T.P. 3
La pile et les sous-programmes

Prérequis : avoir lu les pages 27 à 32 du cours.

Étape 1

Dans cette étape, nous allons reprendre le programme du TP précédent, qui renvoie la valeur absolue d'un entier signé, et le transformer en sous-programme.

- **Tapez le programme** ci-dessous et **sauvegardez-le** sous le nom `"Abs.asm"`.

```
            ; ==============================
            ; Initialisation des vecteurs
            ; ==============================

            org     $4

vector_001  dc.l    Main

            ; ==============================
            ; Programme principal
            ; ==============================

            org     $500

Main        move.l  #-4,d0
            jsr     Abs

            move.l  #-32500,d0
            jsr     Abs

            illegal

            ; ==============================
            ; Sous-programmes
            ; ==============================

Abs         tst.l   d0
            bpl     \quit

            neg.l   d0

\quit       rts
```

Ce code source est divisé en trois parties :

- Initialisation des vecteurs ;
- Programme principal ;
- Sous-programmes.

Notez bien cette structure. Lorsque vous réaliserez vos propres sous-programmes, ils devront être testés de cette manière ; c'est-à-dire à partir de l'appel d'un programme principal. Il vous faudra ainsi créer dans tous les cas un programme principal, contenant un ou plusieurs appels à vos sous-programmes avec différentes valeurs de tests.

Votre programme principal devra se terminer par un point d'arrêt, vos sous-programmes par un `RTS`.

Dans cet exemple, le programme principal contient deux appels au sous-programme **`Abs`** avec deux valeurs de tests et un point d'arrêt sur l'instruction `ILLEGAL`.

- On peut remarquer qu'un caractère *backslash* (\) a été placé devant l'étiquette `quit`. Qu'est-ce que cela signifie ?

  Tout d'abord, il faut savoir qu'une étiquette qui n'est pas précédée d'un caractère *backslash* est une étiquette absolue. Par exemple, les étiquettes `Main` et `Abs` sont des étiquettes absolues. De plus, l'assembleur n'autorise qu'une seule étiquette absolue de même nom dans un fichier source. Si l'on ajoute une seconde étiquette `Abs` autre part dans le fichier, l'assembleur renverra une erreur.

  Une étiquette précédée d'un caractère *backslash* est une étiquette relative. Elle est relative à la première étiquette absolue qui la précède dans le code source. Par exemple, l'étiquette `\quit` est relative à l'étiquette absolue `Abs`. L'assembleur considéra en interne que le nom complet de l'étiquette `\quit` est en fait : `Abs\quit`.

  L'intérêt des étiquettes relatives est de permettre au programmeur de placer plusieurs étiquettes de même nom dans un fichier source. Si un fichier contient plusieurs sous-programmes, chaque sous-programme aura donc la possibilité de posséder sa propre étiquette `\quit`.

  Plusieurs étiquettes relatives (`\loop`, `\quit`, etc.) peuvent être associées à une même étiquette absolue.

  Comme notre fichier source ne possède qu'une seule étiquette `\quit`, il n'était pas nécessaire de transformer cette dernière en étiquette relative. Toutefois, nous prendrons dès maintenant les deux habitudes suivantes :

  - **L'étiquette du point d'entrée d'un sous-programme sera toujours une étiquette absolue.**
  - **Toutes les étiquettes contenues dans un sous-programme seront des étiquettes relatives.**

- **Assemblez le programme**, puis **lancez le débogueur**.

- **Appuyez sur la touche `[F11]`**. La valeur de **D0** est initialisée.

- L'instruction suivante est un appel au sous-programme **`Abs`** (`JSR $51A`). L'adresse de retour du sous-programme devra donc être l'adresse de l'instruction qui suit ce `JSR`. C'est-à-dire l'adresse `$50C`.

- **Appuyez sur la touche `[F11]`**. Nous entrons dans le sous-programme.

  L'adresse de retour a été sauvegardée dans la pile.

  La pile est représentée par la liste sur la partie gauche de la fenêtre. Le sommet de la pile est situé juste en dessous de la partie grisée de la liste. La partie grisée affiche les données qui n'appartiennent plus à la pile. Le contenu de la pile peut être affiché sur 8, 16 ou 32 bits.

- **Exécutez chaque instruction jusqu'au `RTS` (exclu).**

  Le registre **D0** a bien pris la valeur `4` qui est la valeur absolue de `-4`.

- **Exécutez le `RTS`.**

  Le **PC** pointe maintenant sur l'instruction située juste après le `JSR`. L'adresse de retour a été dépilée ; elle se trouve dans la partie grisée.

- Pour exécuter les deux instructions restantes, nous allons utiliser la touche **`[F10]`** au lieu de la touche **`[F11]`**.

- **Appuyez sur la touche `[F10]`.**

  Le registre **D0** est initialisé (même résultat qu'avec l'appui sur **`[F11]`**).

- **Appuyez une nouvelle fois sur `[F10]`.**

  Le sous-programme a été exécuté entièrement. L'adresse de retour n'est plus dans la pile et le résultat est contenu dans le registre **D0**.

  Un appui sur la touche **`[F10]`** permet donc d'exécuter toutes les instructions d'un sous-programme.

`[F11]` permet une exécution pas à pas en pénétrant dans un sous-programme.
`[F10]` permet une exécution pas à pas sans pénétrer dans un sous-programme.

- **Fermez le débogueur.**

Étape 2

Dans cette étape, nous allons reprendre le programme du TP précédent, qui renvoie la taille d'une chaîne de caractères, et le transformer en sous-programme.

- **Tapez le programme** ci-dessous et **sauvegardez-le** sous le nom "`StrLen.asm`".

```
            ; ==============================
            ; Initialisation des vecteurs
            ; ==============================

            org     $4

vector_001  dc.l    Main

            ; ==============================
            ; Programme principal
            ; ==============================

            org     $500

Main        movea.l #String1,a0
            jsr     StrLen

            movea.l #String2,a0
            jsr     StrLen

            illegal

            ; ==============================
            ; Sous-programmes
            ; ==============================

StrLen      move.l   a0,-(a7)       ; Sauvegarde le registre A0 dans la pile.

            clr.l   d0

\loop       tst.b   (a0)+
            beq     \quit

            addq.l  #1,d0
            bra     \loop

\quit       movea.l  (a7)+,a0       ; Restaure la valeur du registre A0.
            rts

            ; ==============================
            ; Données
            ; ==============================

String1     dc.b    "Cette chaine comporte 36 caracteres.",0
String2     dc.b    "Celle-ci en comporte 24.",0
```

- Afin d'améliorer la clarté du code, une quatrième partie qui contiendra les données du programme est ajoutée.

- À partir de cette étape, nous allons imposer la règle suivante :

À l'exception des registres de sortie, aucun registre de donnée ou d'adresse ne devra être modifié en sortie d'un sous-programme.

Attention ! cela ne signifie pas qu'un sous-programme n'a pas le droit de modifier n'importe quels registres de donnée ou d'adresse. Cela signifie qu'au moment de l'exécution de l'instruction `RTS`, les registres qui ne sont pas des registres de sorties devront posséder la même valeur qu'au moment de l'appel au sous-programme.

Par exemple, le registre de sortie du sous-programme **`StrLen`** est le registre **D0**. Ce dernier devra contenir le nombre de caractères de la chaîne à mesurer. Il est donc tout à fait normal que le sous-programme modifie la valeur du registre **D0**.

Par contre, le sous-programme **`StrLen`** utilise et modifie le registre **A0**. Or, le registre **A0** n'est pas un registre de sortie. Il faut donc s'arranger pour que celui-ci récupère sa valeur initiale avant de quitter le sous-programme. Pour cela, on sauvegarde la valeur du registre **A0** dans la pile au début du sous-programme, puis on dépile cette valeur dans le registre **A0** avant de quitter le sous-programme. C'est pour cette raison que les instructions `MOVE` et `MOVEA` ont été ajoutées.

- **Assemblez le programme**, puis **lancez le débogueur**.

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

  Le registre **A0** a été initialisé avec l'adresse de la première chaîne (**A0** = `$0000052E`)

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

  Nous avons sauté au sous-programme **`StrLen`**. L'adresse de retour se trouve au sommet de la pile.

- Nous allons maintenant empiler la valeur du registre **A0**. **Appuyez sur la touche `[F11]`**.

  On constate que la valeur du registre **A0** (`$0000052E`) se trouve au sommet de la pile.

- **Cliquez sur la bordure gauche au niveau l'adresse `$52A`**. Vous venez de placer un point d'arrêt sur l'instruction `MOVEA`.

- **Appuyez sur la touche `[F9]`** afin d'exécuter le reste du sous-programme.

  Le registre **D0** contient le nombre de caractères de la chaîne (**D0** = 36 = $24) et la valeur du registre **A0** a été modifiée.

- **Cliquez sur la bordure gauche au niveau l'adresse `$52A`** afin de supprimer le point d'arrêt.

- **Appuyez sur la touche `[F11]`**. La valeur initiale du registre **A0** a été dépilée et replacée dans le registre **A0**.

- **Appuyez sur la touche `[F11]`** pour sortir du sous-programme.

- **Appuyez sur la touche `[F10]`**. Le registre **A0** est initialisé avec l'adresse de la seconde chaîne (**A0** = `$00000553`).

- **Appuyez sur la touche `[F10]`**. L'intégralité du sous-programme a été exécutée. Le registre **D0** contient le nombre de caractères de la chaîne (**D0** = 24 = $18) et la valeur du registre **A0** n'a pas changée.

- **Fermez le débogueur.**

Étape 3

Transformons maintenant le programme **SpaceCount** du TP précédent en sous-programme. Cette fois, la solution proposée modifie trois registres : **D0**, **D1** et **A0**. Le registre de sortie est **D0**, c'est donc tout à fait normal qu'il soit modifié. Par contre, les registres **D1** et **A0** doivent être restaurés. Nous allons pour cela les empiler. Comme plusieurs registres doivent être empilés, nous pouvons utiliser l'instruction `MOVEM`.

- **Tapez le programme** ci-dessous et **sauvegardez-le** sous le nom "`SpaceCount.asm`".
- **Exécutez-le pas à pas** sans pénétrer dans le sous-programme (touche **[F10]**).
- Vérifiez que les valeurs renvoyées dans **D0** sont correctes.

```
            ; ==============================
            ; Initialisation des vecteurs
            ; ==============================

            org     $4

vector_001  dc.l    Main

            ; ==============================
            ; Programme principal
            ; ==============================

            org     $500

Main        movea.l #String1,a0
            jsr     SpaceCount

            movea.l #String2,a0
            jsr     SpaceCount

            illegal

            ; ==============================
            ; Sous-programmes
            ; ==============================

SpaceCount  movem.l d1/a0,-(a7)     ; Sauvegarde les registres dans la pile.

            clr.l   d0

\loop       move.b  (a0)+,d1
            beq     \quit

            cmp.b   #' ',d1
            bne     \loop

            addq.l  #1,d0
            bra     \loop

\quit       movem.l   (a7)+,d1/a0   ; Restaure les registres.
            rts

            ; ==============================
            ; Données
            ; ==============================

String1     dc.b    "Cette chaine comporte 4 espaces.",0
String2     dc.b    "Celle-ci en comporte 3.",0
```

Étape 4

Transformez le programme **LowerCount** du TP précédent en sous-programme. Si vous utilisez d'autres registres que **D0**, pensez à les sauvegarder dans la pile afin de pouvoir restaurer leurs valeurs initiales.

Vous utiliserez la structure suivante pour tester votre sous-programme :

```
            ; ==============================
            ; Initialisation des vecteurs
            ; ==============================

            org     $4

vector_001  dc.l    Main

            ; ==============================
            ; Programme principal
            ; ==============================

            org     $500

Main        movea.l #String1,a0
            jsr     LowerCount

            movea.l #String2,a0
            jsr     LowerCount

            illegal

            ; ==============================
            ; Sous-programmes
            ; ==============================

LowerCount  ; ...
            ; ...
            ; ...

            ; ==============================
            ; Données
            ; ==============================

String1     dc.b    "Cette chaine comporte 28 minuscules.",0
String2     dc.b    "Celle-ci en comporte 16.",0
```

Étape 5

Réalisez le sous-programme **`AlphaCount`** qui renvoie le nombre de caractères alphanumériques dans une chaîne de caractères.

Entrée : **A0.L** pointe sur le premier caractère d'une chaîne de caractères.

Sortie : **D0.L** renvoie le nombre de caractères alphanumériques de la chaîne.

Indications :

- Un caractère alphanumérique est une lettre (minuscule ou majuscule) ou un chiffre (de 0 à 9).
- Dans un premier temps, réalisez les deux sous-programmes ci-dessous. Pour chacun d'eux, conservez la même structure que **`LowerCount`**.
  - **`UpperCount`** qui renvoie le nombre de majuscules dans une chaîne de caractères ;
  - **`DigitCount`** qui renvoie le nombre de chiffres dans une chaîne de caractères.
- Votre sous-programme **`AlphaCount`** devra appeler les sous-programmes **`LowerCount`**, **`UpperCount`** et **`DigitCount`** et renvoyez la somme des valeurs obtenues.

Vous utiliserez la structure suivante pour tester votre sous-programme :

```
            ; ==============================
            ; Initialisation des vecteurs
            ; ==============================

            org     $4

vector_001  dc.l    Main

            ; ==============================
            ; Programme principal
            ; ==============================

            org     $500

Main        movea.l #String1,a0
            jsr     AlphaCount

            movea.l #String2,a0
            jsr     AlphaCount

            illegal

            ; ==============================
            ; Sous-programmes
            ; ==============================

LowerCount  ; ...

UpperCount  ; ...

DigitCount  ; ...

AlphaCount  ; ...

            ; ==============================
            ; Données
            ; ==============================

String1     dc.b    "Cette chaine comporte 46 caracteres alphanumeriques.",0
String2     dc.b    "Celle-ci en comporte 19.",0
```

Étape 6

Réalisez le sous-programme **`Atoui`** (*ASCII to Unisgned Int*) qui renvoie la valeur numérique d'une chaîne de caractères. On suppose que cette chaîne est non nulle, qu'elle ne contient que des chiffres, et que le nombre qu'elle représente est un entier non signé sur 16 bits (de 0 à 65 535). Par exemple, la chaîne "52146" devra renvoyer la valeur numérique 52146.

Entrée : **A0.L** pointe sur la chaîne à convertir.

Sortie : **D0.L** renvoie la valeur numérique de la chaîne.

Indications :

- On obtient la valeur numérique d'un caractère en lui retranchant le code ASCII du caractère '0'.

  Exemple :

```
        move.b  #'6',d1     ; D1.B = $36 (code ASCII du caractère 6).
        subi.b  #'0',d1     ; D1.B = 6   ($36 - $30 = 6).
```

- Il faut réaliser une boucle qui récupère les caractères de la chaîne, qui les convertit en valeur numérique, et qui les ajoute à une variable contenant le résultat.

  Exemple :
  Pour la chaîne "52146", on aura une variable qui prendra successivement les valeurs suivantes : 5, 52, 521, 5214 et enfin 52146. Une multiplication par 10 devra donc apparaître quelque part dans la boucle.

- Utilisez le registre **D1** pour lire un caractère dans la chaîne et pour réaliser sa conversion numérique.

- Utilisez le registre **D0** pour accumuler le résultat final.

- Vous testerez votre sous-programme en respectant la structure utilisée précédemment (initialisation des vecteurs, programme principal, sous-programme et donnée). Utilisez au moins trois chaînes de tests (par exemple, "52146", "309" et "2570").