|  |  |  |  |
| --- | --- | --- | --- |
| **Logo_Lycée** | **Prise en main du simulateur SDK6800/6811** | | logo%20ac%20orl%E9ans%20toursDescription : Description : Description : pemDescription : Description : Description : pem |
| **TP**  **Découverte** | **Document réponse** | **CORRECTION** |

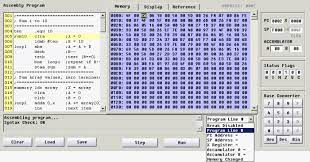


Figure 1 : SDK6800-6811

**Sujet :** 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.

**Compétence**

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

**Rappel de l’objectif du bloc de code** : ajouter 2 entiers positifs codés sur 1 octet (v3 ← v1 + v2)

1. **Correspondance entre le code machine et le code source**
2. En mode pas à pas, observer le déplacement du curseur dans la mémoire et en déduire le nombre d’octets utilisés par chaque instruction. Complétez le tableau ci-dessous.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instruction**  **N°** | **Adresses16** | **Opcode16 et opérande16** | **[label] [opcode] [operand]** | **Opérations symboliques** |
|  |  |  | **.org 0 ; programme** |  |
| **1** | **0000** | **B6 00 20** | debut ldaa $**20** | [A] ← [2016] |
| **2** | **0003** | **BB 00 21** | adda $**21** | [A] ← [A] + [2116] |
| **3** | **0006** | **B7 00 22** | staa $**22** | [2216] ← [A] |
| **4** | **0009** | 20 FE | fin bra fin |  |
|  |  |  | **.org $20 ; données** |  |
|  | **0020** | **28** | v1 .byte 40 |  |
|  | **0021** | **50** | v2 .byte 80 |  |
|  | **0022** | 00 | v3 .byte 0 |  |

1. Quels octets codent les mnémoniques *ldaa, adda* et *staa* ? **ldaa (B616), adda (BB16) et staa (B716)**
2. **Évolution du contenu de l’accumulateur A et du registre d’état pendant l’exécution du programme**
3. Exécutez les instructions suivantes en mode pas à pas et complétez les **registres A** et **Status Flag**.

|  |  |  |
| --- | --- | --- |
| **ldaa v1** | **adda v2** | **staa v3** |
| **0 0 0 0**  **28** | **0 0 0 0**  **78** | **0 0 0 0**  **78**  X: non pris en compte |

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

**Le résultat de l’opération est positif (N = 0) et non nul (Z = 0). L’addition ne génère pas de retenue (C = 0).**

1. Quelle valeur a été mémorisée à l’adresse 2216 lors de l’exécution de ***staa v3*** ? **7816**
2. Est-ce le résultat attendu ? Pourquoi ?

**Oui, v3 = 2816 + 5016 = 7816**

1. Refaites le test avec **v1 ← C816** et **v2 ← 6916** et complétez les registres.

|  |  |  |
| --- | --- | --- |
| **ldaa v1** | **adda v2** | **staa v3** |
| **1 0 0 0**  **C8** | **0 0 0 1**  **31** | **0 0 0 1**  **31** |

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

**Le résultat de l’opération est positif (N = 0) et non nul (Z = 0). L’addition génère une retenue (C = 1) car [A] > FF16.**

1. Quelle valeur a été mémorisée à l’adresse 2216 lors de l’exécution de ***staa v3*** ? **3116**
2. Est-ce le résultat attendu ? Pourquoi ? Proposez une solution ?

**Non, v3 = C816 + 6916 = 13116 ([A] > FF16).**

**Solution : mémoriser v3 dans un mot de 16bits**

**Début**

**Fin**

[A] ← v1

[A] ← [A] + v2

**[v3L] ← [A]**

**C=1**

**[A] ← 1**

**[v3H] ← [A]**

**[A] ← 0**

1. **Modification du programme**

Le résultat de l’opération est maintenant placé dans un mot de 16bits constitué de deux octets : v3H et v3L. On propose l’algorigramme ci-contre.

1. Complétez-le avec les opérations symboliques suivantes :

[v3H] ← [A], [A] ← 1, C = 1, [v3L] ← [A], [A] ← 0

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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opcode**  (Base 16) | **Nombre d’octets** | **Syntaxe assembleur de l’instruction** | **Opération symbolique** | **Remarque** |
| **BB** | 3 | **ADDA** addr16 | [A] ← [A] + [addr16] | Add Memory contents to the Accumulator |
| **25** | 2 | **BCS** disp | (C == 1) ? {[PC] [←](https://webge.fr/6800.html#transfer-desc) [PC] + disp + 2} | Branch if carry set |
| **09** | 2 | **BRA** disp | [PC] ← [PC] + disp + 2 | Unconditional branch |
| **B6** | 3 | **LDAA** addr16 | [A] ← [addr16] | Load Accumulateur A |
| **86** | 2 | **LDA**[**A**](http://www.8bit-era.cz/6800.html#A-reg) #[data8](http://www.8bit-era.cz/6800.html#data8-desc) | [A] ← data8 |
| **01** | 1 | **NOP** |  | No Operation |
| **B7** | 3 | **STAA** addr16 | [addr16] [←](http://www.8bit-era.cz/6800.html#transfer-desc) [A] | Store Accumulator in Memory |

**←** : **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.

1. Enregistrez le fichier *add.asm* sous *addwd.asm* sur le serveur. Codez *addwd.asm* conformément à l’algorigramme en utilisant les informations données dans le tableau ci-dessus.

.**org $20; données**

v1 .byte $C8

v2 .byte $69

v3H .byte 0

v3L .byte 0

.end

*Résultat attendu avec* **v1 ← C816** et **v2 ← 6916***.*

