと（不測の事態）に対処

（重要）

計算器からコンピュータへの進化

- 順序制御機構の装備によって，マシン命令の多彩な（＝様々な環境や条件に応じて処理内容を変える）実行順序の制御が可能に

↓

➢「あらかじめ与えた順序で計算する単なる計算器」→「プログラム（ソフトウェア）によって，種々の状況に適した実行順序での情報処理が可能なコンピュータ」に進化