

本節の授業のテーマ

本日の授業のテーマは、以下のとおりです。

- (1) 負の数の表現
 - 2 の補数

本日の授業のゴールは、以下のとおり。

- 負の数の表現方法がわかること。

1 負の数の表現

自然数の表現は非常に簡単でしたが、負の数を含んだ整数となると、事情が異なります。それごと、約束を作れば良いのですから、それこそ無限の表現方法があるでしょう。無限というのは言い過ぎで、情報科学の分野では、無限を考えてはいけませんから、N ビットだけで表現するならば、 2^N よりも、少し少ない程度の表現方法があると思います。皆さん、どれだけの表現方法があるか、考えてください。

数多くの表現方法がある中で、われわれが使うのは、

- 便利であること
- 表現が簡単であること
- ハードウェアの実現が簡単であること

などを基準にして、表現方法を選択します。通常は、最期に紹介する2の補数表現を使いますが、ほかの表現方法も紹介しておきます。

COMET II では整数は16ビットで表現されます。そのため、これ以降、16ビットで話を進めます。

1.1 絶対値表示

符号のためにビットを1つ設けて、あとは絶対値で表現する方法です。たとえば、図?のような表現です。

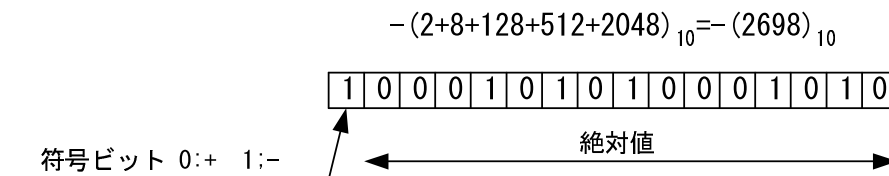


図1 負の表現(絶対値表示)

1.2.1 の補数表示

マイナスの数のビットを反転する方法です。したがって、15ビットが符号を表すことになります。この方法の欠点は、000000000000000と111111111111111がともにゼロを表すことになります。

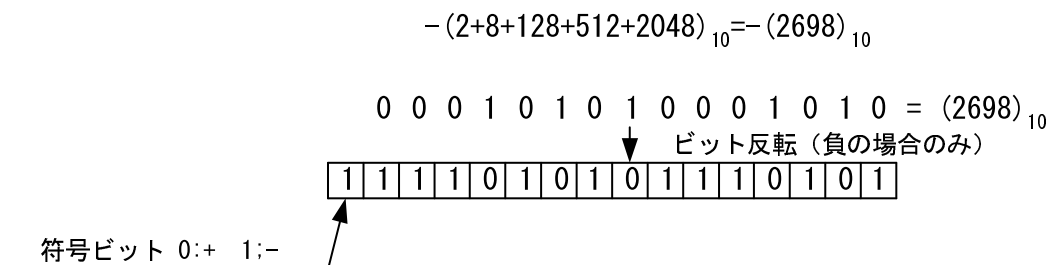


図2 負の表現(1の補数表示)

1.3.2 の補数表示

これが教科書にかかっている方法です。先に示した1の補数表示に1を加えた表示方法です。表現の方法は、図?の通りです。

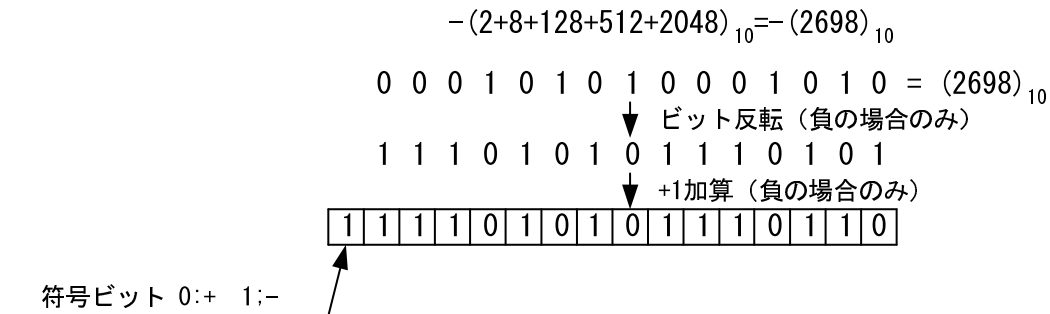


図2 負の表現(2の補数表示)

この方法のメリットは、減算が加算器でできることです。2の補数表現のイメージは、図の通りです。車の距離計に似ています。

表現している数	計算機の内部表現(2進数)	(16進数)
4	000000000000000100	0004
3	000000000000000011	0003
2	000000000000000010	0002
1	000000000000000001	0001
0	000000000000000000	0000
-1	111111111111111111	FFFF
-2	111111111111111110	FFFE
-3	111111111111111101	FFFD
-4	111111111111111100	FFFC

図3 距離計イメージ(2の補数表示)

それでは、なぜ、2の補数表現だと、減算が加算器で可能なのでしょうか。減算の演算は、負の数の加算と同じです。したがって、図3のように負の数を表現すると、負の整数の加算は正の整数の加算と同じと分かるでしょう。したがって、加算器で減算が可能なのです。実際、正の数の減算を行うときは、①ビットの反転②+1加算を実施して、加算器で計算します。

イメージは、図3の通りですが、もう少し、理論的に説明をしましょう。ある正の整数を x としましょう。その負の数、 $-x$ は2の補数表現では、

$$[-x] = (FFFF - x + 1)_{16} \quad (1)$$

となります。左辺の $[-x]$ が $-x$ の意味です。 $[\]$ の意味は、括弧内の負の整数を計算機内部の表現を意味しています。これは、私が作った表記ですので、一般には用いられていません。右辺の $FFFF-x$ がビット反転です。それに1を加えて、2の補数の表現としています。

つぎに、ある整数 y を考えて、 $y-x$ を計算してみましょう。

$$[y-x] = (y + FFFF - x + 1)_{16} \quad (2)$$

$FFFF-x+1$ は、あらかじめ計算されて、コンピューター内部のメモリーに格納されているので、 $[y-x]$ は加算器で可能です。

これは、あたりまえです。重要なことは、この結果が、負の場合、2の補数表現になっており、正の場合、そのままの値になっていることです。最初は、 $y-x$ が負になる場合を考えましょう。すると(1)式は、

$$[y-x] = (FFFF - (x-y) + 1)_{16} \quad (3)$$

と変形できます。この場合、絶対値が $(x-y)$ なので、①絶対値のビット反転②+1加算となっています。つぎに、 $y-x$ が正になる場合を考えましょう。すると(1)式は、

$$\begin{aligned} [y-x] &= (y-x + FFFF + 1)_{16} \\ &= (y-x + 10000)_{16} \end{aligned} \quad (4)$$

となります。10000は計算機内部では、桁上りを示します。16ビットの表示では無視されます。したがって、内部の表現は、正しく表せます。

この方法で負の数を表すことは、1970年頃には常識となったようです。単に、補数といえば、2の補数を表します。驚いたことに、負の数をこの補数で表すアイデアは、パスカルが最初です。パスカルは、パスカリーノという歯車式計算機を1964年に製作しています。その減算を加算器で行うために、補数というものを考えたようです。

1.4 どの方法が有利か、

絶対値表示で負の数を表現すると減算器が必要です。場合によっては、比較器も必要かも知れません。1の補数表示と2の補数表示は似ていますが、0が2通りで表示する1の補数表示は良くないように思えます。2の補数表示は、減算をする場合、加算器が使えます。ただし、ビットの操作が必要ですが・・・。

絶対値表示と2の補数表示の差は、減算の場合、減算器を使うかビット変換器を使うかのさです。じつは、ビット操作の法がずっと簡単です。そのような理由から、2の補数表示が使われるようになったのです。

2 ビット反転と+1 加算の意味

正の整数 x として、 $-x$ を計算機の内部で表現する場合、

- ① x をビット反転する。
- ② +1 加算する。

の操作を行いました。式で表すと、

$$[-x] = (FFFF - x + 1)_{16} \quad (5)$$

でしたね。ここでは、この操作の意味を調べます。結論から言うと、この操作は、符号反転の操作(-1 乗算)です。

それでは、もう一回この操作を繰り返しましょう。すると

$$\begin{aligned} (FFFF - (FFFF - x + 1) + 1)_{16} &= (x)_{16} \\ &= [x] \end{aligned} \quad (6)$$

となります。このことから、この操作は、符号反転であることが分かります。(5)式は、 x の符号反転を示しており、(6)式は $-x$ の符号反転を示しています。

$-x$ のコンピューター内部表現を求める式は、(5)です。したがって、元の x を求めるためには、(5)の反対の演算をすればよく、

- ① 1 減算(-1 加算)する。
- ② ビットを反転する。

となります。式で表すと、

$$\begin{aligned} (FFFF - (FFFF - x + 1 - 1))_{16} \\ &= (x)_{16} \\ &= [x] \end{aligned} \quad (7)$$

です。

しかし、元の表現を得るためには、(6)式の演算でも良いはずですが。(7)式を変形すると(6)式が容易に導くことができます。

これらのことから、以下の結論を導くことができます。

- ・①符号の反転②+1 加算の操作は、コンピューターの内部表現の符号反転である。
- ・この符号の反転と反対の操作①1 減算②符号反転は、まったく同じ操作である。

3 練習問題(2の補数)

COMET II の扱う、16 ビットで、以下の計算を実施してみよう。

(1) 次の負の 10 進数を補数で表現してみましょう。

例	$(-14)_{10}$	0000000000001110	←	14 の 2 進数表現
		1111111111110001	←	ビット反転
		1111111111110010	←	+1 加算

Q1 $(-18)_{10}$

(2) 次の 2 の補数表現を 10 進数に変換してみましょう。

例	1111111111101011		
	1111111111101010	←	1 減算
	0000000000010101	←	ビット反転
	16+4+1=21	←	10 進数に変換
	$-(21)_{10}$		

例の別解法	1111111111101011		
	0000000000010100	←	ビット反転
	0000000000010101	←	+1 加算
	16+4+1=21	←	10 進数に変換
	$-(21)_{10}$		

Q1 11111111111001011

(3) 次の計算を、10 進数の演算を、2 の補数を使って計算してみましょう。

Q1 $21-14$

Q2 $14-21$