Le grand livre

de MECRISP Forth

version 1.0 - 19 décembre 2023

Auteur

- Marc PETREMANN

Table des matières

Un vrai FORTH 32 bits avec MECRISP

MECRISP est un vrai FORTH 32 bits. Qu'est-ce que ça signifie ?

Le langage FORTH privilégie la manipulation de valeurs entières. Ces valeurs peuvent être des valeurs littérales, des adresses mémoires, des contenus de registres...

Les valeurs sur la pile de données

Au démarrage de MECRISP, l'interpréteur FORTH est disponible. Si vous entrez n'importe quel nombre, il sera déposé sur la pile sous sa forme d'entier 32 bits :

```
35
```

Si on empile une autre valeur, elle sera également empilée. La valeur précédente sera repoussée vers le bas d'une position :

```
45
```

Pour faire la somme de ces deux valeurs, on utilise un mot, ici + :

```
+
```

Nos deux valeurs entières 32 bits sont additionnées et le résultat est déposé sur la pile. Pour afficher ce résultat, on utilisera le mot . :

```
. \ affiche 80
```

En langage FORTH, on peut concentrer toutes ces opérations en une seule ligne:

```
35 45 + .  \ display 80
```

Contrairement au langage C, on ne définit pas de type **int8** ou **int16** ou **int32**.

Avec MECRISP, un caractère ASCII sera désigné par un entier 32 bits, mais dont la valeur sera bornée [32..256[. Exemple :

```
decimal
67 emit  \ display C
```

Avec MECRISP Forth, un entier 32 bits signé sera défini dans l'intervalle -2147483648 à 2147483647.

Parfois, on parle de demi-mot. Un demi-mot numérique concerne la partie 16 bits de poids fort ou poids faible d'un entier 32 bits. Un demi-mot sera défini dans l'intervalle 0 à 65,535

Les valeurs en mémoire

MECRISP permet de définir des constantes, des variables. Leur contenu sera toujours au format 32 bits. Mais il est des situations où ça ne nous arrange pas forcément. Prenons un exemple simple, définir un alphabet morse. Nous n'avons besoin que de quelques octets :

- un pour définir le nombre de signes du code morse
- un ou plusieurs octets pour chaque lettre du code morse

```
create mA ( -- addr )
    2 c,
    char . c,   char - c,

create mB ( -- addr )
    4 c,
    char - c,   char . c,   char . c,   char . c,

create mC ( -- addr )
    4 c,
    char - c,   char . c,   char - c,   char . c,
```

Ici, nous définissons seulement 3 mots, `mA`, `mB` et `mC`. Dans chaque mot, on stocke plusieurs octets. La question est: comment va-t-on récupérer les informations dans ces mots?

L'exécution d'un de ces mots dépose une valeur 32 bits, valeur qui correspond à l'adresse mémoire où on a stocké nos informations morse. C'est le mot `c@` qui va nous servir à extraire le code morse de chaque lettre :

```
mA c@ .  \ affiche 2
mB c@ .  \ affiche 4
```

Le premier octet extrait ainsi va nous servir à gérer une boucle pour afficher le code morse d'une lettre :

```
: .morse ( addr -- )
    dup 1+ swap c@ 0 do
        dup i + c@ emit
    loop
    drop
  ;
mA .morse   \ affiche .-
mB .morse   \ affiche -...
mC .morse   \ affiche -.-.
```

Il existe plein d'exemples certainement plus élégants. Ici, c'est pour montrer une manière de manipuler des valeurs 8 bits, nos octets, alors qu'on exploite ces octets sur une pile 32 bits.

Traitement par mots selon taille ou type des données

Dans tous les autres langages, on a un mot générique, genre `echo` (en PHP) qui affiche n'importe quel type de donnée. Que ce soit entier, réel, chaîne de caractères, on utilise toujours le même mot. Exemple en langage PHP :

```
$bread = "Pain cuit";
```

```
$price = 2.30;
echo $bread . " : " . $price;
// affiche   Pain cuit: 2.30
```

Pour tous les programmeurs, cette manière de faire est LA NORME! Alors comment ferait FORTH pour cet exemple en PHP?

```
: pain s" Pain cuit" ;  fonctionne pas @todo à corriger
: prix s" 2.30" ;
pain type   s" : " type    prix type
\ affiche   Pain cuit: 2.30
```

Ici, le mot `type` nous indique qu'on vient de traiter une chaîne de caractères.

Là où PHP (ou n'importe quel autre langage) a une fonction générique et un analyseur syntaxique, FORTH compense avec un type de donnée unique, mais des méthodes de traitement adaptées qui nous informent sur la nature des données traitées.

Voici un cas absolument trivial pour FORTH, afficher un nombre de secondes au format HH:MM:SS:

```
: :##
    #  6 base !
    #  decimal
    [char] : hold
  ;
: .hms ( n -- )
    0 <# :## :## # # #>  type
  ;
4225 .hms  \ display: 01:10:25
```

J'adore cet exemple, car, à ce jour, **AUCUN AUTRE LANGAGE DE PROGRAMMATION** n'est capable de réaliser cette conversion HH:MM:SS de manière aussi élégante et concise.

Vous l'avez compris, le secret de FORTH est dans son vocabulaire.

Conclusion

FORTH n'a pas de typage de données. Toutes les données transitent par une pile de données. Chaque position dans la pile est TOUJOURS un entier 32 bits !

C'est tout ce qu'il y a à savoir.

Les puristes de langages hyper structurés et verbeux, tels C ou Java, crieront certainement à l'hérésie. Et là, je me permettrai de leur répondre : pourquoi avez-vous besoin de typer vos données ?

Car, c'est dans cette simplicité que réside la puissance de FORTH: une seule pile de données avec un format non typé et des opérations très simples.

Et je vais vous montrer ce que bien d'autres langages de programmation ne savent pas faire, définir de nouveaux mots de définition :

```
: morse: ( comp: c -- | exec -- )   fonctionne pas @todo à corriger
    create
        c,
    does>
        dup 1+ swap c@ 0 do
            dup i + c@ emit
        loop
        drop space
  ;
2 morse: mA     char . c,   char - c,
4 morse: mB     char - c,   char . c,   char . c,   char . c,
4 morse: mC     char - c,   char . c,   char - c,   char . c,
mA mB mC    \ display   .- -... -.-.
```

Ici, le mot **morse:** est devenu un mot de définition, au même titre que **constant** ou **variable**...

Car FORTH est plus qu'un langage de programmation. C'est un méta-langage, c'est à dire un langage pour construire votre propre langage de programmation....

Les nombres réels avec MECRISP Forth

Avec MECRISP Forth, les nombres réels sont saisis en mettant le caractère « , » (virgule) dans le nombre saisi :

```
3,2 f.       \ affiche: 3,19999999953433871269226074218750
3,3 f.       \ affiche: 3,29999999813735485076904296875000
4,0 f.       \ affiche: 4,00000000000000000000000000000000
4,5 f.       \ affiche: 4,50000000000000000000000000000000
11,25 f.     \ affiche: 11,25000000000000000000000000000000
11,1234 f.   \ affiche:  11,12339999992400407791137695312500
```

Vous l'avez compris, certaines valeurs décimales induisent des erreurs. On utilisera donc les nombres réels seulement dans certaines situations.

Nombres réels sur 32 bits

Sur un processeur sans unité de calcul flottant, toutes les opérations sont effectuées par logiciel via la bibliothèque du compilateur C (ou mots Forth) et ne sont pas visibles par le programmeur. Mais les performances sont très faibles. Sur un processeur doté d'une unité de calcul flottante, toutes les opérations sont entièrement réalisées matériellement en un seul cycle, pour la plupart des instructions.

Le compilateur C (ou Forth) n'utilise pas sa propre bibliothèque à virgule flottante mais génère directement des instructions natives FPU.

Voici comment est organisé un nombre réel sur 32 bits dans MECRISP Forth :

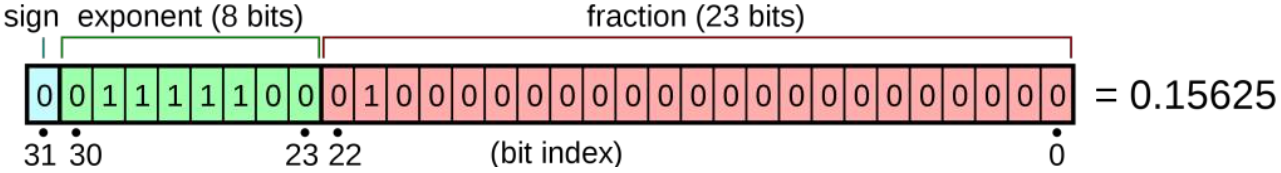

Figure 1: structure d'un nombre réel sur 32 bits

La portée des nombres pouvant être ainsi utilisés est dans l'intervalle [1.18E-38 → 3.40E38]. Plus la partie entière sera grande, moins on peut exploiter de partie décimale. Même un nombre aussi simple que 1/10 sera sujet à des défauts de représentation :

```
0,1 f.       \ affiche: 0,09999999860301613807678222656250
```

On réservera donc l'utilisation des nombres réels pour certaines utilisation. Ici, en trigonométrie :

```
30 sin f.       \ affiche: 0,50000000256113708019256591796875
45 sin f.       \ affiche: 0,71579438890330493450164794921875
```

Ici, si le sinus de 30° est exact, il n'en est pas de même pour celui de 45° qui devrait être 0.70710678...

À éplucher : https://www.spyr.ch/twiki/bin/view/MecrispCube/FloatingPointUnit

Ressources

MECRISP Forth

Site en deux langues (français, anglais) avec plein d'exemples
https://mecrisp.arduino-forth.com/

Mecrisp download

Le site de référence pour récupérer la version qui convient à vos cartes électroniques
https://sourceforge.net/projects/mecrisp/

Mecrisp Stellaris Unofficial UserDoc

Documentation en ligne très complète
https://mecrisp-stellaris-folkdoc.sourceforge.io/index.html

Index lexical