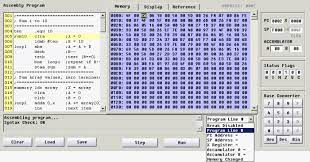
|  |  |  |
| --- | --- | --- |
| **Logo_Lycée** | **Langage machine et assembleur**  **(Assembleur 68xx)** | logo%20ac%20orl%E9ans%20toursDescription : Description : Description : pemDescription : Description : Description : pem |
| **TP**  **Approfondissement** | **Classe :** \_\_\_\_\_\_\_\_  **Nom :** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Prénom :** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |



***Figure 1*** *: Simulateur SDK6800-6811*

**Ressource :** Wiki «[Programmer en assembleur 6800 – 6811](https://webge.fr/dokuwiki/doku.php?id=info:prog:ass6800)»

**Mots-clés :** adresse, donnée, instruction-machine, mnémonique, opcode, opérande, directive d’assemblage, mode d’adressage, assembleur, variable, constante, opération symbolique.

**Capacité attendue**

Dérouler l’exécution d’une séquence d’instructions simples du type langage machine.

**Table des matières**

[Préparation 2](#_Toc182218914)

[A. Multiplication en machine 2](#_Toc182218915)

[B. Rôle du bit v du registre d’état (Status Register) 2](#_Toc182218916)

[C. Instructions de rupture de séquence 3](#_Toc182218917)

[D. Pour aller plus loin : sous-programme 5](#_Toc182218918)

[Annexe 6](#_Toc182218919)

**Table des illustrations**

[***Figure 1*** *: Simulateur SDK6800-6811* 1](#_Toc182212824)

[***Figure 2*** *: Multiplication en assembleur 6811* 2](#_Toc182212825)

[***Figure 3*** *: chaîne dans la mémoire* 3](#_Toc182212826)

[***Figure 4*** *: écran du simulateur SDK6800/11* 4](#_Toc182212827)

[***Figure 5*** *: appel et retour de sous-programme* 5](#_Toc182212828)

**MATÉRIELLES ET LOGICIEL**

Micro-ordinateur, simulateur SDK-6800 (téléchargement)

<https://webge.fr/doc/logiciels/6800IDE.zip>

**CONDITION DE RÉALISATION**

1 élève par poste informatique

**PAGE DE PRÉSENTATION DU TP**

<https://bit.ly/3EtqG6w>

**RESSOURCES**

Document "Organisation du dossier de travail" (en pochette). Annexe - Instructions utilisées dans les programmes.

6800 Instruction Set

<https://webge.fr/6800.html>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

****



## Préparation

Consultez le document en pochette : "**Organisation du dossier de travail**".

## A. Multiplication en machine

***Objectif*** *: effectuer l’opération* ***v3 ← v1 \* v2*** *en assembleur 6800*

*#* *Les variables v1 et v2 sont des octets codés en binaire naturel.*

*#* *La variable v3 est un mot sur 16bits, 0 ≤ v3 ≤ 216 - 1*

*#* *v3 est constituée de 2 octets* ***v3H*** *(octet de poids fort) et* ***v3L*** *(octet de poids faible).* ***H*** *pour High et* ***L*** *pour Low.*

1. **Ouvrez** le fichier ***mul.asm*** dans le simulateur **SDK-6800/6811. Placez** le simulateur en **mode 6811.** La description des instructions est donnée en **annexe.**
2. **Testez** le programme avec **v1 = 1010** et **v2 = 4010.**

On utilise le simulateur en mode [6811](https://www.cs.uaf.edu/2007/fall/cs441/proj1notes/sawyer/inst.html) [https://**bit.ly/3ditlC2**] pour effectuer la **multiplication**. Le 6800 n’en avait pas !

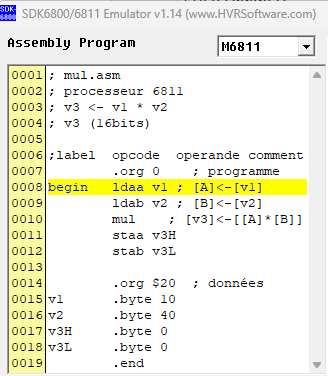
SYNTAX Mode BYTES CODE CYCLES SYMBOLIC OPERATION

mul **INH** 1 $3D 10 [A] ← ([A] \* [B])/256

[B] ← ([A] \* [B])\*256

**3.** Le résultat sur 16bits de l’opération est constitué avec les registres A et B.

Valeur du mot AB ? **AB16 =** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**

***Figure 2*** *: Multiplication en assembleur 6811*

mul.asm

|  |  |
| --- | --- |
| **Informations** | **INH** signifie **implicite** (*inhérent*)  Dans le **mode d'adressage implicite**, l'instruction donne l'adresse de manière implicite (à un pointeur de pile, à un registre d'index, etc.). Les instructions implicites sont utilisées lorsqu'aucun opérande ne doit être récupéré. Pour les processeurs 6800 et 6811, ce sont des instructions sur 1 octet. |

**+1**

-128 0 127

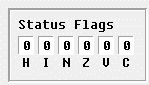
**Débordement arithmétique ⇒ V = 1**

**-1**

**4.** **Convertissez** 19016 en décimal.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**5.** **Conclusion** : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**Figure**  : registre d'état du simulateur SDK6800 - 6811

## B. Rôle du bit v du registre d’état (Status Register)

Rappel : les bits du registre d’état servent d’**indicateurs** (également appelés « drapeaux » ou ***flags***) sur l’état du processeur.

|  |  |
| --- | --- |
| **Informations** | **Le bit v (overflow)**  Le bit v passe à 1 lorsque le résultat n'est pas juste en arithmétique signée. |

1. **Chargez** le programme ***add.asm***dans le simulateur, **exécutez-le** en mode pas-à-pas et complétez le tableau ci-dessous.

add.asm

**v1=12710, v2=110 →**

**v1=-12810, v2=-110 →**

|  |  |  |
| --- | --- | --- |
| **ldaa v1** | **adda v2** | **staa v3** |
| x | x | x  X: non pris en compte |

1. Quelles précisions sont apportées sur le résultat de l’opération par les des bits **N, Z, V** et **C** après l’exécution de *adda v2* ?

**v1=127, v2=1** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**v1=-128, v2=-1** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## C. Instructions de rupture de séquence

|  |  |
| --- | --- |
| **Informations** | Les **ruptures de séquence** sont les **sauts**, les **branchements** et les **appels et retours de procédure**. <https://webge.fr/6800.html>.    ***Figure 3*** *: chaîne dans la mémoire* |

**1. Chaîne de caractères**

***Objectif****:* *détecter la fin d'une chaîne de caractères.*

|  |  |
| --- | --- |
| **Informations** | En informatique, une **chaîne de caractères** est conceptuellement une suite ordonnée de **caractères** et physiquement une suite ordonnée d'unités de code. |

En assembleur 6800, la fin d'une chaîne de caractères est identifiée par le caractère **ASCII NULL** (**0x00**). Il est automatiquement ajouté à la fin de la chaîne par l'assembleur lorsqu’elle est déclarée avec la directive **.str**.

Exemple : *001D* 48 65 6C 6C 6F 20 57 6F 72 6C 64 **00** msg .str "Hello World"

|  |  |
| --- | --- |
| **Informations** | Le caractère de fin de chaîne facilite la manipulation d’une chaîne lors de son transfert de la mémoire centrale vers la mémoire de l’écran. Le code ci-dessous parcourt la chaîne "*Hello*" tant qu’un caractère **NULL** n'est pas détecté. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithme** FinChaine  **Variables**  msg <- "Hello" : chaîne de caractères  **Registres**  X : registre 16bits, adresse d'un caractère de la chaîne msg  A : registre 8bits, valeur pointée par X  **début**  X <- adresse du premier caractère de msg  **faire**  A <- \*X // \*X est la valeur pointée par X  X <- X+1  **tant que** (A ≠ 0)  **fin** | *;label opcode operand comment*  .org 0 ; **programme**  start ldx #msg ; on charge l'adresse du premier caractère  ; de la chaîne msg dans le registre x  loop ldaa 0,X ; [A] ← [0 + [X]]  cmpa #0 ; [A] - 0  **beq** fol ; **si** [A]=0 (Z=1) **alors** branchement à fol  inx ; **sinon** [X] ← [X] + 1  **bra** loop ; branchement inconditionnel à l’étiquette loop  fol nop ; suite du programme  .org $20 ; d**onnées**  msg .str "Hello" ; une chaîne de caractère est entourée de ""  .end ; fin du code source | |  |  |  | | --- | --- | --- | | Adresse16 | Code16 ASCII |  | | 0020 | 48 | '**H**' | | 0021 | 65 | '**e**' | | 0022 | 6C | '**l**' | | 0023 | 6C | '**l**' | | 0024 | 6F | '**o**' | | 0025 | **00** | NULL | |

1. **Complétez l’algorigramme** correspondant au programme *FinChaine* avec les **instructions** en **assembleur**.

bra loop

Début Finchaîne

[X] ← msg

[A] ← [0 + [X]]

[A] = 0

[X] ← [X] + 1

ldx #msg

Suite prog

ldaa 0,x

|  |  |
| --- | --- |
| **Informations** | **Adressage indexé**. L’instruction **ldaa 0**,**x** fait référence au registre d’index X pour calculer l’adresse de la donnée à charger dans A. Un registre d'index participe au calcul de l'adresse d'un opérande durant l'exécution d'un programme, par exemple pour faire des **opérations répétitives sur les éléments d'un tableau**. |

1. **Test 1** : analyse **du premier passage dans la boucle** du programme *FinChaine*

Les instructions ***cmpa***, ***beq*** et ***nop*** sont données en **annexe**. **Chargez** le programme ***finchaine.asm***dans le simulateur, **exécutez-le** en mode pas-à-pas et complétez le tableau ci-dessous lorsque [A] = 4816.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Adresses16 | **opcode**16 et opérande16 | [label] [**opcode**] [operande] | X16 | A16 | Status Flag Z | PC16 |
| *0000* | *0000* | *0* | *0000* |
| 0000 | **CE** 00 20 | start **ldx** #msg | 0020 | 0000 | 0 |  |
| 0003  ***finchaine.asm*** | **A6** 00 | loop **ldaa** 0,x | **48** |  |  |
| 0005 | **81** 00 | **cmpa** #0 |  |  |
| 0007 | **27** 03 | **beq** fol |  |  |
| 0009 | **08** | **inx** | 0021 |  |  |
| 000A | **20** F7 | **bra** loop |  |  |
| 000C | **01** | fol **nop** |  |  |  |  |
|  |  | .org $20 |  |  |  |  |
| 0020 | 48 65 6C 6C 6F 00 | msg .str "Hello" |  |  |  |  |

1. A quelle adresse se poursuit le programme lors de l’exécution de ***bra*** *loop* ?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| **Informations** | L’assembleur calcule le saut *disp* (ici F716) à partir de l’adresse de l’instruction ***bra*** *loop* (@départ) et de l’adresse de l’étiquette *loop (*@arrivée*)*. Le tableau de l’annexe précise l’effet de *bra disp* sur le compteur programme : [[PC](https://webge.fr/6800.html#PC-reg)] [←](https://webge.fr/6800.html#transfer-desc) [[PC](https://webge.fr/6800.html#PC-reg)] + [disp](https://webge.fr/6800.html" \l "disp-desc) + 2.  Comment l’assembleur a-t-il obtenu disp=F716 ?  [[PC](https://webge.fr/6800.html#PC-reg)] [←](https://webge.fr/6800.html#transfer-desc) [[PC](https://webge.fr/6800.html#PC-reg)] + [disp](https://webge.fr/6800.html" \l "disp-desc) + 2 peut s’exprimer sous la forme @arrivée = @départ + disp + 2 donc disp = @arrivée - @départ - 2 => disp = 000316 - 000A16 - 216  Soit disp = 310 - 1010 – 210 = -910. Or 910=000010012 donc -910 = 111101102 + 1 = 111101112 = **F716** |

1. **Calculez** l’adresse contenu dans le registre PC après l’exécution de ***beq*** *fol*.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Test 2** : analyse du programme lorsqu’il a atteint le caractère de fin de chaîne

**Remplacez** *ldx #msg* par *ldx #$25 pour que le programme pointe sur le caractère de fin de chaîne* ***00*** *à l’adresse* ***$0025***. Exécutez le programme en mode pas-à-pas et c**omplétez** le tableau ci-dessous.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Adresses16 | **opcode**16 opérand16 | [label] [**opcode**] [operand] | X16 | A16 | Status Flag Z | PC16 |
| *0000* | *00* | *0* | *0000* |
| 0000 | **CE** 00 20 | start **ldx** #msg | **0025** | 00 | 0 | 0003 |
| 0003 | **A6** 00 | loop **ldaa** 0,x |  |  |  |
| 0005 | **81** 00 | **cmpa** #0 |  |  |  |
| 0007 | **27** 03 | **beq** fol |  |  |  |
| 0009 | **08** | **inx** |  |  |  |  |
| 000A | **20** F7 | **bra** loop |  |  |  |  |
| 000C | **01** | fol **nop** | **0025** |  |  |  |
| 000D | **01** | **nop** |  |  |  |  |
|  |  | .org $20 |  |  |  |  |
| 0020 | 48 65 6C 6C 6F 00 | msg .str "Hello" |  |  |  |  |

1. Que fait le programme lorsqu’il a détecté lecaractère **ASCII NULL ?**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

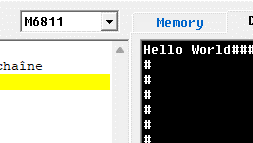
Le programme *FinChaine* peut être optimisé en supprimant l’instruction *cmpa #0* car **Z=1** (condition de branchement) dès que [A]=0.

1. **Supprimez** *cmpa #0*et vérifiez le fonctionnement du programme.

**2. Transfert d’une zone mémoire vers une autre**

***Objectif****:* *afficher sur l’écran du simulateur une chaîne de caractères située dans la zone mémoire des données.*

**Présentation**



***Figure 4*** *: écran du simulateur SDK6800/11*

Le programme *Finchaine* a été modifié pour afficher la chaîne de caractères "***Hello World***" sur l’écran du simulateur. *Finchaine* a été renommé en ***hello***. La programmation a été simplifiée par l’utilisation d’un processeur **M6811** qui dispose d’un deuxième **registre d’index** : **Y**.

Les instructions suivantes ont été ajoutées :

|  |  |
| --- | --- |
| Opérations symboliques | Instructions assembleur 6811 |
| [Y] ← [Y] + 1 | iny |
| [Y] ← #$fb00 | ldy #$fb00 |
| [0 + [Y]] ← [A] | staa 0,y |

L’écran du simulateur se compose de **20 lignes** de **54 caractères**. L’adresse du premier caractère (en haut et à gauche de l’écran) est **FB00**16. L’écran est visible en sélectionnant l’onglet ***Display***. Le programme transfère la chaîne "Hello" vers la mémoire de l’écran.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mémoire pointée par : **X** |  |  | Mémoire pointée par : **Y** | Ecran |
| $0020 | H | A copier vers-> | $FB00 | H |
| $0021 | e | A copier vers-> | $FB01 | e |
| $0022 | l | A copier vers-> | $FB02 | L |
| $0023 | l | A copier vers-> | $FB03 | L |
| $0024 | o | A copier vers-> | $FB04 | 0 |

1. **Chargez** le fichier ***hello.asm***dans le simulateur. **Complétez l’algorigramme** correspondant au programme ***hello*** avec les **instructions** en **assembleur**.

hello.asm

ldy #$fb00 ; adresse du 1er caractère de l’écran

Début

[X] ← #msg

[A] ← [0 + [X]]

[A] = 0

[0 + [Y]] ← [A]

ldx #msg

[Y] ← #bscreen

[X] ← [X] + 1

[Y] ← [Y] + 1

Suite prog

1. **Affichez** l’écran en cliquant et testez le fonctionnement du programme.

**3. Remplir une zone mémoire avec un caractère ASCII**

***Objectif****:* *effacer l’écran de l’émulateur.*

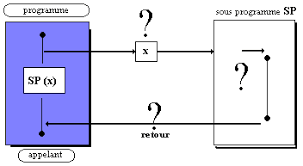
cls.asm

Le code ci-dessous peut effacer (partiellement ou totalement) l'écran de l’émulateur. le fichier ***cls.asm*** et testez-le.

|  |  |
| --- | --- |
| *label opcode operand comment]*  ; Constantes (@ ⬄ adresse)  bscreen .equ $FB00 ; @ de la première position de l'écran  escreen .equ $FB14 ; @ de la dernière position à effacer  space .equ $20 ; code ASCII du caractère espace  .org 0  cls ldaa #space  ldx #bscreen  loop staa 0,x  inx  cpx #escreen  bne loop  .end | **Algorithme** clearScreen  **Constantes**  bscreen ← FB0016, adresse de la première position de l'écran : mot 16bits  escreen ← adresse de la dernière position à effacer : mot 16bits  space ← 2016 : code ASCII du caractère espace : octet 8bits  **Registres**  X <- bscreen : registre 16bits  A <- space : registre 8bits  **début**  A <- space  X <- bscreen  **faire**  \*X <- A // \*X est la valeur pointée par X  X <- X+1  **tant que** (X ≠ escreen)  **fin** |

**1.** **Modifiez** le programme pour qu'il efface la première ligne de l’écran. Notez ci-dessous les modifications apportées.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



***Figure 5*** *: appel et retour de sous-programme*

## D. Pour aller plus loin : sous-programme

***Objectif****:* *effacer l’écran de l’émulateur et afficher "Hello World".*

|  |  |
| --- | --- |
| **Informations** | **En informatique, un sous-programme est** un programme secondaire pouvant être utilisé en plusieurs points d'un programme principal. |

Le programme précédent peut être transformé en un **sous-programme** en le terminant avec l’instruction **RTS**. Ceci a été réalisé dans le fichier ***cls2.asm.*** *Chargez-le*dans le simulateur. Vous constaterez que le code source comprend deux parties : le sous-programme identifié par l’étiquette ***cls*** et se terminant avec l’instruction **rts** et le programme principal identifié par l’étiquette ***main***.

**1.** **Testez** le programme en mode pas à pas puis **modifiez**-le pour qu’il efface la première ligne et la remplace par le texte : *Hello World*.

Appel prof

Pour faire vérifier le fonctionnement

## Annexe – Description des instructions utilisées dans les programmes

Le tableau ci-dessous est extrait de la source : <https://webge.fr/6800.html> . Il permet de connaître l’organisation d’une instruction, son rôle et sa syntaxe en assembleur **MC6800 / 6811**.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Opcode**  (Base 16) | **Nombre d’octets (Opcode + Opérande(s))** | **Syntaxe assembleur de l’instruction** | **Opération symbolique** | **Description** | **Remarque** |
| **27** | 2 | **BEQ** disp | (Z == 1) ? {[[PC](http://www.8bit-era.cz/6800.html#PC-reg)] [←](http://www.8bit-era.cz/6800.html#transfer-desc) [[PC](http://www.8bit-era.cz/6800.html#PC-reg)] + [disp](http://www.8bit-era.cz/6800.html" \l "disp-desc) + 2} | **Si** l’instruction précédente a produit un résultat nul **alors** PC= PC + disp + 2 **sinon** PC=PC+2.  **Attention**, disp est codé en complément à 2. | **B**ranch if **E**qual to zero |
| **20** | 2 | **BRA** disp | [[PC](https://webge.fr/6800.html#PC-reg)] [←](https://webge.fr/6800.html#transfer-desc) [[PC](https://webge.fr/6800.html#PC-reg)] + [disp](https://webge.fr/6800.html" \l "disp-desc) + 2 | Saut inconditionnel à la position PC + disp + 2.  **Attention**, disp est codé en complément à 2. | **BR**anch **A**lways |
| **81** | 2 | **CMPA** #data8 | [A] – data8 | Compare le contenu de l’accumulateur avec data8 | **C**o**MP**are **A** with data8 |
| **08** | 1 | **INX** | [X] ← [X] + 1 | Incrémente le registre d’index X. | **IN**crement **X** |
| **B6** | 3 | **LDA**[**A**](http://www.8bit-era.cz/6800.html#A-reg) addr16 | [A] ← [addr16] | Charge l’accumulateur A avec la donnée sur 8 bits (opérande) située à la position addr16. | **L**oad **A**ccumulateur **A** from memory |
| **F6** | 3 | **LDA**[**B**](http://www.8bit-era.cz/6800.html#A-reg) addr16 | [B] ← [addr16] | Charge l’accumulateur B avec la donnée sur 8 bits (opérande) située à la position addr16. | **L**oad **A**ccumulateur B from memory |
| **CE** | 3 | **LDX** #addr16 | [[X](https://webge.fr/6800.html#X-reg)[(HI)](https://webge.fr/6800.html#HI-desc)] [←](https://webge.fr/6800.html#transfer-desc) data16[(HI)](https://webge.fr/6800.html#HI-desc), [[X](https://webge.fr/6800.html#X-reg)[(LO)](https://webge.fr/6800.html#LO-desc)] [←](https://webge.fr/6800.html#transfer-desc) data16[(LO)](https://webge.fr/6800.html#LO-desc) | Charge le registre d’index X avec la donnée sur 16bits (opérande) data16. | **L**oad**D** the index register **X** |
| **01** | 1 | **nop** |  |  | **N**o **O**peration |
| **B7** | 3 | **STAA** addr16 | [addr16] [←](http://www.8bit-era.cz/6800.html#transfer-desc) [A] | Sauvegarde le contenu de A à l’adresse addr16. | **ST**ore **A**ccumulator **A** in Memory |
| **F7** | 3 | **STAB** addr16 | [addr16] [←](http://www.8bit-era.cz/6800.html#transfer-desc) [B] | Sauvegarde le contenu de B à l’adresse addr16. | **ST**ore **A**ccumulator **B** in Memory |

**Seulement MC6811 (**[6811 instructions set](https://www.cs.uaf.edu/2007/fall/cs441/proj1notes/sawyer/inst.html)**)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Opcode**  (Base 16) | **Nombre d’octets (Opcode + Opérande(s))** | **Syntaxe assembleur de l’instruction** | **Opération symbolique** | **Description** | **Remarque** |
| **3D** | 1 | **MUL** | [D] ← [A] \* [B] | D est constitué de A et B tel que  D(HI) = A et D(LO)=B. | **MUL**tiplication **A** with **B** |

**←** : **affectation** (la donnée ou l’adresse est transférée dans la direction de la flèche). **[...]** : contenu de ...

**$**: la valeur qui suit l’**opcode** est en hexadécimal. **disp** : déplacement d’adresse **signé sur 8 bits**.

**#** : la valeur qui suit l’**opcode** est une donnée (en l’absence de # c’est une adresse). **addr16** : adresse codée sur 16bits.

**data8** : donnée codée sur 8bits.