



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



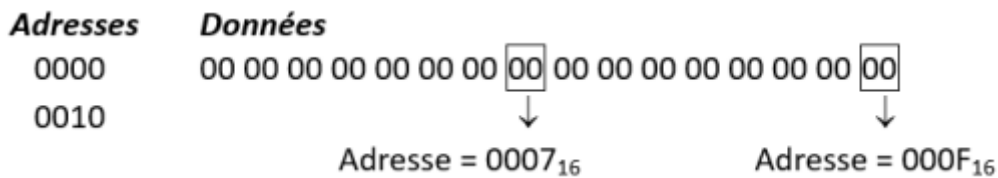
- **Les registres du MC6800**

- **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**

Code source en assembleur 6800/11

The screenshot shows the simulator interface with the following components:

- Assembly Program:**

```

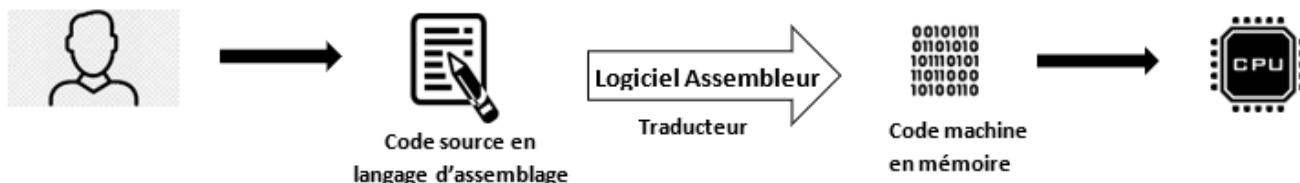
001 ;-----
002 ; Sum 1 to 10
003 ;-----
004 ten .equ 10
005 sum10 clra ;A = 0
006 ldab #ten ;B = 10
007 loop1 aba ;A = A + B
008 decb ;B--
009 tstb ;test (B==0)
010 bne loop1 ;repeat if B!=0
011 staa sum ;sum = A
012 ;-----
013 ; Sum Array values, zero terminate
014 ;-----
015 summary ldx array ;X = array
016 clra ;A = 0
017 clrb ;B = 0
018 loop2 adda 0,x ;A += array[0]
019 inx ;next item
                
```
- Memory Display:** A matrix showing addresses from 0000 to 0130 and their corresponding hex values. The value 'C6' is highlighted at address 0002.
- Registers:**
 - PC: 0002 X: 0000
 - SP: F000 IR: 004F
 - ACCUMULATOR: A: 00 B: 00
 - Status Flags: 0 0 0 1 0 0 (H I N Z U C)
- Base Converter:** A keypad for converting between Hex, Dec, and Bin.
- Control Panel:** Buttons for Clear, Load, Save, Step, and Run.
- Tooltip:** "Program Line # Break Disabled Program Line # PC Address = SP Address = X Register = Accumulator A = Accumulator B = Memory Changed"

3. Langage d'assemblage et code machine

ldx #001D
 Instruction en
 Langage d'assemblage 6800

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

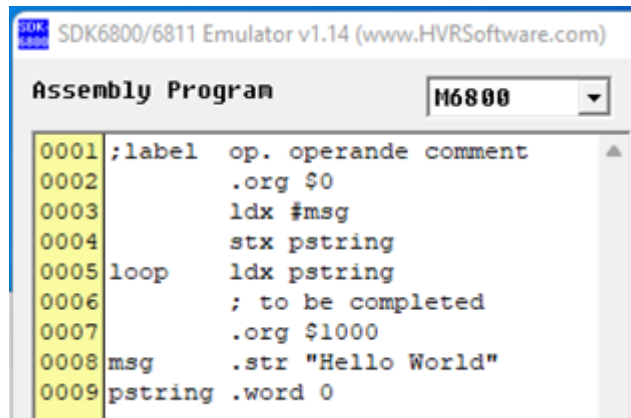
par un être humain. Un **traducteur** transforme ce langage vers le langage machine équivalent : c'est l'**assembleur**.



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 1000₁₆ et 100A₁₆)
- 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

$v3 \leftarrow v1 + v2$ # 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

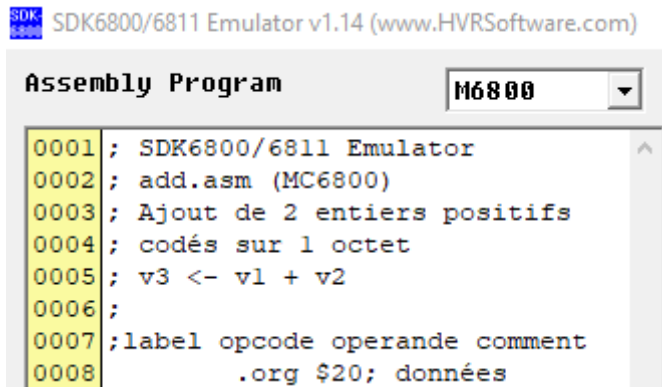
Pour effectuer ces opérations, nous allons utiliser des directives d'assemblage et des étiquettes.

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 20_{16} comme ci-dessous. Le symbole \$ signifie que la valeur qui suit est en **hexadécimal** (base 16).



```
SDK-6800 SDK6800/6811 Emulator v1.14 (www.HVRSoftware.com)
Assembly Program M6800
0001 ; SDK6800/6811 Emulator
0002 ; add.asm (MC6800)
0003 ; Ajout de 2 entiers positifs
0004 ; codés sur 1 octet
0005 ; v3 <- v1 + v2
0006 ;
0007 ;label opcode operande comment
0008          .org $20; données
```

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).

The screenshot shows the SDK6800/6811 Emulator v1.14 interface. The 'Assembly Program' window displays the following code:

```

0001 ; SDK6800/6811 Emulator
0002 ; add.asm (MC6800)
0003 ; Ajout de 2 entiers positifs
0004 ; codés sur 1 octet
0005 ; v3 ← v1 + v2
0006 ;
0007 ;label opcode operande comment
0008 |         .org $20; données
0009 v1      .byte 40
  
```

Below the code, a table shows memory contents:

Adresse	Donnée
\$0020	00

An arrow points from the text 'L'étiquette v1 est une adresse en mémoire.' to the '\$0020' address in the table.

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 $v3 \leftarrow v1 + v2$



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.

```

SDK6800/6811 Emulator v1.14 (www.HVRSoftware.com)
Assembly Program M6800
0001 ; SDK6800/6811 Emulator
0002 ; add.asm (MC6800)
0003 ; Ajout de 2 entiers positifs
0004 ; codés sur 1 octet
0005 ; v3 <- v1 + v2
0006 ;
0007 ;label opcode operande comment
0008 .org 0 ; programme
0009 debut ldaa v1;[A] <- [v1]
0010 adda v2;[A] <- [A]+[v2]
0011 staa v3;[v3] <- [A]
0012 fin bra fin
0013
0014 .org $20; données
0015 v1 .byte 40
0016 v2 .byte 80
0017 v3 .byte 0
0018 .end

```

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

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

Memory	Display	Reference	ADDRESS: 0003
0000:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0010:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0020:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

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

Memory	Display	Reference	ADDRESS: 0003
0000:	B6 00 20 BB 00 21 B7 00 22 20 FE 00 00 00 00 00		
0010:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
0020:	28 50 00 00 00 00 00 00 00 00 00 00 00 00 00		

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

- Consultez la documentation suivante : "[Simulateur 6800](#)" pour mettre en oeuvre le simulateur.

Travail demandé

Complétez le document réponse.

From:

<https://webge.fr/dokuwiki/> - **WEBGE Wikis**

Permanent link:

<https://webge.fr/dokuwiki/doku.php?id=info:prog:ass6800&rev=1730366093>

Last update: **2024/10/31 10:14**

