■**目次**（**ほげ**が今回の範囲）



第 2 回の授業では，0 と 1 のみで数を表す方法を学び，第 3 回の授業では，0 と 1 のみを扱う回路で 2 つの数の加算ができることを学んだ．しかし，これまで扱ってきた 2 進数による数の表現は，0 以上の数のみを対象としていた．また，論理回路の話も，加算に限定されていた．今回は，2 進数による負の数の表現法と，減算を行う論理回路を作る方法について学ぶ．

# ★ 3　負の数の表現と減算の仕組み

## ★ 3.1　負数を表現したい，減算を実現したい

$$7 - 2 = 7 + (-2) = 5$$

という式が成り立つことから分かるように，10 進数の世界で我々が通常考えている減算は，「引く数の符号を反転させた数を作る」→「それを加算する」という手順（「2 を引く」かわりに「-2 を加える」）でも実行できる．2 進数でも同様のことができれば，加算器を使って減算を実現できそうである．しかし，上記では負の数を表すために特別な記号「−」（**負号** (☆ 1)）を使っている．なるべく簡単なルールの方がうれしい（その方が単純な論理回路で作れそうだから）ので，「負号を使わずに負数を表す」方法を考えてみよう．

☆ 1) どちらも「ふごう」だけれど意味が違う．符号は'+' も含むが，負号は含まない．また，どちらも大金持ちではない．

### ★ 3.1.1　まずは 10 進数で負号を使わない方法を考えよう

まずは 10 進数で考える．ただし，桁数は固定とし，例として 4 桁で考えてみる．$123 + (-123) = 0$ なんだから，右図 (ア) の箱の中に 0 から 9 の数を入れて計算が成立するようにできるなら，その箱の中の数の並びが $-123$ を表すと考えてもよいだろう．しかし，我々の知っている加算のルールでは，これは実現不可能である．

そこで，右図 (イ) のように，元の数の最上位の桁からその上の桁への桁上げが 1 になることを許す（例では 4 桁目から 5 桁目への桁上げが 1 になっているが，これを無視して考える）ことにしてみたらどうだろうか．

(ア)

| | | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| + | □ | □ | □ | □ |
| 0 | 0 | 0 | 0 | 0 |

(イ)

| | | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| + | □ | □ | □ | □ |
| 1 | 0 | 0 | 0 | 0 |

**Q1.** 上図 (イ) の式が成り立つような数の並びを求めなさい．さらに，足される数（上の行の数）が 10 進数 4 桁の数 0001 および 0002 の場合についても同様のものを求めなさい．

**Q2.** Q1 で得られた $-123, -1, -2$ に対応する数の並びを用いて，$200-123$ つまり $200+(-123)$ 等の計算を行い，正しい結果が得られることを確認しなさい．ただし上述のように，加算の際には 4 桁目からの桁上げを無視すること．

このように，4 桁の 10 進数の場合，「ある数と加算すると 5 桁の数 10000 になるような数」を考えると，4 桁目からの桁上げを無視するという約束のもとで，その数を元の数の符号を反転させたものと考えられそうである．

### ★ 3.1.2 次は 2 進数で

同様のことが 2 進数でもできるか調べてみよう．例として，4 ビットの 2 進数を考える．ただし，上記と同様に，最上位の桁上げを無視するという約束にしよう．

(ウ)

| | | | | |
|---|---|---|---|---|
| | 0 | 0 | 0 | 1 |
| + | □ | □ | □ | □ |
| 1 | 0 | 0 | 0 | 0 |

**Q3.** 上図 (ウ) の式が成り立つようなビットパターンを求めなさい．さらに，足される数（上の行の数）が 0010 と 0011 の場合にも同様のものを求めなさい．

4 ビットの 2 進数 $0001, 0010, 0010$ は，10 進数ではそれぞれ $1, 2, 3$ である．したがって，Q3 の答えは，これら 3 つの数の符号を反転させて作った負の数，つまり 10 進数の $-1, -2, -3$ に対応するビットパターンとみなすことができそうである．

**Q4.** Q3 で得られた $-2$ に対応するビットパターン用いて，4 ビットの 2 進数で $7-2$ つまり $7+(-2)$ の計算を行い，正しい結果 5 が得られることを確認しなさい．ただし上述のように，加算の際には最上位の桁からの桁上げを無視すること．

### ★ 3.1.3 一般化しよう

上記の考え方をもとにして，任意の 2 進数に対して，その符号を反転させた数に対応するビットパターンを作ることができるのだろうか．

**Q5.** 4 ビットの 2 進数を一つ適当に作り，その各ビットの 0/1 を反転させたビットパターンも作りなさい．両者を加算すると和はいくつになるか．他の数や 4 ビット以外の場合でもやってみなさい．

Q5 の答えを眺めると，例えば 4 ビットの場合，得られた和は常に 1111 であり，これにさらに 1 を加えると 10000 という 5 ビットの数が得られることがわかる．したがって，★★ (ここが肝心の部分．講義時に解説します) ★★

## ★ 3.2　2 の補数

★ 3.1.3 で説明しているようにして負の数を表す方法を，(2 進法表記における) **2 の補数**による表現という (☆ 2)．コンピュータで負の数を表現し，加算器で減算を実行できるようにするために広く用いられている．

4 ビットで 2 の補数による表現を用いると，10 進数の $-7$ に対応するのは 1001 ということになる．しかし，0 以上の整数のみ考えていたこれまでの流儀では，このビットパターンは 9 を表している．このように，同じビットパターンでも，0 以上の整数のみを扱う表現と 2 の補数による表現では異なる数を表すことになる場合がある．$n$ ビットで 2 の補数による表現を用いる場合，下表に示すように，$2^n$ 通りの数のうち半分を 0 以上の数に，残り半分を負の数に割り当てる．

☆ 2) 2 の補数は，元の数と足し合わせた時に最上位からの桁上げが 1 になり，他のビットが全て 0 になる数である．一方，元の数と足し合わせると全てのビットが 1 になるような数を **1 の補数**という．Q5 で示したように，元の数の各ビットを反転させると 1 の補数が得られる．

● 4 ビットを全て正の整数の表現に用いた場合

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 8 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 9 | 1 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 10 | 1 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 11 | 1 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 12 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 13 | 1 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 | 14 | 1 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 | 15 | 1 | 1 | 1 | 1 |

● 2 の補数を用いて符号付き数を表現した場合

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | | 1 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | | 1 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | | 1 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | | 1 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 | | 1 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 | | 1 | 1 | 1 | 1 |

**Q6.** 右の表を完成させなさい．

2 の補数の成り立ちと上の表の両方からわかる通り，2 の補数による表現では，最上位のビットが $\boxed{\text{A}}$ ならば 0 以上の数を，$\boxed{\text{B}}$ ならば負の数を表すことになる．また，ビット長が $n$ の場合，表現できる最大の数は， 0 の後に 1 が $n-1$ 個続いたものであるから，10 進数の $\boxed{\text{C}}$ に相当する．一方，表現できる最小の数は，1 の後に 0 が $n-1$ 個続いたものであり，10 進数の $\boxed{\text{D}}$ に相当する．

**Q7.** $\boxed{\text{A}}$ と $\boxed{\text{B}}$ に入る数を答えなさい．また，$\boxed{\text{C}}$ と $\boxed{\text{D}}$ を $n$ の式で表しなさい．

ところで，上記の表において，0 と $-8$ 以外の数については，符号を反転した数とペアを組んでおり，互いに相手の 2 の補数となっている．しかし，0 と $-8$ にはそのような相手がおらず，自分自身が 2 の補数となっている．

## ★ 3.3　減算を行う論理回路

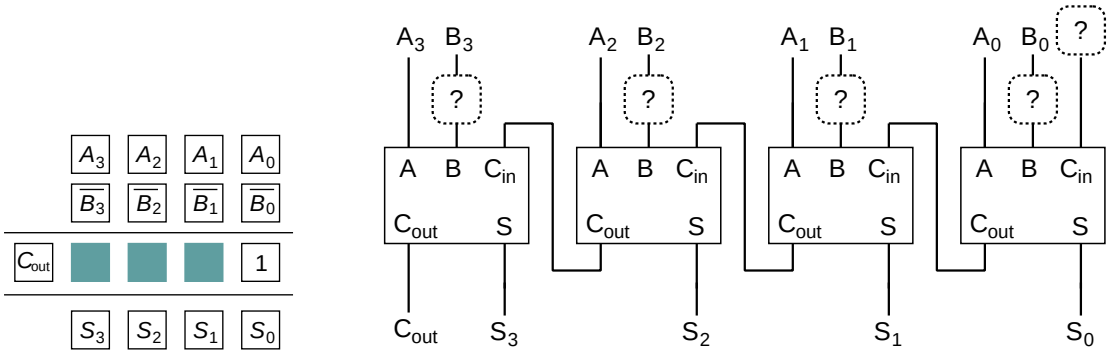
2 の補数表現を利用すると，減算という演算は「引く数の 2 の補数を求め，それを加算する」という計算で実行できる．この計算手順を論理回路で実現するには，次のような手順を踏むことになる．

1. 引く数の 2 の補数を作る．この手順はさらに次の 2 つに分けられる
   (a) 引く数のビットを反転する
   (b) 得られたものに 1 を加える
2. 加算器に 2 つの数を入力して和を求める

しかし，1(b) の手順は，加算器の最下位桁への桁上げ入力を 1 にすることで実行できる．したがって，次の手順で済ませることができる．

1. 引く数のビットを反転する
2. 加算器に 2 つの数を入力して和を求める．ただし，最下位桁への桁上げ入力を 1 にしておく

下図右は，2 の補数表現を利用して，4 ビットの 2 進数 $A_3A_2A_1A_0$ から $B_3B_2B_1B_0$ を引いた差 $S_3S_2S_1S_0$ を計算する回路である．上述のように，最下位桁への桁上げ入力 $C_{\rm in}$ は 1 にしておく．また，最上位桁からの桁上げ $C_{\rm out}$ は，無視することになる．




**Q8.** 上図右の回路図の ? の部分を埋めて減算回路を完成させなさい．