Chapitre 2

I:\ Ce qui est "visible" :

Ce tutorial est prévu pour les calculatrices Texas Instrument modèles TI-89/92/V-200. A peu de choses près, ces trois machines sont identiques : leurs différences majeures sont au niveau de la taille de l'écran, et du clavier.

Il existe deux versions matérielles différentes : les HW1, et HW2.
Les HW1 sont les plus anciennes, les HW2 les plus récentes. Les HW2 comportent quelques fonctionnalités supplémentaires par rapport aux HW1 (par exemple, les HW1 n'ont pas de support d'horloge). La V-200 est considérée comme une HW2 (Il n'existe pas de V-200 HW1).
Au cours de notre apprentissage de la programmation en Assembleur, il sera rare que nous ayons à nous soucier des différences entre les différentes versions matérielles, mais, quand il le faudra, nous préciserons que c'est le cas.

Il existe aussi plusieurs versions de "cerveaux". Ce "cerveau" est le logiciel qui vous permet de faire des mathématiques, de programmer en TI-BASIC, de dessiner, de lancer des programmes en Assembleur, ... ; bref, ce "cerveau" est ce grâce à quoi votre machine est plus qu'un simple tas de composants électroniques.
Différentes versions allant de 1.00 sur TI-92+, 1.01 sur TI-89, et 2.07 sur V-200, à, pour le moment, 2.09 (sortie durant le second trimestre 2003) ont été diffusées par Texas Instrument. En règle générale, plus la version est récente, plus elle comporte de fonctions intégrées, directement utilisables en Assembleur.
Ce "cerveau" est généralement appelé "AMS" (pour Advanced Mathematic Software), ou, par abus de langage, "ROM" (Pour Read Only Memory), puisque l'AMS est stocké dans un mémoire flashable de ce type.
Le plus souvent possible, nous essayerons de rédiger des programmes compatibles entre les différentes versions de ROM, mais, dans les rares cas où ce ne sera pas possible (par exemple, parce que nous aurons absolument besoin de fonctions qui ne sont pas présentes sur certaines anciennes ROM), nous le signalerons.

II:\ Le Microprocesseur :

Nos TIs sont dotées d'un processeur Motorola 68000, cadencé à 10MHz sur HW1, et à 12MHz sur HW2.
(Ce processeur était utilisé sur les premiers Macintosh, sur certaines consoles ATARI, et est toujours utilisé sur les rames de TGV atlantiques !)

Ce processeur comporte 8 registres de données, nommés de d0 à d7, et 8 registres d'adresse, de a0 à a7. (Un registre est un petit emplacement mémoire situé à l'intérieur même du processeur, et qui est donc très rapide ; c'est le type de mémoire le plus rapide, mais aussi le plus cher, ce qui explique pourquoi il n'y en a que si peu). Ces 16 registres ont une taille de 32 bits.
Les registres de données, de même que ceux d'adresse (de a0 à a6 seulement : a7 est un peu particulier), sont généraux, ce qui signifie que l'on peut mettre un peu ce qu'on veut dedans (Il n'y a pas, comme sur certains autres processeurs, un registre réservé pour les boucles, ou autre). Cependant, étant donné le faible nombre de registres, il vaut mieux les utiliser de façon judicieuse !
Le registre a7 est particulier dans le sens où il joue le rôle de "pointeur de pile" : c'est lui qui contient l'adresse du sommet de la pile. La pile ("Stack", en anglais), est une zone mémoire un peu particulière, sur laquelle on "empile" ou "dépile" des données... Nous verrons plus tard ce que cela signifie exactement.

Le M68k (Abréviation de Motorola 68000) possède également un registre nommé "PC" (Program Counter) qui sert à indiquer en permanence au processeur quelle est la prochaine instruction qu'il devra utiliser. Rassurez-vous, lors de l'exécution normale d'un programme, c'est le processeur lui-même qui se charge de modifier de manière adéquate la valeur du PC. Ce registre est de 24 bits.

Ce processeur dispose aussi d'un registre de statut, nommé "SR" (Status Register), sur 16 bits. Nous étudierons tout d'abord les 8 bits de poids faible, puis les 8 bits de poids fort.
Voici le schéma représentatif du SR :

Nom	T	-	S	-	-	I2	I1	I0	-	-	-	X	N	Z	V	C
N° bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Registre de Flags :
Il s'agit de l'octet de poids faible du SR.
Voici, globalement, la signification des différents bits utilisés le comportant :

- Bit C (Carry) : Utilisé comme retenue.
  Ce bit servira par exemple de neuvième bit si vous additionnez deux nombres de huit bits.
- Bit V (oVerflow) : Ce bit sera armé (vaudra 1) en cas de valeur trop grande pour être représentable.
- Bit Z (Zero) : Ce bit sera armé si le résultat d'un calcul (en général) vaut 0.
- Bit N (Negative) : Ce bit sera armé si le bit de poids fort d'un résultat l'est (Nous verrons bientôt pourquoi, quand nous étudierons le complément à deux).
- Bit X (eXtended) : Ce bit est une copie du bit C, mais ne sera pas modifié de la même façon.

Octet Système :
Il s'agit de l'octet de poids fort du SR.
Voici à présent, de façon rapide, la signification des différents bits utiles le composant :
Attention, ceci est assez technique. Si vous n'avez pas déjà de bonnes connaissances en architecture et fonctionnement des processeurs, il est probable que vous ne saisirez pas ce qui suit. Ce n'est pas pour l'instant grave : ceci n'est utilisé qu'à un niveau assez avancé de programmation. Vous comprendrez en temps voulu.

- Bits I0, I1, I2 : Ces trois bits permettent de définir le masque d'interruptions du processeur. Plus la valeur codée par ces trois bits est élevée, plus le niveau de priorité est élevé.
- Bit S (Superviseur) : Quand ce bit est armé, c'est que le processeur n'est pas en mode utilisateur (le mode par défaut, le plus utilisé), mais en mode Superviseur. Le mode Superviseur vous donne accès à toutes les instructions du processeur, notamment à celles de contrôle du système, qui ne sont pas accessibles en mode utilisateur. Vous ne pouvez pas modifier l'octet système du SR lorsque vous n'êtes pas en mode Superviseur ; donc, pour passer du mode utilisateur au mode Superviseur, il convient d'appeler une routine système spéciale.
  Travailler en mode Superviseur n'est utile que pour écrire des programmes contrôlant le système, et nécessitant un niveau avancé de programmation. Nous ne travaillerons pas en mode Superviseur avant fort longtemps.
- Bit T (Trace) : Lorsque ce bit est armé, une interruption est générée à la fin de chaque instruction. Ce bit est utilisé, ainsi que cette fonctionnalité, par les débogueurs. Je ne voie pas d'autre type de programme qui puisse lui fournir un emploi.

III:\ Façon que la TI a de calculer :

Ce dont nous allons ici parler tient encore une fois plus au processeur qu'à la TI, puisque nous nous pencherons sur la représentation des nombres en interne (au niveau du processeur et de la mémoire).

A: Représentation des nombres :

Les processeurs Motorola stockent les nombres en Big Endian ; c'est-à-dire que l'octet de poids fort d'un nombre est rangé "à gauche".
Un petit exemple rendra les choses plus clair : le nombre $12345678 sera représenté en mémoire par la suite d'octets suivant : $12, $34, $56, $78.
A titre d'information, les processeurs Intel de la famille ix86 (386, 486, et Pentiums) auraient représenté ce nombre par la suite d'octets suivant : $78, $56, $34, $12.
Force est de reconnaître que la technique utilisée par Motorola permet au programmeur de parfois se "simplifier la vie" !

B: Codage des entiers en Binaire :

Les entiers, que ce soit négatifs, ou positifs, sont codés selon la technique dite du complément à deux.

Pour les entiers positifs :
Nous avons déjà vu comment passer du décimal au binaire ; c'est exactement ce qu'il faut faire ici.
Par exemple, en travaillant avec des mots de deux octets, on aura :

16849 = %0100000111010001 (=$41D1)

Pour les entiers, le bit de poids fort est considéré comme un bit de signe. S'il vaut 0, l'entier est positif, et s'il vaut 1, l'entier est négatif.
Par exemple, %1101111000010001 ne sera pas considéré comme 56849, mais comme nombre négatif !

Pour les entiers négatifs :
Cette fois-ci, c'est un peu plus difficile ; mais il suffira de retenir la technique utilisée, qui sera toujours la même. (Il y a d'autres méthodes, mais c'est celle-ci que j'utilise, car c'est celle que je considère comme la plus simple).
Voici, toujours avec des mots de deux octets, comment encoder la valeur -16849 :
On commence par "traduire" la valeur absolue du nombre ; ici, 16849 :

16849 = %0100000111010001

Puis, on inverse la valeur de tous les bits (les bits à 0 passent à 1, en inversement ; cela revient à faire le complément à un du nombre), on obtient :

%1011111000101110

Enfin, on ajoute 1 au résultat :

```
  %1011111000101110
+ %0000000000000001
= %1011111000101111
```

Remarque : Une addition en binaire se fait de la même façon qu'en décimal. La seule chose dont il faut se rappeler, c'est que 1+1 = 10.

(Précisons que la TI est dotée d'une fonction de conversion qui parvient à effectuer ceci ! Il suffit de saisir -16849>Bin).

Utiliser la technique du complément à deux permet de mémoriser :

- Sur un octet, des valeurs allant de -128 (%1000.0000) à 127 (%0111.1111).
- Sur deux octets, des valeurs allant de -32768 (%1000.0000.0000.0000) à 32767 (% 0111.1111.1111.1111).
- Sur quatre octets, des valeurs allant de -2147483 à 2147483647.

Comme vous avez pu le remarquer, la valeur minimale représentable est égale à la valeur maximale représentable, plus un. Ceci n'est pas sans inconvénients, et peut parfois poser problème, si l'on ne fait pas assez attention.
Il arrive que l'on travaille avec des valeurs dites "non-signées" ("unsigned" en anglais). Dans ce cas, les 8, 16, ou 32 bits sont tous utilisés pour coder une valeur positive, comprise entre 0 et, respectivement, 2^8-1, 2^16-1 et 2^32-1.

IV:\ Un peu de vocabulaire :

Le processeur Motorola 68000 que nous utilisons, pour lequel nous allons programmer, est capable de reconnaître trois types de données entières. A chacun de ces types correspond une lettre, qui sera souvent utilisée comme suffixe pour certaines instructions, afin que le processeur puisse déterminer sur quel type de données elles doivent travailler ; le suffixe sera séparé de l'instruction par un point.

- Octet : Le type le plus petit (mis à part le bit, bien entendu !), mais qui constitue la base de toutes nos mémoires et systèmes informatiques. Octet se dit "Byte" en anglais (ne pas confondre avec "bit" : un "byte" est composé de huit "bits"), ce qui explique que le suffixe qui lui corresponde soit ".b".
- Mot : Le type le plus courant sur processeur M68k. Le mot est codé sur 16 bits (le 68000 est un processeur 16 bits en externe). "Mot" se dit "Word" en anglais. Le suffixe correspondant est donc ".w".
- Mot Long : Un mot long ("Longword" en anglais) est codé sur 32 bits (le M68k est un processeur 32 bits en interne, comme en témoigne la taille des registres). Le suffixe associé aux mot longs est ".l".

Je n'ai pas l'intention de vous forcer à ingérer plus de théorie tout de suite, même si je suis d'avis que celle-ci pouvait s'avérer nécessaire, sans vous permettre enfin de programmer ; dès le prochain chapitre, nous réaliserons notre premier programme en Assembleur.
J'aurai pu, comme j'en avais l'intention au départ, partir dès le premier chapitre sur un exemple de programme, et vous fournir les explications que je vous ai donné dans les chapitres 1 et 2 en même temps que celles portant sur cet exemple, mais j'ai vite réalisé que cela ferait beaucoup trop de nouveautés en même temps. J'ai donc préféré cette solution qui même si elle est moins "ludique" car ne commence pas par du code, est probablement plus adaptée pour une bonne raison : lorsque vous parviendrez au premier exemple de code, vous disposerez déjà de bases qui, bien que assez peu approfondies, devraient vous faciliter la compréhension.

Bonne continuation !