Nombres flottants au format IEEE 754

Table des matières

Nombres flottants au format IEEE 754

1 – Origine, objectif et précision

1.1 – Origine, objectif

L'origine de cette note sans prétention est le fil de discussion UART et décodage virgule fixe dont l'objet est de convertir 3 octets représentant un nombre signé en virgule fixe codé sur 24 *bits* en un nombre flottant normalisé afin de faire des calculs le plus précis possible à partir de mesures physiques délivrées par un circuit électronique tel que le CS5490. En effet ce circuit offre 2 canaux de mesure d'énergie avec une précision de 0.1% et s'interface avec le protocole UART (cf. page 45 - § *6.6.29 Voltage Swell Level* pour plus de détails sur le codage sur 24 *bits* d'une valeur particulière en virgule fixe).

Nous prolongerons cette simple conversion permettant de bien appréhender le format des nombres flottants en proposant des algorithmes d'addition, de soustraction, de multiplication et de division de ces nombres dans le but avoué de ne pas utiliser les bibliothèques logicielles et mathématiques lors de l'implémentation de ces algorithmes. Enfin, nous verrons que la conception d'un opérateur flottant (matériel et logiciel) est envisageable sur les bases de ce que est présenté tout au long de cette note technique accessible au plus grand nombre des lecteurs passionnés du sujet.

1.2 – Précision

Nous nous intéresserons donc dans un premier temps à la conversion dans le format simple 32 *bits* de la norme IEEE 754 d'un nombre signé au format virgule fixe codé sur 24 *bits* (1 *bit* de signe et 23 *bits* de valeur) dont les nombres sont compris dans l'intervalle]-1.0 , +1.0[(bornes exclues) avec un pas de $\frac{1}{2^{23}}$ soit environ 1.2×10^{-7} égal à $0.\underbrace{000000}_{6\,chiffres\,0}12$ permettant de travailler dans cet intervalle avec des incréments proches de $\frac{1}{10^{7}}$.

A noter que ce codage permet d'obtenir au moins 6 chiffres décimaux exacts après la virgule et que cette conversion n'entraîne pas de perte de précision du fait de la même longueur de la mantisse non signée de 23 *bits* dans les 2 formats. Dans la pratique nous constatons que les conversions sont exactes avec plus de 6 chiffres après la virgule voire avec tous les chiffres exacts dans le cas de toutes les puissances négatives du nombre 2 et de leurs sommes comme: 0.001953125 ($\frac{1}{2^{9}}$), -0.0000152587890625 (- $\frac{1}{2^{16}}$), 0.0666656494140625 ($\frac{1}{2^{4}} + \frac{1}{2^{8}} + \frac{1}{2^{12}} + \frac{1}{2^{16}}$), etc.

Inversement, un nombre aussi simple que $\frac{1}{10}$ n'est pas exactement représenté car dans les 2 formats celui-ci est compris entre les 2 conversions disponibles égales respectivement à $0.1\underbrace{0000000}_{7\,chiffres\,0}1490116119384765625$ et $0.0\underbrace{9999999}_{7\,chiffres\,9}4039535522460937 5$ ($\frac{1}{10}$, comme la plupart des nombres décimaux compris dans l'intervalle] -1.0 , +1.0 [(bornes exclues), ne peut être strictement égal à une somme finie d'inverses de puissances croissantes de 2).

2 – Rappels, notations et problème

Ayant indiqué et justifié ci-dessus que les 2 formats étaient semblables et même identiques d'un point de vue de la longueur de la mantisse, il serait logique pour réaliser cette conversion de recopier les 23 *bits* de la mantisse du nombre codé en virgule fixe (que nous noterons par la suite comme V) dans les 23 *bits* de la mantisse du nombre flottant codé dans le format 32 *bits* de la norme IEEE 754 noté F; soit : $F_{mant} = V \text{ masqué avec } \underbrace{11111111111111111111111}_{23\,bits}$ noté par convention $V_{mant} = V \,\&\, 7FFFFF_H$ (masque avec les 23 *bits* de poids faible).

Quant au signe de F noté F_{sign}, il est immédiat que c'est le signe de V; soit : $F_{sign} = V_{sign}$ (égal à la valeur du *bit* MSB de V correspondant au *bit* de poids fort) et que l'exposant de F noté F_{exp} peut être fixé à 0 correspondant à 2^0 puisque V est dans l'intervalle]-1.0, +1.0[(bornes exclues).

Malheureusement cette opération très simpliste conduit à convertir le nombre V codé en virgule fixe dans l'intervalle] -1.0 , +1.0 [(bornes exclues) en un nombre flottant au format 32 *bits* de la norme IEEE 754 dans les 2 intervalles]-2.0, -1.0] $\cup$ [+1.0, +2.0[(bornes extrêmes exclues).

En effet, il est convenu qu'un nombre flottant F exprimé dans le format 32 *bits* de la norme IEEE 754 est : $F = F_{sign} \times (1 + F_{mant}) \times 2^{(Fexp-127)}$ avec une mantisse égale à $\sum_{p=1}^{p=23} \frac{b_p}{2^p}$ où $b_p \in \{0,1\}$ et alors que le nombre d'origine exprimé en virgule fixe V codé sur 24 *bits* est : $V = V_{sign} \times V_{mant}$ avec la même valeur de la mantisse égale à $\sum_{p=1}^{p=23} \frac{b_p}{2^p}$ où $b_p \in \{0,1\}$.
Après cette conversion « brutale », nous obtenons donc un nombre converti en flottant augmenté ou diminué de 1 suivant la valeur du signe V_{sign} . Il va de soit que cela n'est pas satisfaisant mais que le but est proche puisque après cette conversion, il suffit de soustraire ou d'ajouter le nombre flottant 1.0 suivant le signe de V pour obtenir le résultat correct. Le signe F_{sign} de F étant celui de V, le problème se résume à comment soustraire le nombre 1 au moyen d'opérations élémentaires sur les b_p de la valeur $(1 + F_{mant}) = (1 + \sum_{p=1}^{p=23} \frac{b_p}{2^p})$ où $b_p \in \{0,1\}$ et ces valeurs comprises dans l'intervalle [+0.0 , +1.0[(borne supérieure exclue) obtenue après cette conversion « brutale » après abandon du signe V_{sign} .

3 – Solution

Soit k l'indice du premier p pour lequel $b_p \in \{0,1\}$ est différent de 0 et donc $b_k = 1$.

Le nombre (1 + F_{mant}) = (1 + $\sum_{p=1}^{p=23} \frac{b_p}{2^p}$) se simplifie en (1 + $\sum_{p=k}^{p=23} \frac{b_p}{2^p}$) puisque pour tous les p avec $1 \le p < k$, les b_p sont égaux à 0.

Soit S la somme $\sum_{p=k}^{p=23} \frac{b_p}{2^p}$ correspondant à la mantisse F_{mant} codée sur 23 *bits*.

Multiplions cette somme **S** par 2^k pour obtenir la somme **S'** = $2^k . \sum_{p=k}^{p=23} \frac{b_p}{2^p} = \sum_{p=k}^{p=23} \frac{b_p}{2^{p-k}} =$

$\frac{b_k}{2^{k-k}} + \sum_{p=k+1}^{p=23} \frac{b_p}{2^{p-k}} = \frac{b_k}{2^0} + \sum_{p=k+1}^{p=23} \frac{b_p}{2^{p-k}} = 1 + \sum_{p=k+1}^{p=23} \frac{b_p}{2^{p-k}}$ car $b_k = 1$.

Cette somme S' devant représenter une mantisse codée sur 23 *bits* et donc un nombre inférieur à 1, cette somme est par conséquent égale à : S' = $\sum_{p=k+1}^{p=23} \frac{b_p}{2^{p-k}}$ (abandon du terme 1 +).

Construisons le nouveau nombre (1 + F'$_{mant}$) = (1 + **S'**) = $1 + \sum_{p=k+1}^{p=23} \frac{b_p}{2^{p-k}} = \frac{b_k}{2^0} + \sum_{p=k+1}^{p=23} \frac{b_p}{2^{p-k}} =$

$\frac{b_k}{2^{k-k}} + \sum_{p=k+1}^{p=23} \frac{b_p}{2^{p-k}} = \sum_{p=k}^{p=23} \frac{b_p}{2^{p-k}} = 2^k . \sum_{p=k}^{p=23} \frac{b_p}{2^p}$

Par conséquent, (1 + F'$_{mant}$) = $2^k . \sum_{p=1}^{p=23} \frac{b_p}{2^p}$ qui, multiplié par 2^{-k} , est égal à $\sum_{p=k}^{p=23} \frac{b_p}{2^p}$ qui représente la quantité (F_{mant}.- 1).

Nous avons donc par ce moyen soustrait 1 du nombre (1 + F_{mant}) ce qui était le but des opérations.

3.1 – Algorithme de conversion d'un nombre en virgule fixe

L'algorithme de conversion de V en un nombre flottant F est donc le suivant en 6 étapes élémentaires:

1. appliquer : $F_{sign} = V_{sign}$ en isolant le *bit* de poids fort
2. appliquer : $F_{mant} = V_{mant}$ en conservant les 23 *bits* à droite
3. initialiser : F_{exp} = 127 car l'exposant est signé ; 127 correspondant à $2^0 = 1$
4. déterminer dans V_{mant} le nombre de *bits* qui sépare la position du *bit* de signe avec le premier *bit* à 1 le plus à gauche ; soit n ce nombre avec $1 \le n \le 23$. Ce nombre n est également la puissance dans 2^{-n} non égale à 1.
5. décaler de n positions à gauche de F_{mant} en ne conservant que les 23 *bits* à droite
6. soustraire ce nombre n à F_{exp} correspondant à la multiplication par 2^{-n}

Le nouveau nombre ainsi construit égal à : $F_{sign} \times (1 + F_{mant}) \times 2^{(Exp-127)}$ est le nombre flottant F codé au format IEEE 754 et égal au nombre V codé en virgule sur 24 *bits* signés.

3.2 – Exemples

Appliquons cet algorithme sur des exemples simples et vérifions les avec ce convertisseur en ligne

❶ $V = 01000000\ 00000000\ 00000000_B = \underbrace{0}_{1\,bit}\ \underbrace{1000000\,00000000\,00000000}_{23\,bits}{}_B = 0.5$

- $F_{sign} = V_{sign} = 0$
- $F_{mant} = V_{mant} = 100\ 0000\ \ 00000\ 00000\ \ 00000\ 0000_B = 40\ 00\ 00_H$
- $F_{exp} = 127$
- $n = 1$
- décalage de 1 position à gauche de F_{mant} sur 23 *bits* ; $F_{mant} = 00\ 00\ 00_H$
- soustraire 1 à F_{exp} ; $F_{exp} = 126 = 7E_H$

$F = \underbrace{0}_{1\,bit}\ \underbrace{7E}_{8\,bits}\ \underbrace{000000}_{23\,bits}{}_H = 03\ F0\ 00\ 00_H = 0.5$

❷ $V = 10001000\ 10001000\ 10000000_B = \underbrace{1}_{1\,bit}\ \underbrace{0001000\,10001000\,10001000}_{23\,bits}{}_B =$

$$-\left(\frac{1}{2^4} + \frac{1}{2^8} + \frac{1}{2^{12}} + \frac{1}{2^{16}}\right) =$$

$$-(0.0625 + 0.00390625 + 0.000244140625 + 0.0000152587890625) = -0.0666656494140625$$

- $F_{sign} = V_{sign} = 1$
- $F_{mant} = V_{mant} = 000\ 1000\ \ 1000\ 1000\ \ 1000\ 0000_B = 08\ 88\ 80_H$
- $F_{exp} = 127$
- $n = 4$
- décalage de 4 positions à gauche de F_{mant} sur 23 *bits* ; $F_{mant} = 08\ 88\ 80_H$
- soustraire 4 à F_{exp} ; $F_{exp} = 123 = 7B_H$

$F = \underbrace{1}_{1\,bit}\ \underbrace{7B}_{8\,bits}\ \underbrace{088880}_{23\,bits}{}_H = BD\ 88\ 88\ 00_H = -6.66656494140625 \times 10^{-2}$

❸ Conversion de $V = 00000000\ 11111111\ 11111111_B = \underbrace{0}_{1\,bit}\ \underbrace{0000000\,11111111\,11111111}_{23\,bits}{}_B =$

$$\sum_{p=8}^{p=23} \frac{1}{2^p} = \frac{1}{2^8} \sum_{p=0}^{p=15} \frac{1}{2^p} = \frac{2^{16}-1}{2^7.2^{16}} = 0.007812380790710449218750$$

- $F_{sign} = V_{sign} = 1$
- $F_{mant} = V_{mant} = 000\ 0000\ \ 1111\ 1111\ \ 1111\ 1111_B = 00\ FF\ FF_H$
- $F_{exp} = 127$
- $n = 8$
- décalage de 8 positions à gauche de F_{mant} sur 23 *bits* ; $F_{mant} = 7F\ FF\ 00_H$
- soustraire 8 à F_{exp} ; $F_{exp} = 119 = 77_H$

$F = \underbrace{0}_{1\,bit}\ \underbrace{77}_{8\,bits}\ \underbrace{7FFF00}_{23\,bits}{}_H = 3B\ FF\ FF\ 00_H = 7.81238079071044921875 \times 10^{-3}$

Nombres flottants au format IEEE 754

❹ $V = 01111111\ 11111111\ 11111111_B = \underbrace{0}_{1\,bit}\ \underbrace{11111111111111111111111}_{23\,bits}{}_B =$

$\sum_{p=1}^{p=23} \frac{1}{2^p} = \frac{1}{2}\sum_{p=0}^{p=22} \frac{1}{2^p} = \frac{2^{23}-1}{2^{23}} = 0.99999988079071044921875$

- $F_{sign} = V_{sign} = 0$
- $F_{mant} = V_{mant} = 111\ 1111\ \ 1111\ 1111\ \ 1111\ 1111_B = 7F\ FF\ FF_H$
- $F_{exp} = 127$
- $n = 1$
- décalage de 1 position à gauche de F_{mant} sur 23 *bits* ; $F_{mant} = 7F\ FF\ FE_H$
- soustraire 1 à F_{exp} ; $F_{exp} = 126 = 7E_H$

$F = \underbrace{0}_{1\,bit}\ \underbrace{7E}_{8\,bits}\ \underbrace{FFFFE}_{23\,bits}{}_H = 03\ F7\ FF\ FE_H = 9.99999880790710449218750 \times 10^{-1}$

❺ $V = 10000000\ 0000000\ 0000001_B = \underbrace{1}_{1\,bit}\ \underbrace{0000000\,00000000\,00000001}_{23\,bits}{}_B = -\frac{1}{2^{23}} =$

$- 0.00000011920928955078125$

- $F_{sign} = V_{sign} = 1$
- $F_{mant} = V_{mant} = 000\ 0000\ \ 0000\ 0000\ \ 0000\ 0001_B = 00\ 00\ 01_H$
- $F_{exp} = 127$
- $n = 23$
- décalage de 23 positions à gauche de F_{mant} sur 23 *bits* ; $F_{mant} = 00\ 00\ 00_H$
- soustraire 23 à F_{exp} ; $F_{exp} = 104 = 68_H$

$F = \underbrace{1}_{1\,bit}\ \underbrace{68}_{8\,bits}\ \underbrace{000000}_{23\,bits}{}_H = B4\ 00\ 00\ 00_H = -1.1920928955078125 \times 10^{-7}$

Avant de poursuivre avec l'implémentation en Langage C de cet algorithme, remarquons que celui-ci est efficace car :

- il n'utilise que des opérations élémentaires (affectation, masque de *bits*, addition, soustraction et décalage de *bits*)
- s'agissant du décalage de *bit* de la mantisse V_{mant} , celui-ci n'est que d'une position à gauche pour « la moitié » des nombres à convertir et compris dans les 2 intervalles]-1.0, -0.5] ∪ [+0.5, +1.0[(bornes extrêmes exclues), de 2 positions à gauche pour « un quart » des nombres compris dans les 2 intervalles]-0.5, -0.25] ∪ [+0.25, +0.5[(bornes extrêmes exclues), … , de 22 positions à gauche pour les 4 nombres $\{ -2^{-22}-2^{-23}$, -2^{-22} , 2^{-22} , $2^{-22}+2^{-23}\ \}$ et enfin de 23 positions à gauche pour les 2 derniers nombres $\{ -2^{-23}$, $2^{-23}\ \}$. A noter que le nombre de nombres signés V codés en virgule fixe avec une mantisse de 24 *bits* et à convertir en flottant est, en incluant le zéro : $2(2^{23}-1)+1 = 2^{24}-1 = 16777215$

4 – Implémentation de la conversion

La 1ère implémentation de cet algorithme est proposée en Langage C pour une raison de portabilité sur de multiples plates-formes. De plus, ce langage offre l'avantage d'utiliser les unions au sens qu'une zone mémoire peut être directement *mappée* sur différents types de données qui, dans notre cas, simplifie grandement les choses en évitant de faire appel notamment à la bibliothèque « math.h » qui est coûteuse en espace programme et données occupées. Enfin, cette implémentation ouvre la voie, en s'inspirant du code généré, à une implémentation en assembleur transposable dans n'importe quel type de micro-contrôleur.

Explications du programme exécuté sur un système où un *int* est codé sur 32 bits :

- lignes 5 à 13 : Définition d'un nombre non signé sur 32 bits décomposable 3 *bytes* + 1 *byte* non utilisé pour la transcription directe d'une mantisse de 24 *bits* signés.
  ⇨ A noter que la place mémoire occupée est de 4 *bytes*.

- lignes 15 à 23 : Définition d'un nombre flottant IEEE 754 sur 32 *bits* avec sa décomposition en 23 *bits* de mantisse non signée, 8 *bits* d'exposant signés et 1 *bit* de signe.
  ⇨ A noter que la place mémoire occupée est également de 4 *bytes*.

- lignes 32 à 35 : Extraction des 3 arguments définis comme des bytes donnés en hexadécimal et construction directe d'une mantisse signée de 24 *bits*.

- lignes 38 à 46 : Conversion brutale de cette mantisse signée en un nombre flottant qui est dans les 2 intervalles]-2.0, -1.0] ∪ [+1.0, +2.0[(bornes extrêmes exclues).

- lignes 52 à 61 : Soustraction de 1.0 de ce nombre flottant ; le résultat de la conversion de la mantisse de 24 *bits* signée est disponible sous forme d'un nombre flottant au format IEEE 754 dans l'intervalle]-1.0, +1.0[(bornes extrêmes exclues) et présenté à l'écran à la ligne 63.
  ⇨ A noter que la variable *n* peut être déclarée à la ligne 53 comme un *unsigned char* puisque 23 décalages maximum sont nécessaires dans le pire des cas.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

union VALUE_24_BITS {
  unsigned int i;
  struct {
    unsigned int lsb   : 8;
    unsigned int mid   : 8;
    unsigned int msb   : 8;
    unsigned int dummy : 8;
  }s;
} value_24_bits;

union MY_FLOAT {
  float f;
  struct {
    unsigned int mant  : 23;
    unsigned int exp   : 8;
    unsigned int sign  : 1;
  } ieee754;
  unsigned int i_32bits;
} my_float;

int main(int argc, char **argv)
{
  union VALUE_24_BITS v_mant;
  union MY_FLOAT my_float;

  if (argc > 3) {
    // Prise des 3 bytes de mantisse en hexa en argument et conversion immediate en un 'int'
    v_mant.i = 0;
    v_mant.s.msb = (unsigned char)strtol(argv[1], NULL, 16);
    v_mant.s.mid = (unsigned char)strtol(argv[2], NULL, 16);
    v_mant.s.lsb = (unsigned char)strtol(argv[3], NULL, 16);

    // Initialisation nombre flottant a 0.0
    my_float.i_32bits = 0;

    if (v_mant.i & 0x800000) {
      my_float.ieee754.sign = 1;  // Nombre negatif
      v_mant.i &= 0x7fffff;       // Abandon du bit de signe mantisse
    }

    my_float.ieee754.mant = v_mant.i;
    my_float.ieee754.exp  = v_mant.i ? 0x7f : 0x00;

    printf("Mantisse %s [0x%06x]\n",
      (my_float.ieee754.sign ? "negative" : "positive"), v_mant.i);

    // Soustraction de 1.0
    int m = my_float.ieee754.mant;
    int n = 0;
    if (my_float.ieee754.mant) {
      while (!(m & 0x800000)) {
        m <<= 1;
        n += 1;
      }
      my_float.ieee754.mant <<= n;
      my_float.ieee754.exp -= n;
    }

    printf("=> Float [0x%08x] = [%.8f] (sign [%d] exp [0x%02x] mant [0x%06x])\n",
      my_float.i_32bits, my_float.f,
      my_float.ieee754.sign, my_float.ieee754.exp, my_float.ieee754.mant);
  }
  else {
    printf("Usage: $%s 0x<mantisse MSB> 0x<mantisse MID> 0x<mantisse LSB>\n", argv[0]);
    printf("Example: $%s 0x40 0x00 0x01\n", argv[0]);
    return 2;
  }

  return 0;
}
```

Annexes et références

Articles Wikipedia

- *IEEE 754 - Format simple précision (32 bits)*
- *Virgule fixe*
- *Séries géométriques* (en particulier et si $n \geq 0$: $\sum_{p=0}^{p=n} \frac{1}{2^p} = \frac{1-\frac{1}{2^{n+1}}}{1-\frac{1}{2}} = \frac{2^{n+1}-1}{2^n}$)

Autres

1) Convertisseur en ligne : *Float to decimal and vice versa*
2) Langage C : *les unions*