Lycée Polyvalent PIERRE EMILE MARTIN	**Langage machine et assembleur**	académie d'Orléans-Tours
TP1 **Découverte**	**Programmation en Assembleur 6800** Opérations arithmétiques et transferts mémoire	

Objectifs du TP

- Écrire et tester des programmes en assembleur 6800
- Manipuler les registres et la mémoire
- Comprendre l'exécution pas à pas d'un programme
- Analyser le contenu de la mémoire après exécution

Prérequis : Avoir complété le TD1 - Découverte Assembleur 6800

Date : _____ **Classe** : _____

Nom : _____

Prénom : _____

Rappels

Instructions de base du MC6800

Instruction	Opcode (Extended)	Signification	Cycles
ldaa adr	B6	Charger A depuis la mémoire	5
ldab adr	F6	Charger B depuis la mémoire	5
staa adr	B7	Stocker A en mémoire	6
stab adr	F7	Stocker B en mémoire	6
adda adr	BB	[A] ← [A] + [adr]	5
addb adr	FB	[B] ← [B] + [adr]	5
suba adr	B0	[A] ← [A] - [adr]	5
subb adr	F0	[B] ← [B] - [adr]	5

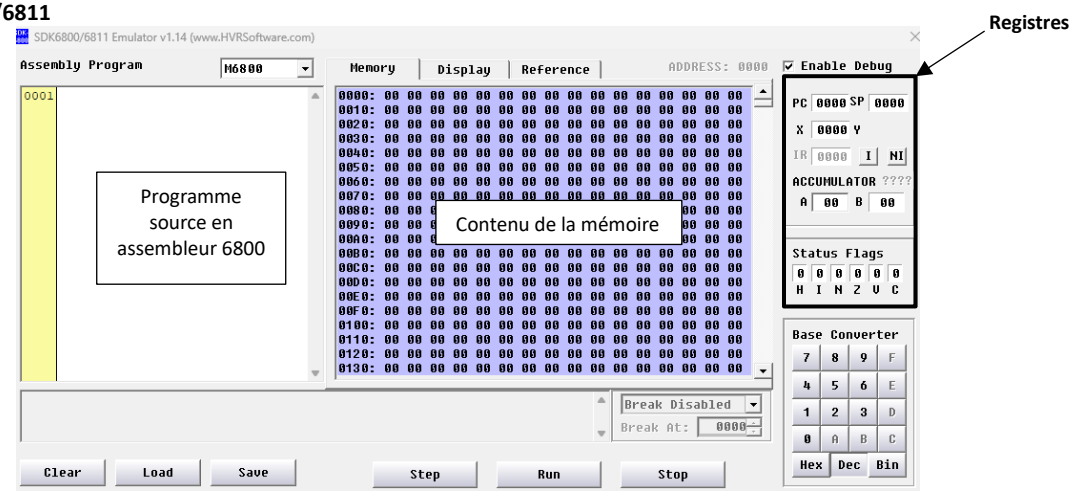
Notation : [adr] signifie "contenu de l'adresse adr"

Structure d'un programme assembleur

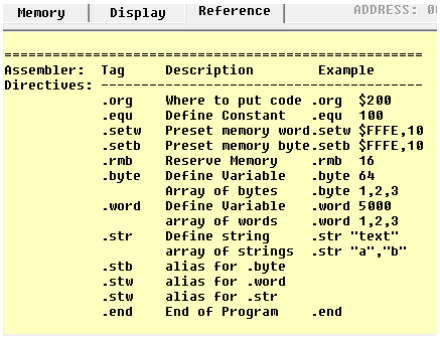
```
; [Label]   Operation  [operand]        [comment]
          .org $0000            ; Origine du programme en mémoire
; --- Instructions ---
          ldaa v1               ; Commentaire explicatif
          adda v2
          staa v3

          .org $00A0            ; Origine des données en mémoire (variables)
; --- Données ---
v1        .byte 10              ; Définition d'une variable
v2        .byte 20
v3        .byte 0
```

⚠ Remarque : **le champ operation** contient un **opcode** (ex : ldaa) ou une **directive d'assemblage** (ex .org)

Simulateur SDK6800/6811



EXERCICE 1 : Addition simple (Prise en main)



Énoncé

Écrire un programme qui calcule : **resultat ← a + b**

Avec :

- a = 25_{10}
- b = 47_{10}
- resultat initialisé à 0

Travail à réaliser

1.1 Complétez le code source suivant :

```
; [Label]    Operation  [operand] [comment]
            .org $0000              ; EX1.asm
; Instructions du programme en mémoire
            ldaa _____              ; Charger a dans A
            _____ b                 ; Ajouter b à A
            _____ resultat          ; Stocker A dans resultat

            .org $00B0              ; Données du programme (variables)
a           .byte 25
b           .byte 47
resultat    .byte 0
```

Directives d'assemblage

.org : la directive ORG indique l'adresse de départ du code assemblé.

.byte : la directive BYTE indique que la valeur qui suit est un octet.

1.2 Avant d'exécuter le programme dans l'émulateur :

- Calculez manuellement le résultat attendu en **décimal** : _____
- Convertissez ce résultat en **hexadécimal** : _____

⚠ **Préparez** votre dossier ***home*** sur le serveur en suivant la fiche **« Organisation du dossier de travail »**

1.3 Emulateur SDK6800/6811 :

- Lancez l'émulateur en cliquant sur SDK-6800
- Chargez le fichier **EX1.asm** situé sur le serveur dans **home/TP1_SDK68xx**. Saisissez votre **code source** dans l'émulateur.
- Assemblez le code avec un clic sur Step et sauvegardez le fichier sous **EX1.asm**
- Relevez le contenu de la mémoire (**code machine**) :

0000: _____ _____ _____ _____ _____ _____ _____ _____ _____

...
00B0: _____ _____ _____

Organisation du programme source

Les instructions d'assemblage contiennent les champs suivants :

←----Instruction -→
[Label] Operation [operand] [comment]

Remarque : chaque champ doit être séparé par au moins un espace.

1.4 Identifiez dans la mémoire :

- Les trois **opcodes** des instructions : _____, _____, _____
- Les adresses des variables (**opérandes**) : _____, _____, _____
- Combien d'octets occupent chaque instruction ? _____

1.5 Exécutez le programme pas à pas (un pas = une action sur Step) :

- Après ldaa a : A = _____ (en hexa)
- Après adda b : A = _____ (en hexa)
- Après staa resultat : Mémoire[00B2] = _____ (en hexa)

1.6 Le résultat correspond-il à la prédiction de la question 1.2 ? _____

EXERCICE 2 : Soustraction

Énoncé

Écrire un programme qui calcule : **diff ← x - y**

Avec :

- $x = 100_{10}$
- $y = 37_{10}$
- diff initialisé à 0

Travail à réaliser

2.1 Complétez le code source suivant :

```
        .org $0000          ; EX2.asm
                            ; Instructions du programme en mémoire
        ldaa _____
        _____ y             ; Instruction de soustraction
        staa _____


        .org $00C0          ; Données du programme (variables)
x       .byte 100
y       .byte 37
diff    .byte 0
```

2.2 Prédiction :

- Valeur de ***diff*** attendu en **décimal** : _____
- En **hexadécimal** : _____

2.3 Chargez le fichier **EX2.asm** situé sur le serveur dans **home/TP1_SDK68xx**. Saisissez votre **code source** dans l'émulateur. **Exécutez-le.**

Valeur de ***diff*** après l'exécution du programme en **hexadécimal** _____ et en **décimal** _____ ?

2.4 ⚠ Cas limite : Modifiez maintenant les valeurs :

- x = 50
- y = 80

Réassemblez et exécutez le programme. Quel résultat obtenez-vous en **hexadécimal** _____ et en **décimal** _____ ?

2.5 Expliquez ce résultat surprenant (💡 Indice : pensez à la capacité d'un registre 8 bits et aux nombres signés/non signés).

_____

_____

_____

_____

_____

EXERCICE 3 : Calcul à trois opérandes

Énoncé

Écrire un programme qui calcule : **somme ← a + b + c**

Avec

- a = 15_{10},
- b = 28_{10},
- c = 42_{10}

Travail à réaliser

3.1 Réfléchissez à l'algorithme :

- Combien d'instructions ldaa, adda, staa sont nécessaires ? _____
- Dans quel ordre devez-vous les écrire ?

  _____

3.2 Complétez le code source suivant :

```
                .org $0000          ; EX3.asm
                                    ; instructions du programme (4 instructions)
                _____________________________________________
                _____________________________________________
                _____________________________________________
                _____________________________________________

                .org $00D0          ; Données du programme (variables)
a               .byte 15
b               .byte 28
c               .byte 42
somme           .byte 0
```

3.3 Chargez le fichier **EX3.asm** situé sur le serveur dans **home/TP1_SDK68xx**. Saisissez votre **code source**. **Tracez l'évolution du registre A :**

Après l'instruction	Valeur de A (hexa)	Valeur de A (décimal)
ldaa a		
adda b		
adda c		
staa somme	(inchangé)	

Valeur de ***somme*** après l'exécution du programme en **hexadécimal** _____ et en **décimal** _____ ?

3.4. Combien d'octets occupent votre programme en mémoire ? _____

EXERCICE 4 : Utilisation des deux registres A et B

Date : _____ **Classe** : _____

Nom : _____

Prénom : _____

Énoncé

Calculez : **resultat = (a + b) - (c + d)**

Avec a = 50_{10}, b = 30_{10}, c = 20_{10}, d = 15_{10}

Stratégie :

- Utilisez le registre **A** pour calculer (a + b)
- Utilisez le registre **B** pour calculer (c + d)
- Puis faites la soustraction

Les registres A et B sont des accumulateurs. Ils servent à stocker temporairement des données, à réaliser les calculs, et à échanger des valeurs avec la mémoire ou les périphériques.

Travail à réaliser

4.1 Complétez le code source suivant :

```
                .org $0000
; Instructions du programme en mémoire
                ldaa a                  ; [A] ← [a]
                adda b                  ; [A] ← [A] + [b]
; Calcul de (c + d) dans B
                _________ c             ; [B] ← [c]
                _________ d             ; [B] ← [B] + [d]
; Soustraction A - B
; ⚠ Problème : il n'existe pas d'instruction "suba B" !
; Solution : on doit stocker B en mémoire temporaire
                _________  temp         ; [temp] ← [B]
                suba temp               ; [A] ← [A] - [temp]
                staa resultat           ; [resultat] ] ← [A]

                .org $00E0      ; Données du programme (variables)
a               .byte 50
b               .byte 30
c               .byte 20
d               .byte 15
temp            .byte 0         ; Variable temporaire
resultat        .byte 0
```

4.2 Prédiction :

- (a + b) = __50 + _____ en décimal _____ en hexadécimal _____
- (c + d) = _____ en décimal _____ en hexadécimal _____
- Résultat final = _____ en décimal _____ en hexadécimal _____

4.3 Chargez le fichier **EX4.asm** situé sur le serveur dans **home/TP1_SDK68xx**. Saisissez votre **code source**. **Tracez l'évolution des registres :**

Instruction	A (hexa)	B (hexa)	Temp (hexa)
Initial	00	00	00
ldaa a			
adda b			
ldab c			
addb d			
stab temp			
suba temp			
staa resultat			

Valeur de ***resultat*** après l'exécution du programme en **hexadécimal** _____ et en **décimal** _____ ?

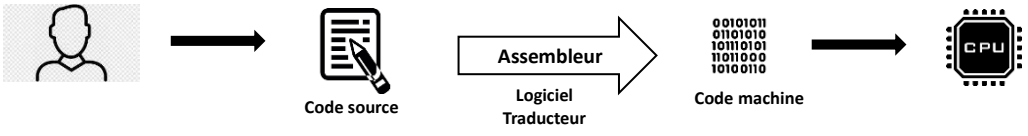
4.4. Combien d'octets occupent votre programme en mémoire ? _____

Tableau récapitulatif (à compléter)

Exercice	Opération	Octets code	Octets données	Résultat
Ex1	a + b	9 octets	3 octets	$72_{10} = 48_{16}$
Ex2	x - y	9 octets	3 octets	$63_{10} = 3F_{16}$
Ex3	a + b + c			
Ex4	(a+b) – (c+d)			

Points clés à retenir

- Le **code source** (écrit par le programmeur) n'étant pas destiné à être exécuté par le processeur, un programme de **traduction** automatique (l'**assembleur**) est nécessaire.



- Bien que plus facile à manipuler que les "codes machines", l'assembleur est **fastidieux à écrire,** car comme on le voit ci-dessus il faut aligner un grand nombre d'instructions pour obtenir un résultat, même simple. De plus, il **ne s'adresse qu'à un seul modèle de processeur**. Tout changement de machine nécessite une **réécriture** plus ou moins complète du **code**.

- Pour pallier ces défauts, des **langages évolués** comme le **C**, le **PHP** ou le **Python** ont été développés. Ils permettent au programmeur de se concentrer sur l'algorithmique des applications. Comme les instructions ne sont plus compréhensibles par l'ordinateur, une phase de traduction est nécessaire. C'est le rôle des **interpréteurs** et des **compilateurs**.

Aujourd'hui, l'assembleur reste utilisé pour écrire des parties des **systèmes d'exploitation**, **gestionnaires de périphériques,** etc.