COMET IIのメモリーと負の整数の表現

山本昌志*

2005年11月4日

1　本日の学習内容

本日の講義内容は以下の通りで，教科書のp.9〜12に対応する．

- COMET IIの取り扱う数値
  - 情報の単位
  - COMET IIのメモリー
  - 整数表現とその最大値と最小値
  - Pentiumのメモリー
- 負の数の表現方法
  - 符号ビットを用いる方法 (絶対値表示)
  - 2の補数

2　コメットIIのメモリー

2.1　情報の単位

情報の単位はビット (bit[1]) と言う．1ビットは，”0”か”1”か，”yes”か”no”かのように，2種類の情報を分けることができる[2]．”yes”か”no”とかのように表現すると後の計算が面倒なので，今後はすべて”0”か”1”の記号を用いることにする．ようするに，”0”か”1”のいずれかが生じる事象[3]で，”0”または”1”を得たとき，1ビットの情報を得たという．

それでは，2ビットとはどういう情報であろうか?．それは，”00”か”01”か”10”か”11”のように4つの事象があるとき，それらのいずれかを受け取ったとき，2ビットの情報を受け取ったことになる．同様に，”0

*独立行政法人　秋田工業高等専門学校　電気工学科

[1] binary unitの略

[2] 2つの場合に分けることができる情報があると言った方がよいかも．

[3] 正確に言うと，”0”と”1”は同じ確率で生じる必要がある．ここでは，そんなことは気にしないことにする．

と 1 が n 個”あるような事象がある時に，それらの一つの事象を知らされた場合，n ビットの情報を受け取ったことになる．

これまでの話から，2 進数の桁数がビット数になることが分かる．n 桁の 2 進数は n ビットの情報を表すことが可能で，それは 2^n 個の事象を分けることができる．k 個の事象を分けるために x ビット必要となると，

$$x = \log_2 k \tag{1}$$

の関係がある．

[**練習問題**]　秋田高専の学生数を 800 人とすると，一人一人を区別するために，必要な情報量をビット数で答えよ．さらに，2 進数で表現すると何桁必要か?.

[**解答**]　ビット数は，

$$\log_2 800 = 9.64\cdots$$

である．2 進数で表現すると，10 桁必要となる．

- 情報の単位は，ビットが用いられる．それは，”0”か”1”の 2 つの事象の 1 つの状態を表す．
- 2 進数の桁数がビット数である．
- 16 進数の 1 桁は 2 進数の 4 桁なので，16 進数の 1 桁は 4 ビット，2 桁は 8 ビットである．
- k 個の事象があるとき，それの一つの情報を得るためには，$\log_2 k$ ビット必要である．

2.2　COMET II のメモリー

アセンブラ言語 CASL II が動作するコンピューターを COMET II という．これには CPU とメモリーがある．第 1 回の講義で述べたように，メモリーにはアドレスとデータの内容がある．この COMET II のメインメモリー (主記憶装置) は，図 1 に示すとおり，アドレス 16 ビット，メモリー 16 ビットとなっている．

- アドレスが 16 ビット，メモリーが 16 ビットになっている．
- アドレス 1 個あたり，1 個のデータがある．

この図を見ても分かるように，2 進数表示は紙面の面積が必要で紙の無駄である．もう少し紙を節約し，更に分かりやすくするために，通常は図 2 の 16 進数表示が使わる．

COMET II では，データは 16 ビット単位で扱われる．この 16 ビットの単位を 1 ワード (1 語) と呼ぶ．この 1 ワードは，教科書に書かれているように，便宜上，上位 8 ビットと下位 8 ビットに分けられる．そうして，最下位のビットから番号がつけられている．最下位のビット番号が 0 で，最上位が 15 である．コンピューターの世界では，整数は 0 から数えることが多いので，それに慣れる必要がある．こうすると便利なことは，整数を表す場合，ビット番号が 2 進数の指数を表す．すなわち，第 0 ビットは 2^0，第 7 ビットは 2^7，第 15 ビットは 2^{15} の桁を表すのである．まことに便利である．

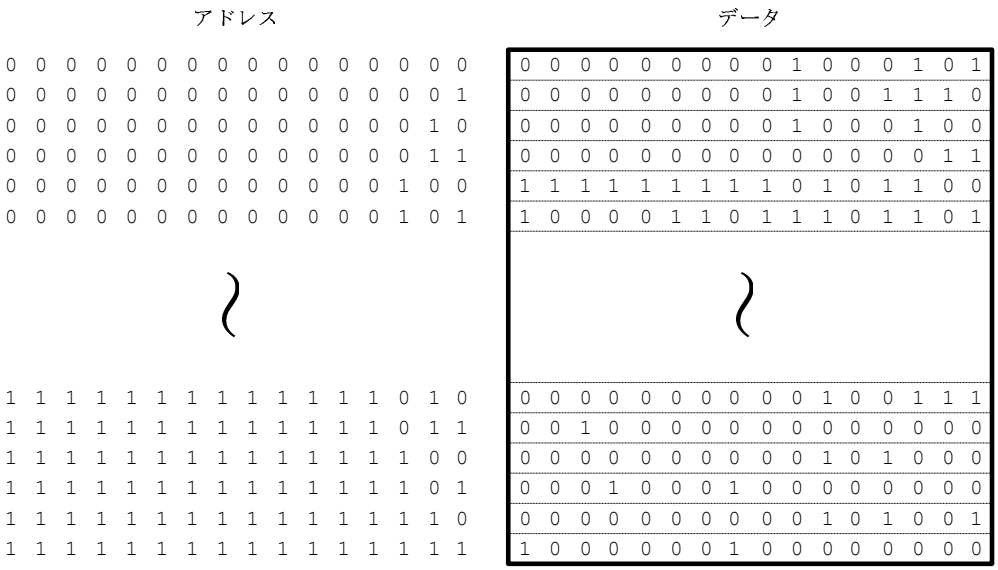

図 1: メモリーのモデル (2 進数表示)

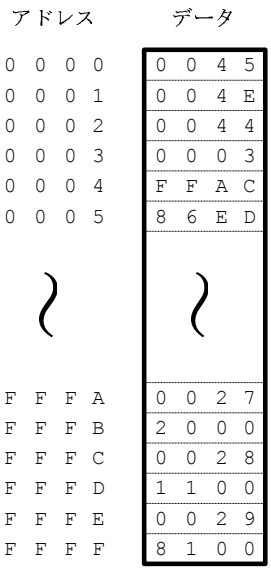

図 2: メモリーのモデル (16 進数表示)

2.3 整数表現とその最大値と最小値

CASL II のプログラムではよほどのことが無い限り，整数しか取り扱わない．ひとつの整数は，メモリーのひとつのアドレスに格納される．符号無し整数の場合，2 進数で表現した整数がそのまま，メモリー上のデータとなる．メモリーに格納された整数の様子を図 3 に示す．ただし，符号付整数か，符号無し整数かの判別は，プログラム次第である．符号付きの整数については，次の章で示す．

その整数の範囲については，教科書の通りである．電卓で自分で確認してみよう．

アドレス	データ	符号付整数	符号無整数
9 F F F	? ? ? ?		
A 0 0 0	0 0 0 A	$(10)_{10}$	$(10)_{10}$
A 0 0 1	0 A 0 B	$(2571)_{10}$	$(2571)_{10}$
A 0 0 2	7 F F F	$(32767)_{10}$	$(32767)_{10}$
A 0 0 3	8 0 0 0	$(-32768)_{10}$	$(32768)_{10}$
A 0 0 4	A B C D	$(-21555)_{10}$	$(43981)_{10}$
A 0 0 5	F F F F	$(-1)_{10}$	$(65535)_{10}$
A 0 0 6	? ? ? ?		

図 3: 整数を格納しているメモリーの状態 (16 進数表示)

2.4 Pentium のメモリー

COMET II のメモリーは先ほど述べたとおりである．皆さんが使っている Intel の Pentium のメモリーはどうなっているのだろうか?．まず，アドレスバスは 32 ビットである．通常のコンピューターのデータは，8 ビット単位で扱われ，その単位を 1 Byte と言う．一つのアドレスに 1 Byte(8 bits) のデータが格納されている．Pentium のアドレスバスは 32 ビットので，それが取り扱うことができるデータ数は，2^{32} である．2^{30} で G(ギガ) [4]なので，4 GBytes のデータを扱うことができる．

また，データバスは 64 bits で，レジスター[5]は 32 bits である．一度に 32 bits のデータを取り扱うことができるから，32bit CPU と呼ばれている．ここで，ひとつ疑問が生じる．1 つのアドレスには 8 bits のデータしか格納されないのに，64 bits のデータを一度に取り扱うのは変と感じる．どうやっているかというと，pentium では一度に 8 個のアドレスのデータを読み書きできるのである．

ひとつのアドレスに，64 bits を格納できるようにすれば，問題がなくなるように考えられる．そうすると他の問題が生じる．今まで，8 bits 単位でデータを取り扱ってきたので，過去のデータやソフトウェアーとの互換性がなくなる可能性がある．

実際のコンピューターで使われるメモリーは，RAM(Random Access Memory) である．1 ビットのデータは，RAM の内部のコンデンサー[6]に電荷が有れば 1 に、無ければ 0 になる。いったい、メモリー 1 枚にどれほどのコンデンサーがあるのだろうか?。

- 近頃の市販の 1 枚の RAM は、512 MBytes である。

[4]コンピューターの世界の補助単位は，2^{10} で k(キロ)，2^{20} で k(メガ)，2^{30} で G(ギガ) となる．
[5]CPU の中の記憶装置
[6]DRAM (Dynamic Random Access Memory) は、1 個のトランジスターとコンデンサーで 1 ビットを蓄える。

- 1 Byte は 8 ビットなので、1 Byte あたり 8 個のコンデンサーがある。
- 512 MBytes=2^{29} Byte である。

以上のことから、512 MBytes の RAM の中に、$2^{32} = 4294967296$ のコンデンサーがある。約 43 億個である。非常に驚かされる。これが、1 個の間違いも無く動く、どうなっているのだろうか?

3 負の整数の表現

コンピューター内部で負の整数を表現する方法はいろいろ考えられる．ここでは，2 通りの方法を示すが，符号ビットを用いる方法は現在では使われていないので，忘れてしまっても良い．諸君は，2 の補数を用いる方法を理解しなくてはならない．

3.1 符号ビットを用いる方法 (絶対値表示)

これまでは，正の整数を 2 進数あるいは 16 進数で表現することを学習した．次に，負の整数を表す方法を学習する．通常の負の数は，

$$(-0100110011101111)_2 = (-4CEF)_{16} = (-19695)_{10}$$

と，マイナスの記号を書く．コンピューターでこれを表現するためには，符号ビットとして，1 ビット用意すれば良い．たとえば，先頭のビットが 1 の場合，それはマイナスを表すとする．例えば，$(-19695)_{10}$ は，コンピューター内部で `1100110011101111` と表すのである．

これは良い方法のように思える．しかし，`1000000000000000` と `0000000000000000` が同じゼロを表すことになる．明らかに不便である．また，詳しくは述べないが，これを使った負の数の演算のためのハードウェアー (CPU) が複雑になり，現代ではこの方法は使われていない．実際には，次に述べる 2 の補数を使って，コンピューター内部では，負の数を表すのである．

3.2 2 の補数

3.2.1 理論的な話

負の整数は，補数 (complement) を使って，コンピューター内部では表現される．それを図 4 に示すが，手順は，次の通りである．

1. 絶対値を 2 進数のビットパターンで表現し，その反転を行う．
2. 反転されたビットパターンに 1 を加算する．

このようにしてできたビットパターンをメモリーに記憶させ，それを負の数として取り扱う．

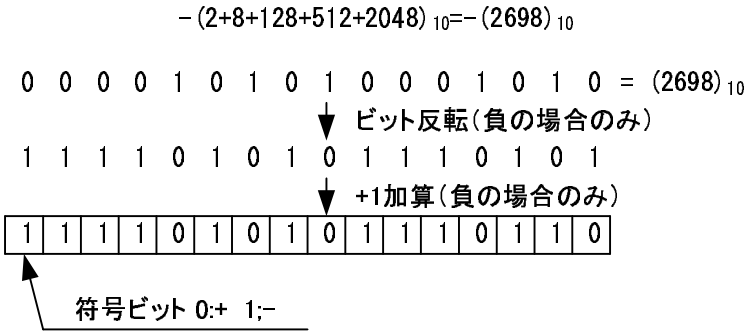

図 4: 負の整数をメモリーに格納する方法

2 の補数表現のイメージは，図 5 の通りである．車の距離計に似ている．

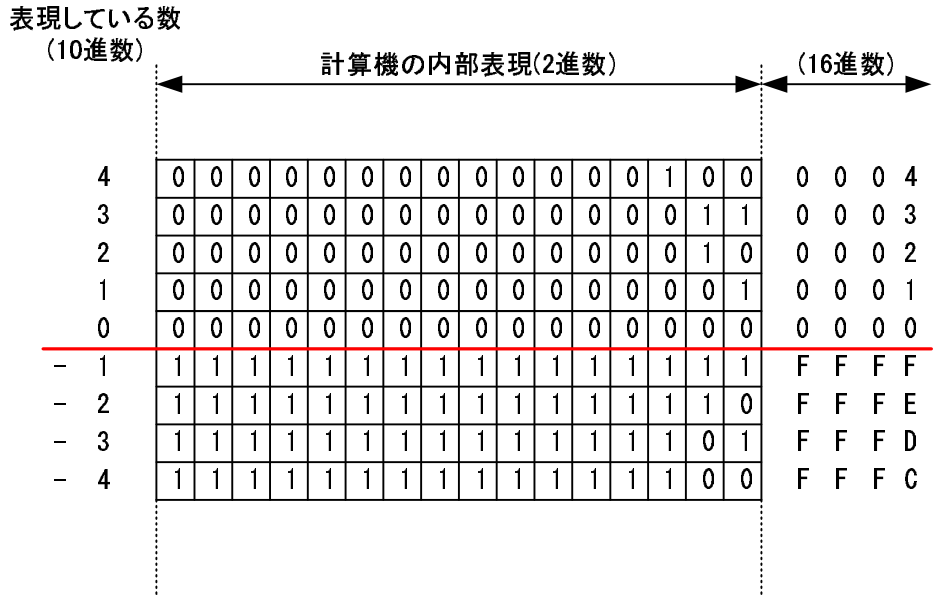

図 5: 距離計イメージ (2 の補数表示)

2 の補数を使うメリットは，減算が加算器で可能なことである．それでは，なぜ，補数表現だと，減算が加算器で可能なのだろうか?．減算の演算は，負の数の加算と同じである．したがって，図 5 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あろう．したがって，加算器で減算が可能となる．実際，正の数の減算を行うときは，ビットの反転と +1 加算を実施して，加算器で計算する．イメージは，図 5 の通りであるが，もう少し，理論的に説明をおこなうとしよう．ある正の整数を x とする．その負の数，$-x$ は補数表現では，

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

となる．左辺の $[-x]$ が $-x$ の意味である．[　] の意味は，括弧内の負の整数を計算機内部の表現を表している．これは，私が作った表記なので，一般には用いられていない．右辺の $FFFF-x$ がビット反転になっている．ここでは，16 ビットで整数を表現しようとしているので，$FFFF$ から x を引いてビット反転させている．疑問に思う者は実際に計算して見よ．それに 1 を加えて，補数の表現としている．つぎに，ある整数 y を考えて，y-x を計算してみよう．

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

$FFFF-x+1$ 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，$[y-x]$ は加算器で可能である．これは，あたりまえである．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ある．

演算の結果，y-x が負になる場合を考えよう．すると式 (3) は，

$$[y-x] = \{FFFF - (x-y) + 1)\}_{16} \tag{4}$$

と変形できる．この場合，絶対値が $(x-y)$ なので，絶対値のビット反転と +1 加算となっていることが理解できる．つぎに，$y-x$ が正になる場合を考えましょう．すると式 (3) は，

$$\begin{aligned}[y-x] &= (y - x + FFFF + 1)_{16} \\ &= (y - x + 10000)_{16}\end{aligned} \tag{5}$$

となる．$(10000)_{16}$ は計算機内部では，桁上がりを示す．16 ビットの表示では無視される．したがって，内部の表現は，正しく表せる．

コーヒーブレイク

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．驚いたことに，負の数をこの補数で表すアイディアは，パスカルが最初です．パスカルは，パスカリーヌという歯車式計算機を 1642 年頃に製作しています．そこ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．

3.2.2 ビット反転と +1 加算の意味

x を正の整数として，$-x$ をコンピューター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．

の操作で得られたものその内部表現になる．式で表すと，

$$[-x] = (FFFF - x + 1)_{16} \tag{6}$$

である．この操作の意味を調べてみよう．結論から言うと，この操作は符号反転 (-1 乗算) の操作になっている．それを示すために，もう一度この操作を繰り返してみる．すると，

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

となる．このことから，この操作は，符号反転であることが理解できる．式 (6) は，x の符号反転を示しており，式 (7) は $-x$ の符号反転を示している．

式 (6) は，-x のコンピューターの内部表現を表している．従って，元の x を求めるためには，その逆の操作

1. 1 減算 (-1 加算) する．
2. ビット反転する．

をすればよく，式だと

$$\begin{aligned}[FFFF - \{(FFFF - x + 1) - 1\}] &= (x)_{16} \\ &= [x] \end{aligned} \tag{8}$$

となる．しかし，元の表現を得るためには，式 (7) の演算でも良いはずである．式 (8) を変形すると，容易に式 (7) を導くことができる．これらのことから，以下の結論を導くことができる．

- ビット反転と +1 加算の操作は，整数のコンピューターの内部での表現の符号反転である．
- この符号反転の反対の操作である 1 減算とビット反転の操作は，ビット反転と 1 加算と同じ操作である．

3.2.3　2 の補数表現と 10 進数の通常の表現の変換

これは，教科書に書いてあるとおり．

4　実数の表現

CASL II では，実数を取り扱わない．そのため，ここでは実数の表現方法について，学習しないことにする．しかし，実際のコンピューターでは実数が使われているので，その方法につては，付録に示しておく．興味のある者は，読んでみるとよい．さらに，実際にプログラムを作成して，調べてみると良い．

5　課題 (レポート)

5.1　情報量

[**問 1**]　ある特定の人間の性別を表す場合に，必要なビット数を答えよ．

[**問 2**]　秋田高専の学生の生年月日を記述したい．一人あたりの必要なビット数を考えよ．

[**問 3**]　640×400 ピクセルのカラー写真の場合，どれくらいのビット数が必要か調べよ．

5.2　符号無し整数

プログラマーが次のように，符号無し整数を格納した．CMOET II のメモリーの内容を図 6 に記述せよ．2 進数と 16 進数で記述すること．

- アドレスの B000 から，符号無しの整数 (10 進数) で $(1, 2, 4, 8, 16, 25, 100, 511, 32768, 65535)$ を格納した．以下のヒントを利用せよ．

  $(32768)_{10} = (2^{15})_{10}$

  $(65535)_{10} = (2^{16} - 1)_{10}$

アドレス (16 進数)	データ (2 進数)																データ (16 進数)
AFFF	1	0	0	1	1	1	1	1	0	1	0	1	1	0	1	0	9F5A
B000																	
B001																	
B002																	
B003																	
B004																	
B005																	
B006																	
B007																	
B008																	
B009																	
B00A																	

図 6: 符号無し整数を格納したときのメモリーの内容

5.3 符号付整数整数

次のように，符号無し整数を格納した．CMOET II のメモリーの内容を図 7 に記述せよ．2 進数と 16 進数で記述すること．

- アドレスの B000 から，符号有りの整数 (10 進数) で (-1, -2, -4, -8, -16, -25, -100, -511, -32768, 517, 32767) を格納した．以下のヒントを利用せよ．
  - 負の数は，2 の補数で表す．2 の補数は次のようにして，求める．
    - 絶対値を 2 進数で表す．
    - その 2 進数をビット反転させる
    - ビット反転させた値に 1 を加算する．

メモリーの内容が出来上がったならば，以下を確認せよ．

- -1 に 1 を加算したら，0 になることを確認せよ．
- 16 ビットのビットパターンを見ると，符号付の 32767 の次は-32768 になることを確認せよ．
- データの内容を符号無しの場合と比較せよ．面白いことに気がつくであろう．
- 負の数の場合，第 15 ビットが 1 になっていることを確認せよ．
- 正の数の場合，第 15 ビットが 0 になっていることを確認せよ．
- 第 15 ビットを見れば，正負が分かるので，これを符号ビットと言う．

アドレス (16 進数)	データ (2 進数)																データ (16 進数)
AFFF	1	0	0	1	1	1	1	1	0	1	0	1	1	0	1	0	9F5A
B000																	
B001																	
B002																	
B003																	
B004																	
B005																	
B006																	
B007																	
B008																	
B009																	
B00A																	

図 7: 符号付整数を格納したときのメモリーの内容

5.4 レポート提出要領

提出方法は，次の通りとする．

期限	11 月 9 日 (金)PM1:00 まで
用紙	A4
提出場所	山本研究室の入口のポスト
表紙	表紙を 1 枚つけて，以下の項目を分かりやすく記述すること． 授業科目名「電子計算機」 課題名「課題 3　メモリー中の整数の表現」 3E　学籍番号　氏名 提出日
内容	問題の解答．計算課程をきちんと書くこと．

6　付録

6.1　実数の表現

実際のコンピューターを用いた計算では，実数がよく使われる．ここでは，C 言語の倍精度実数型「`double`」で変数を宣言したときの，データの格納の仕方を示す．諸君にとってはかなり難しいと思うので，ここは分かる者のみトライせよ．講義では説明しない．

6.1.1　浮動小数点表示

浮動小数点表示とは，指数化（例えば，-0.123×10^{-2}）して数値を表現する．これは非常に便利な方法で，自然科学では多くつかわれる．コンピューターでも同様で，データが整数と指定されない限りこの浮動小数点が用いられる．実際，この仮数部の (-0.123) と指数の (-2) をメモリーに格納する．この方法の長所と短所は，以下の通りである．

長所 決められたビット数内で，非常に小さな数値から大きな数値まで表現可能になる．

短所 桁落ち誤差が発生する場合がある．

浮動小数点表示を学習するために，必要な言葉の意味は，図 8 の通りである．1 年生の数学の授業で学習したはず．

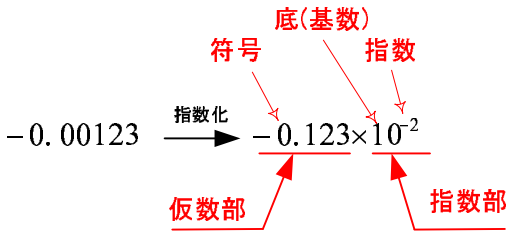

図 8: 指数表現の名称

6.1.2　C 言語の倍精度実数型

IEEE の規格の C 言語の倍精度実数型の「`double`」の表現について説明する．まず，浮動小数点表示のための正規化を図 9 に示す．当然，仮数部，指数部とも 2 進数表現です．仮数部は，符号と `1.XXXX` のように表す．

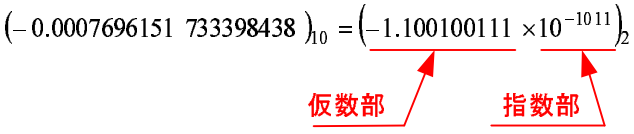

図 9: IEEE 規格表現のための規格化

つぎに，これを IEEE 規格の浮動小数点に表すことを考える．まずその規格の仕様は，以下のようになっている．

- 64 ビット (第 0 ビット〜第 63 ビット) で，浮動小数を表わす．各ビットの構成は，図 10 の通りである．
- 最上位の第 63 ビットが仮数部の符号ビットである．正の場合ゼロで，負の場合 1 になる．
- 指数は 11 ビットでオフセットバイナリ方式で表す．11 ビットで 0〜2047 の値になる．ただし，指数部 11 ビットの値 0 と 2047 は例外処理のために予約されている．11 ビットで表現される値からオフセット値 1023 を引くことにより指数の値が-1022〜1023 の範囲になるように定められている．
- 仮数部は 52 ビットである．小数点以下を，絶対値で表現する．規格化のための整数部は 1 と分かっているので，このためのビットは割り当てられていない．

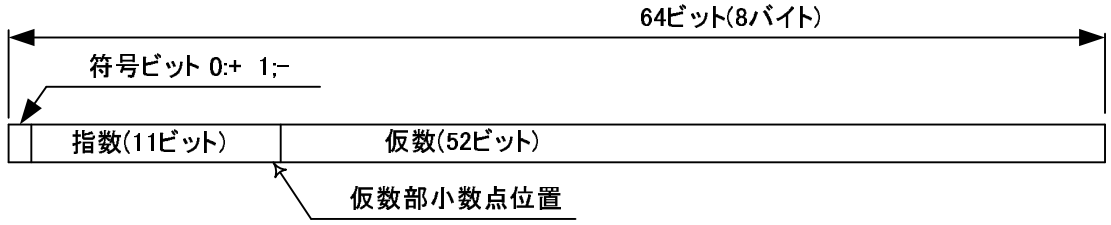

図 10: IEEE 規格 (C 言語の倍精度実数) 表現のビットの内訳

以上の仕様をもとに，図 9 で規格化された数を浮動小数点表示を示す．ほとんどの部分は規格化で分かるが，指数のみ計算が必要である．指数は，オフセットバイナリーで計算するために，まず 10 進数で表す．

$$(-1011)_2 = (-8 - 2 - 1)_{10} = (-11)_{10} \tag{9}$$

不動小数表示の指数は，この式の値に 1023 を加算して求める．すると，

$$(-11 + 1023)_{10} = (1012)_{10} = (1111110100)_2 \tag{10}$$

となる．

これで，すべて準備が整った．不動小数点表示は，図 11 のようになる．実際のコンピューターには，この 64 ビットのデータが格納される．メモリーは 8 ビット (1 バイト) 毎アドレスが割り当てられているので，8 番地分のデータ領域が必要である．

$$\left(-0.0007696151\ 733398438\ \right)_{10} = \left(-1.100100111 \times 10^{-1011}\right)_2$$

指数オフセットバイナリーの計算

$$\left(-11+1023\right)_{10} = (1012)_{10} = (1111110100)_2$$

図 11: IEEE 規格の浮動小数点表示の例