Programmer en assembleur 6800 - 6811

[Mise à jour le : 31/10/2024]

- **Ressources**
  - En ligne : Jeu d'instruction du MC6800
- **A télécharger** si travail maison
  - Logiciel : simulateur SDK6800/6811
  - Document réponse du TP

Mots-clés

Adresse, donnée, instruction machine, mnémonique, opcode, opérande, directive d’assemblage, mode d’adressage, assembleur, variable, constante, opération symbolique.

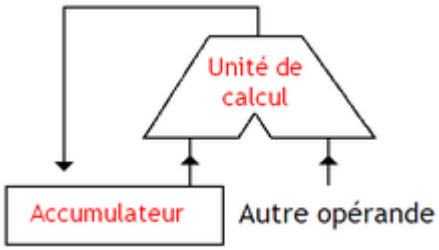
1. Généralités

« Le 6800 est un microprocesseur **8 bits** produit par Motorola et sorti peu de temps après l'Intel 8080 en 1975. La famille de processeurs **6800** a alimenté l'explosion précoce de l'informatique domestique. Ses **dérivés** ont été les processeurs de choix pour de nombreux ordinateurs personnels, notamment **Apple**, Commodore64, **Nintendo**, etc., et de nombreuses consoles de jeux. Ses descendants directs tels que le 68HC11 sont encore utilisés aujourd'hui.» Wikipédia

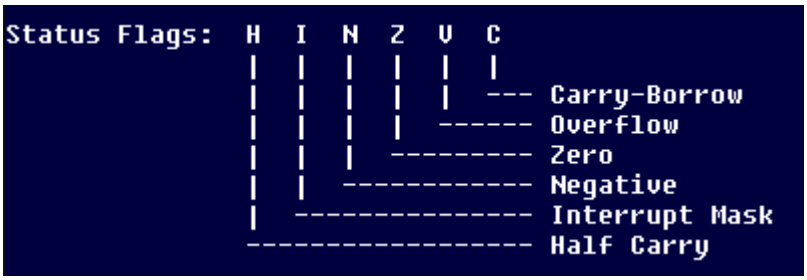
- **Les registres du MC6800**

Motorola 6800 registers

^{15} ^{14} ^{13} ^{12} ^{11} ^{10} ^{09} ^{08} ^{07} ^{06} ^{05} ^{04} ^{03} ^{02} ^{01} ^{00} *(bit position)*	
Main registers	
A	A accumulator
B	B accumulator
Index registers	
IX	Index register
SP	Stack Pointer
Program counter	
PC	Program Counter
Status register	
1 1 H I N Z V C	Flags



- **Accumulateurs A, B**
  Ces registres servent à stocker temporairement les opérandes et les résultats lors des calculs arithmétiques et logiques.
- **Registre d'index X**
  Ce registre participe au calcul de l'adresse d'un opérande durant l'exécution d'un programme, par exemple pour faire des opérations répétitives sur plusieurs éléments d'un vecteur ou d'un tableau.
- **Compteur programme PC**
  Ce registre est également connu sous le nom de pointeur d'instruction ou simplement PC. Il fait le suivi de l'adresse mémoire de la prochaine instruction à exécuter dans un programme.
- **Registre pointeur de pile**
  Lorsqu'une fonction est appelée, le pointeur de pile est utilisé pour allouer de l'espace sur la pile pour les variables locales de la fonction et pour stocker l'adresse de retour.



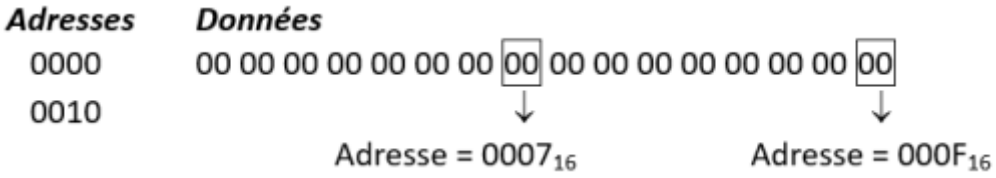
- **Le registre d'état**
  Le registre d'état, ou registre de drapeaux (flags register), est un ensemble de bits représentant des drapeaux (flags) au sein d'un processeur. Le MC6800 possède six drapeaux.
    - **N** : passe à 1 lorsque le résultat d'une opération est négatif.

- **Z** : passe à 1 lorsque le résultat d'une opération est nul.
- **V** : passe à 1 lorsque le résultat n'est pas juste en arithmétique signée.
- **C** : passe à 1 lorsque le résultat d'une opération est > 255.
- **H** : donne l'état de la retenue lors d'une opération entre le bit3 et le bit4.
- **I** : autorise une interruption IRQ lorsqu'il est à 0.

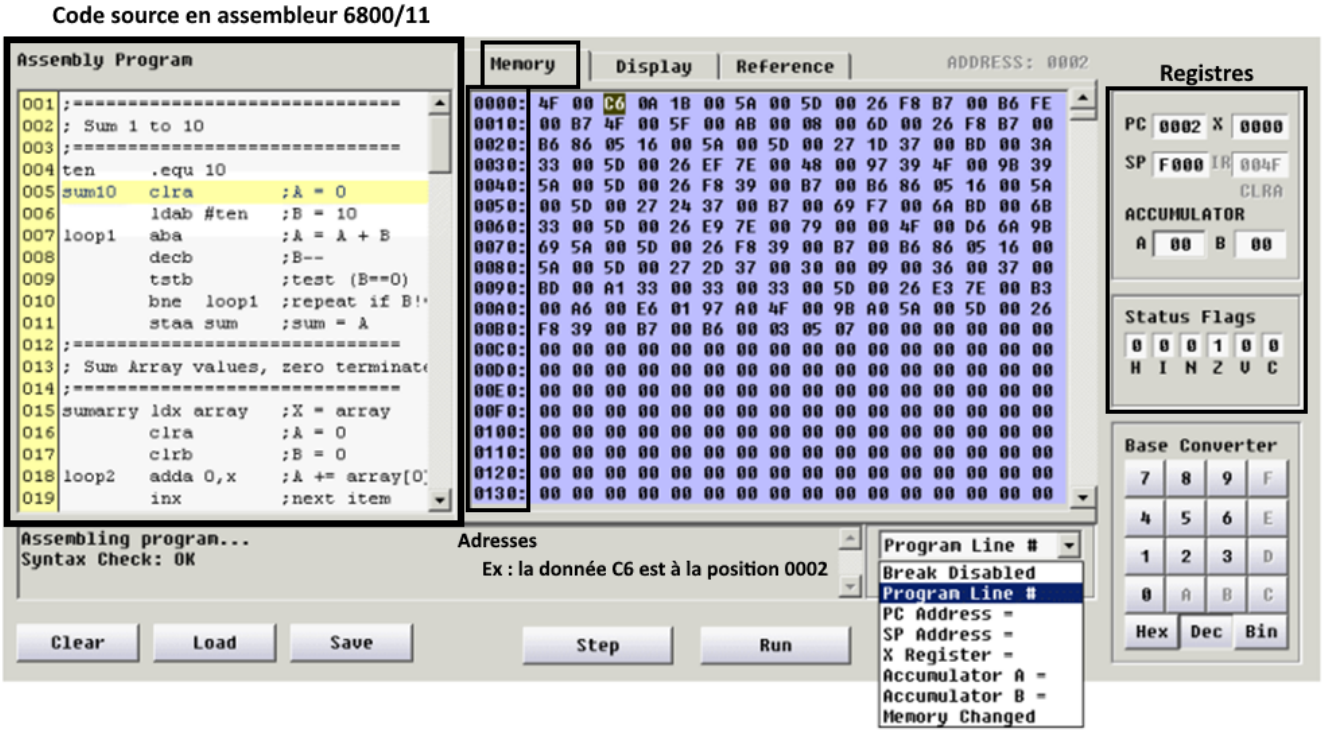
2. Le simulateur SDK6800/6811

- **Ressource** : Mise en oeuvre du simulateur

  L'assembleur place les instructions-machine et les données en mémoire conformément aux directives d'assemblage. La **mémoire du simulateur** est présentée sous forme **matricielle** comme ci-dessous :



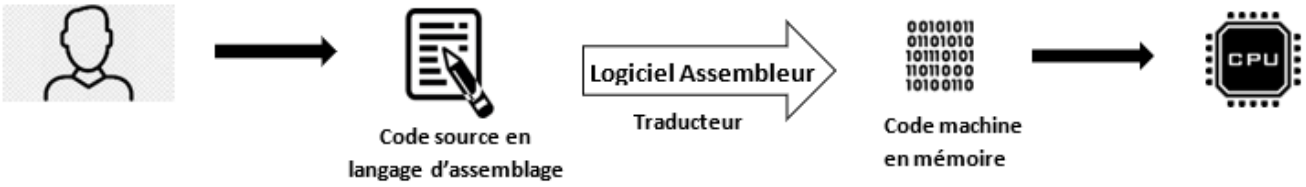
- **Organisation du simulateur**



3. Langage d'assemblage et code machine

Le **langage d'assemblage** est un équivalent du langage machine pour lequel les chaînes binaires de l'instruction-machine sont remplacées par des **mnémoniques alphanumériques** plus aisément mémorisable et manipulable

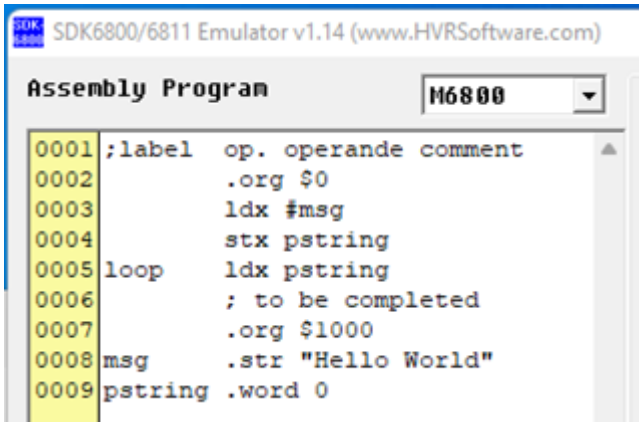
par un être humain. Un **traducteur** transforme ce langage vers le langage machine équivalent : c'est l'**assembleur**.



3.1 Organisation d'une ligne de code

Le **code source** d'un programme écrit en langage d'assemblage se décompose en champs de texte dans le simulateur :

[*Étiquette*] Opération [*Opérande(s)*] [*Commentaire*]



- Le champ **Étiquette**(**Label**) est utilisé pour définir un symbole. Pour ignorer ce champ, on introduit au moins un espace ou une tabulation.
  - Exemple : *msg*
- Le champ **Opération** est occupé par un **opcode** ou une **directive d'assemblage**.
  - Exemple d'opcode
    - ***ldx*** signifiant loadx, charge une valeur dans le registre x.
  - Exemple de directives d'assemblage
    - **.str** "*chaîne*" (str signifie string). Cette directive force l'assembleur à coder en ASCII les caractères de la chaîne entre guillemets .
    - **.org** *adresse* (.org signifie origine). Cette directive force l'assembleur à placer ce qui suit à la position *adresse* (ici le texte "Hello World" est placé entre les adresses 1000_{16} et $100A_{16}$)
- Le champ **Opérande(s)** contient une adresse ou des données. Il est ignoré lorsque l'instruction utilise le mode d'adressage implicite.
  - Exemples d'opérandes : #msg, pstring, $1000, "Hello World"
- Le champ **Commentaire** est utilisé pour la documentation du logiciel. Un commentaire commence par un **point-virgule** et peut se situer à la fin ou au début d'une déclaration.
  - Exemple : ;label op. operande comment

3.2 Un premier programme étape par étape

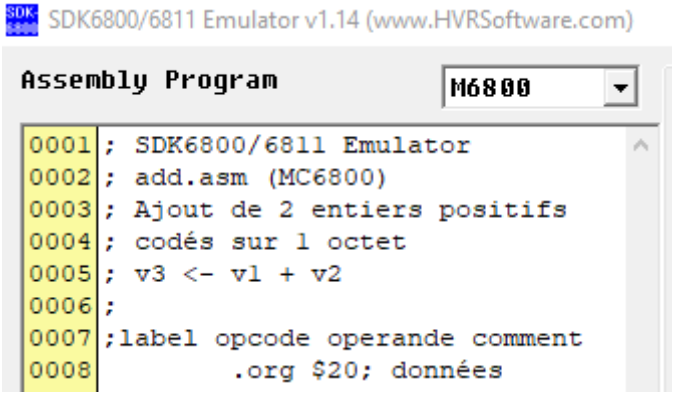
☛ **Ouvrez** le simulateur SDK6800/6811 en cliquant sur 

> **Opération à réaliser**
> **v3 ← v1 + v2** # Les variables v3, v2 et v1 sont des entiers, $0 \leq vx_{16} \leq FF$

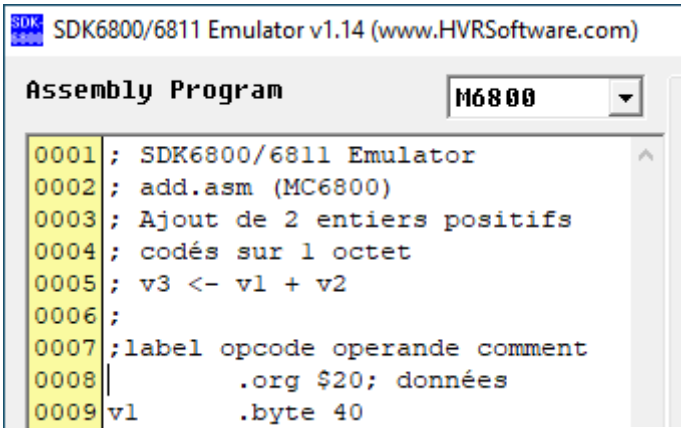
- **ÉTAPE 1 - Placer, déclarer et initialiser les variables dans la mémoire**
  Pour effectuer ces opérations, nous allons utiliser des directives d'assemblage et des étiquettes.

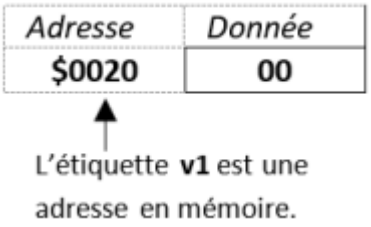
> **Directive d'assemblage**
> Les directives d'assemblage sont des **pseudo-instructions** : elles ne correspondent à aucune instruction-machine ; ce sont des ordres destinés à l'assembleur.

1. **Fixer la position des variables dans la mémoire** avec la directive **.org**
   ☛ **Fixez** la position de la zone des variables à partir de l'adresse $\mathbf{20}_{16}$ comme ci-dessous. Le symbole **$** signifie que la valeur qui suit est en **hexadécimal** (base 16).



2. **Déclarer et initialiser des variables**
   La position des variables étant fixée, vous allez les **identifier** à l'aide d'une **étiquette**, les **déclarer** à l'aide de la directive **.byte** et les **initialiser** en leur affectant une valeur.
   ☛ **Complétez** le code source comme ci-dessous. Placez **v2** avec la valeur **80** puis **v3** avec la valeur **0** sous v1. **Sauvegarder** le code source sous le nom ***add.asm*** sur le serveur (voir prof).

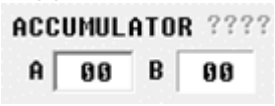




Etiquette(Label)

Une **étiquette** est une chaîne de caractères permettant de **nommer** une **instruction** ou une **variable**. Une étiquette correspond à une **adresse** dans le programme.

- **ÉTAPE 2 - Fixer la position et écrire le code source du programme**
  Rappel : on souhaite effectuer l'opération v3 ← v1 + v2

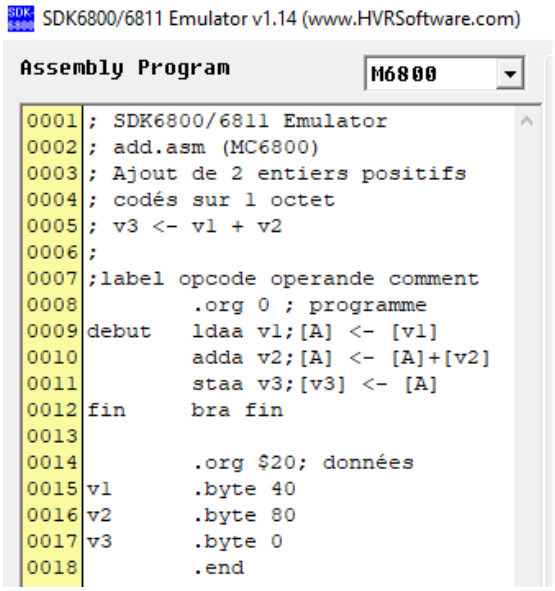


L'opération d'addition entre v1 et v2 se fera à l'aide d'un **accumulateur** . Comme cela est décrit dans les généralités, le MC6800 possède deux accumulateurs (**A** et B). La manière dont les registres accèdent aux données est appelée : **mode d'adressage**.

Le programme est réalisé avec **3 instructions** :

Instruction n°	Description	Instruction 6800	Opération symbolique
1	**Charger** la valeur (**opérande**) située à la position mémoire **v1** dans l'accumulateur **A.**	**ldaa** *v1*	[A] ← [*v1*]
2	**Ajouter** le contenu de l'accumulateur **A** à la valeur située à la position mémoire **v2** (le résultat de l'opération est automatiquement placé dans A)	**adda** *v2*	[A] ← [A] + [*v2*]
3	**Stocker** le contenu de l'accumulateur **A** à la position mémoire **v3.**	**staa** *v3*	[*v3*] ← [A]

☛ **Complétez** le code source comme ci-dessous.



Lecture d'une **opération symbolique**
[] ⇔ contenu de
Ex : [v1] : contenu de la mémoire à l'adresse v1

← : placer dans
Ex : [A] ←[v1] : placer le contenu de la mémoire à l'adresse v1 dans le registre A

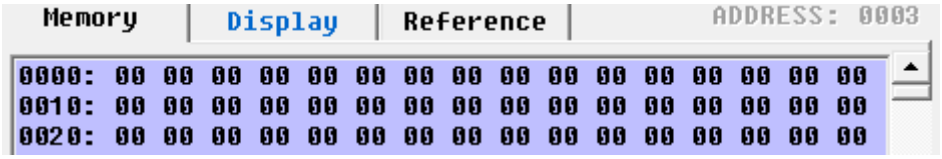
Remarque : dans **ce** programme, les instructions **ldaa** v1, **adda** v2 et **staa** v3 mettent en œuvre le mode d'**adressage étendu**.

Adressage étendu

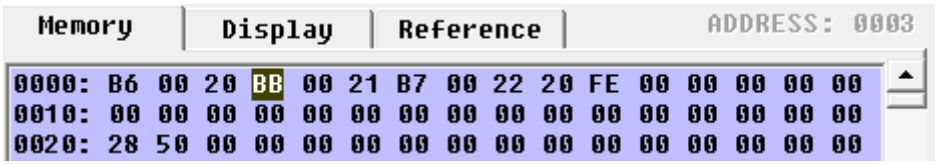
Dans le mode d'**adressage étendu**, l'adresse contenue dans le deuxième octet de l'instruction est utilisée comme octet supérieur de l'adresse de l'opérande. Le troisième octet de l'instruction est utilisé comme octet inférieur de l'adresse de l'opérande. Il s'agit d'une **adresse absolue** dans la mémoire. ldaa *v1*, adda *v2* et staa *v3* sont codées sur **3 octets**.

Exemple : ldaa $1000 charge le contenu de la mémoire située à l'adresse 1000_{16} dans l'accumulateur A.

- **ÉTAPE 3 - Assembler le code source**
  Le code source écrit dans l'étape 2 n'a pas encore été assemblé. La mémoire est "vide" comme dans l'exemple ci-dessous.



  Cliquez sur le bouton Step pour effectuer l'assemblage. La mémoire doit se remplir comme ci-dessous.



- **ÉTAPE 4 - Tester le programme en mode pas à pas**

- Documentation “Simulateur 6800”.

Travail demandé

Comlétez le document réponse du TP. (voir prof)