T.P. 1
Premiers pas en assembleur 68000

Ce TP a pour objectif de vous familiariser avec les différentes étapes nécessaires à la réalisation d'un programme en assembleur 68000. Ainsi, dans un premier temps, vous étudierez le comportement de quelques programmes simples, puis, dans un second temps, vous serez capable de concevoir, d'assembler et de déboguer vos propres programmes.

Étape 1

Commençons par quelque chose de très simple afin d'acquérir les notions de base.

- **Ouvrez un nouveau fichier** dans l'éditeur de texte.
- **Sauvegardez-le** sous le nom `"Etape_01.asm"`.
- **Tapez le texte ci-dessous :**

```
            org     $4

Vector_001  dc.l    Main

            org     $500

Main        move.b  #$41,d0
            move.w  #65535,d1
            move.l  #-5,d2
```

- Essayons dans un premier temps de bien comprendre chacune de ces lignes.

  La première ligne utilise la directive d'assemblage `ORG`. Cette directive permet de préciser à l'assembleur à partir de quelle adresse seront assemblées les lignes qui suivent. Dans notre cas l'assemblage commencera donc à l'adresse `$4`.

  Il faut savoir que le 68000 réserve les 1024 premières adresses de la mémoire à ce que l'on appelle les vecteurs d'exception. Il y a 256 vecteurs et chaque vecteur contient une adresse codée sur 32 bits. La zone mémoire utilisée par les vecteurs est comprise entre l'adresse `$0` et l'adresse `$3FF`. Nous n'aborderons pas le traitement des exceptions dans ce premier TP. Cette zone mémoire ne devra donc pas être utilisée.

  Toutefois, il y a quand même un vecteur dont nous aurons besoin. Il s'agit du **vecteur 1**. Ce vecteur est situé en mémoire à partir de l'adresse `$4` et contient l'adresse du point d'entrée d'un programme. C'est l'adresse qui sera chargée dans le registre **PC** après un *reset* du 68000. Lorsque le 68000 démarre, il va donc récupérer les 32 bits situés à l'adresse `$4` et charge cette valeur dans le registre **PC**. Le 68000 peut alors procéder à l'exécution du programme.

La première chose à faire dans un code source est donc de préciser au 68000 où se trouve le point d'entrée du programme. Il faut donc écrire à l'adresse `$4`, l'adresse du point d'entrée du programme. C'est pour cette raison que notre programme démarre par la directive `ORG $4` et que celle-ci est suivie d'une directive `DC.L Main`. Cette dernière va placer l'adresse `Main` (qui est le point d'entrée de notre programme) à l'adresse $4.

Ces deux premières lignes devront être présentes dans tous vos codes sources afin d'initialiser correctement le 68000.

On pourra remarquer que l'étiquette `Vector_001` n'est pas nécessaire. Elle est présente uniquement à titre indicatif.

Nous pouvons maintenant passer au programme principal. La directive `ORG $500`, nous indique qu'il va se situer à l'adresse `$500`.

Le programme principal est constitué de plusieurs instructions `MOVE` qui serviront à initialiser certains registres de donnée à l'aide de données immédiates. On peut également remarquer la présence de l'étiquette `Main`, placée devant la première instruction à exécuter.

- Vous pouvez maintenant assembler le programme. **Appuyez sur la touche `[F9]` sous Windows ou `[F8]` sous Linux.**

  Une fenêtre apparaît en bas de l'écran et contient le résultat de l'assemblage. Si vous avez fait une erreur, c'est ici qu'elle sera signalée. Dans ce cas, reprenez votre code source au numéro de ligne indiqué puis corrigez votre ou vos erreurs.

  Si votre code source ne contient aucune erreur, la ligne suivante doit s'afficher :
  `End of assembly - no errors were found.`

- Vous pouvez maintenant exécuter le programme. **Pressez la touche `[F10]` sous Windows ou `[F5]` sous Linux.**

  Une nouvelle fenêtre s'affiche au milieu de l'écran : il s'agit du **débogueur 68000**. Celui-ci vous permettra d'explorer vos programmes dans leurs moindres détails.

  Sa fenêtre principale contient votre code assembleur. L'affichage y est toutefois légèrement différent. En effet, les directives d'assemblages et les étiquettes ont totalement disparu. Elles ont été remplacées par les adresses et le code machine.

  Les instructions `ORI` qui apparaissent n'appartiennent pas à votre programme. Le contenu de la mémoire est initialisé à zéro et l'instruction `ORI.B #0,D0` correspond au code machine `0000 0000`.

La partie inférieure de la fenêtre affiche le contenu de tous les registres du 68000 et une petite partie de la mémoire.

La partie supérieure laisse apparaître une zone grisée indiquant qu'aucune instruction n'est exécutée. Dans un premier temps, nous ne tiendrons pas compte de cette zone ni de ce qu'elle contient.

- Il faut maintenant exécuter votre première instruction. **Pressez la touche `[F11]`.**

  La valeur du registre **D0** vient d'être modifiée par l'instruction : elle s'affiche en rouge. La valeur d'un registre s'affiche au format hexadécimal sur 32 bits. C'est donc la valeur `$00000041` qui s'affiche. Elle correspond à la valeur qui vient d'être chargée dans le registre.

  À droite de l'affichage hexadécimal, on peut voir la correspondance ASCII de la valeur contenue dans le registre **D0**. La valeur `$41` est donc le code ASCII du caractère `'A'`.

- **Appuyez de nouveau sur `[F11]`** afin d'exécuter l'instruction suivante.

  Cette fois, c'est **D1** qui s'affiche en rouge. La valeur chargée dans le registre est 65 535. Ce qui nous donne au format hexadécimal sur 32 bits : `$0000FFFF`.

  Il est possible d'obtenir la valeur du registre dans tous les formats usuels (décimal, hexadécimal, binaire, signé, non signé, etc.).

- **Cliquez pour cela sur le bouton `[D1]`.**

  Une nouvelle fenêtre apparaît et affiche la valeur actuelle de **D1** dans tous les formats usuels.

  Cette fenêtre peut également servir de calculatrice de conversion. Saisissez par exemple la valeur `-1` puis cliquez sur le bouton **`[=]`**. Essayez également avec les expressions `$40+D0` et `(5+%101)*4`.

- Il est possible de modifier manuellement la valeur du registre. Par exemple, **saisissez la valeur 255 et appuyez sur le bouton `[OK]`**.

  La valeur de **D1** a été modifiée et vaut maintenant `$000000FF`.

- **Appuyez de nouveau sur `[F11]`** afin d'exécuter l'instruction suivante.

  Cette fois, c'est le registre **D2** qui est modifié. Attention, la valeur -5 apparaît au format hexadécimal sur 32 bits. Pour voir la valeur -5 s'afficher au format décimal, il suffit de cliquer sur le bouton **`[D2]`**.

- **Fermez le débogueur.**

Étape 2

Continuons par un programme qui additionne le contenu de deux registres sur 8 bits.

- **Ouvrez un nouveau fichier** dans l'éditeur de texte.
- **Sauvegardez-le** sous le nom `"Etape_02.asm"`.
- **Tapez le texte ci-dessous :**

```
            org     $4

Vector_001  dc.l    Main

            org     $500

Main        move.b  #18,d0  ; 18 -> D0.B
            move.b  #12,d1  ; 12 -> D1.B
            add.b   d0,d1   ; D0.B + D1.B -> D1.B (12 + 18 = 30).
```

- **Assemblez, corrigez les erreurs si besoin est, puis lancez le débogueur.**

- **Exécutez pas à pas le programme à l'aide de la touche `[F11]`.**

- Une fois la dernière instruction exécutée, assurez-vous que la valeur contenue dans **D1** est bien 30_{10}. Vous pouvez pour cela cliquer sur le bouton **`[D1]`**.

- Intéressons-nous maintenant à la zone grisée située dans la partie haute. Cette zone indique les dernières instructions à avoir été exécutées par le 68000.

  Il est possible d'annuler l'exécution des instructions précédemment exécutées.

- **Appuyez sur la touche d'effacement arrière `[Back Space]`.**

  Le 68000 et la mémoire reviennent dans l'état où ils étaient avant l'exécution de l'instruction `ADD`. On a **D1** qui retrouve la valeur 12.

- **Appuyez deux fois sur la touche d'effacement arrière `[Back Space]`.**

  Nous sommes revenus au point de départ.

- Pour l'instant, nous avons toujours exécuté les instructions une par une. Il s'agit du mode d'exécution pas à pas.

  Toutefois, l'émulateur est capable d'exécuter plusieurs instructions successivement, mais cela nécessite de positionner un point d'arrêt (*breakpoint*).

- Il est possible de positionner un **point d'arrêt sur une adresse**.

  Pour cela, **cliquez dans la bordure gauche au niveau de l'instruction** (située à l'adresse `$00050A`) **qui suit votre programme.** Le point d'arrêt s'affiche sur un fond rouge.

  Vous pouvez maintenant lancer votre programme jusqu'à ce point d'arrêt.

- **Pressez la touche `[F9]`.**

  Attention ! si vous oubliez de positionner le point d'arrêt, il faudra appuyer sur la touche **`[Echap]`** pour stopper l'émulateur.

  On constate que les trois instructions ont été exécutées jusqu'au point d'arrêt.

- Il est possible d'effectuer un *reset* du 68000 en appuyant sur les touches **`[Ctrl+R]`**.

  On constate que les registres ne sont pas réinitialisés à zéro. En effet, le 68000 n'initialise jamais les registres de donnée ou d'adresse lors d'un *reset*. Il ne faut donc jamais supposer qu'un registre sera initialisé à zéro au moment de l'allumage. C'est l'émulateur, et non pas le 68000, qui initialise les registres à zéro lors du lancement du débogueur.

- **Fermez le débogueur.**

  Il est également possible de positionner un **point d'arrêt sur une instruction**.

- **Ajouter l'instruction `ILLEGAL` après l'instruction `ADD` :**

```
            org     $4

Vector_001  dc.l    Main

            org     $500

Main        move.b  #18,d0  ; 18 -> D0.B
            move.b  #12,d1  ; 12 -> D1.B
            add.b   d0,d1   ; D0.B + D1.B -> D1.B (12 + 18 = 30).

            illegal         ; Point d'arrêt sur une instruction.
```

- **Assemblez puis lancez le débogueur.**

  Vous pouvez maintenant lancer votre programme jusqu'à ce point d'arrêt.

- **Pressez la touche `[F9]`.**

L'émulateur s'arrête alors à l'instruction `ILLEGAL`. Ainsi pour terminer un programme, n'hésitez pas à y ajouter cette dernière instruction qui fait office de point d'arrêt. Vous pouvez également constater que ce type de point d'arrêt s'affiche sur un fond bleu.

Attention ! il faut bien comprendre que dans un fonctionnement normal du 68000, **l'instruction `ILLEGAL` n'est pas un point d'arrêt**. C'est le débogueur qui détourne cette instruction de son fonctionnement normal afin de s'en servir comme point d'arrêt.

- **Fermez le débogueur.**

Étape 3

Nous souhaitons maintenant additionner le contenu de deux cases mémoire de 16 bits.

- **Ouvrez un nouveau fichier** dans l'éditeur de texte.
- **Sauvegardez-le** sous le nom `"Etape_03.asm"`.
- **Tapez le texte ci-dessous :**

```
            org     $4

Vector_001  dc.l    Main

            org     $500

Main        move.w  NUMBER1,d0  ; (NUMBER1) -> D0.W
            add.w   NUMBER2,d0  ; (NUMBER2) + D0.W -> D0.W
            move.w  d0,SUM      ; D0.W -> (SUM)

            illegal

            org     $550

NUMBER1     dc.w    $2222       ; Le nombre $2222 est stocké à l'addresse NUMBER1.
NUMBER2     dc.w    $5555       ; Le nombre $5555 est stocké à l'addresse NUMBER2.
SUM         ds.w    1           ; On réserve 16 bits pour stocker la somme.
```

Cette fois, les nombres à additionner ne sont plus dans les registres, mais dans la mémoire. Nous allons commencer par déterminer où ces nombres se trouvent en mémoire.

Nous remarquons assez rapidement que le premier nombre est stocké à l'adresse `$550` et le second à l'adresse `$552`. La somme sera, quant à elle, stockée à l'adresse `$554`. Les adresses assignées aux différentes étiquettes seront donc les suivantes :

➔ `NUMBER1 = $550`
➔ `NUMBER2 = $552`
➔ `SUM     = $554`

- **Assemblez puis lancez le débogueur.**

- Il est possible d'afficher le contenu de la mémoire dans les formats hexadécimal et ASCII. **Cliquez sur l'onglet `[Mémoire]`.**

  Le contenu de la mémoire est représenté à partir de l'adresse `$500`. Les premiers octets ne sont donc rien d'autre que le code machine du programme.

  Repérez où se trouve l'adresse `$550` et vérifiez que les deux nombres `$2222` et `$5555` sont bien présents en mémoire. Repérez également l'emplacement où sera stockée la somme une fois le programme exécuté.

  Vous pouvez remarquer que lorsque le pointeur de la souris survole un octet, une infobulle apparaît. Celle-ci indique l'adresse de l'octet ainsi que ses représentations hexadécimale, décimale et ASCII.

- **Cliquez sur l'onglet `[Désa]`** afin de revenir à la vue de désassemblage.

  Avant d'exécuter le programme, vous pouvez noter qu'une petite flèche jaune, à gauche de la fenêtre, indique la prochaine instruction à exécuter (uniquement sur la version Linux). Cette instruction est également affichée à droite du bouton **`[PC]`**.

- **Appuyez sur la touche `[F11]`**.

  Le nombre `$2222` est chargé dans le registre **D0**.

- **Appuyez sur la touche `[F11]`**.

  Le nombre `$5555` est additionné au registre **D0**. Le registre **D0** contient maintenant la somme : `$2222` + `$5555` = `$7777`

- **Appuyez sur la touche `[F11]`**.

  La somme est copiée en mémoire à l'adresse `$554`.

- **Cliquez sur l'onglet `[Mémoire]`** et vérifiez que la somme est bien présente dans la mémoire à la bonne adresse.

- **Fermez le débogueur.**

Étape 4

Dans cette étape, nous allons calculer la somme de cinq nombres présents dans un tableau. Chaque nombre est codé sur 8 bits. On suppose que la somme ne dépassera jamais 255.

- **Ouvrez un nouveau fichier** dans l'éditeur de texte.
- **Sauvegardez-le** sous le nom `"Etape_04.asm"`.
- **Tapez le texte ci-dessous :**

```
            org     $4

Vector_001  dc.l    Main

            org     $500

Main        movea.l #TAB,a0     ; On initialise A0 avec l'adresse du tableau.

            move.b  (a0)+,d0    ; Nombre 1        -> D0.B ; A0 + 1 -> A0
            add.b   (a0)+,d0    ; Nombre 2 + D0.B -> D0.B ; A0 + 1 -> A0
            add.b   (a0)+,d0    ; Nombre 3 + D0.B -> D0.B ; A0 + 1 -> A0
            add.b   (a0)+,d0    ; Nombre 4 + D0.B -> D0.B ; A0 + 1 -> A0
            add.b   (a0),d0     ; Nombre 5 + D0.B -> D0.B

            move.b  d0,SUM      ; D0.B -> (SUM)

            illegal

            org     $550

TAB         dc.b    18,3,5,9,14 ; Tableau contenant les 5 nombres.
SUM         ds.b    1           ; On réserve 8 bits pour stocker la somme.
```

- **Assemblez puis lancez le débogueur.**

- **Appuyez sur la touche `[F11]`.**

  Le registre **A0** est initialisé avec l'adresse de départ du tableau. Sa nouvelle valeur s'affiche en rouge et il s'agit de l'adresse `$550`.

  À droite de la valeur du registre, le débogueur affiche le contenu de la mémoire à partir de l'adresse du registre (aux formats hexadécimal et ASCII). On peut donc y trouver les cinq nombres à additionner.

- **Appuyez sur la touche `[F11]`.**

  Le premier nombre à additionner est chargé dans le registre **D0**.

  La valeur du registre **A0** a été incrémentée. Le contenu de la mémoire (à droite de la valeur hexadécimale du registre) s'affiche donc à partir de l'adresse `$551`. Le premier nombre à s'afficher est donc le deuxième nombre du tableau.

- Terminez l'exécution du programme (en appuyant plusieurs fois sur **[F11]** ou une seule fois sur **[F9]**.

- Vérifiez que la somme stockée en mémoire à l'adresse `SUM` (`$555`) est correcte (onglet **[Mémoire]**).

- **Fermez le débogueur.**

- Il est également possible d'accéder au membre d'un tableau à l'aide du mode d'adressage indirect par registre d'adresse avec déplacement : d16(An).

```
            org     $4

Vector_001  dc.l    Main

            org     $500

Main        movea.l #TAB,a0     ; On initialise A0 avec l'adresse du tableau.

            move.b  (a0),d0     ; Nombre 1        -> D0.B
            add.b   1(a0),d0    ; Nombre 2 + D0.B -> D0.B
            add.b   2(a0),d0    ; Nombre 3 + D0.B -> D0.B
            add.b   3(a0),d0    ; Nombre 4 + D0.B -> D0.B
            add.b   4(a0),d0    ; Nombre 5 + D0.B -> D0.B

            move.b  d0,SUM      ; D0.B -> (SUM)

            illegal

            org     $550

TAB         dc.b    18,3,5,9,14 ; Tableau contenant les 5 nombres.
SUM         ds.b    1           ; On réserve 8 bits pour stocker la somme.
```

Étape 5

1. Déterminez manuellement (sans l'aide de l'assembleur ni du débogueur) le résultat des additions suivantes, ainsi que le contenu des bits **N**, **Z**, **V** et **C** du registre d'état.
   - Addition sur 8 bits : `$B4 + $4C`
   - Addition sur 16 bits : `$B4 + $4C`
   - Addition sur 16 bits : `$4AC9 + $D841`
   - Addition sur 32 bits : `$FFFFFFFF + $00000015`

2. Servez-vous du débogueur afin de vérifier que vos réponses à la question précédente sont correctes. Pour cela, réalisez un programme en assembleur 68000 qui effectue les quatre additions ci-dessus. Assemblez votre programme et exécutez-le à l'aide du débogueur après avoir localisé où ce dernier affiche les bits du registre d'état.

Étape 6

On souhaite réaliser l'addition de deux nombres entiers codés sur 128 bits. Proposez un programme en assembleur 68000 qui effectue cette addition. Vous respecterez les indications suivantes :

Entrées : **D3:D2:D1:D0** = Entier sur 128 bits (**D0** étant les 32 bits de poids faible).
D7:D6:D5:D4 = Entier sur 128 bits (**D4** étant les 32 bits de poids faible).
Sortie : **D3:D2:D1:D0** = **D3:D2:D1:D0** + **D7:D6:D5:D4**

Utilisez uniquement les instructions `ADD` et `ADDX`.

Étape 7

En utilisant uniquement des instructions de rotation, donnez quelques instructions qui modifient la valeur de **D1** afin de lui donner les valeurs ci-dessous. Pour chaque cas, la valeur initiale de **D1** est `$76543210`.

- **D1** = `$76543120`
- **D1** = `$75640213`
- **D1** = `$54231067`
- **D1** = `$05634127`

Utilisez uniquement les instructions `ROL`, `ROR` et `SWAP`.