T.P. 1
Initiation

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 déjà vus en TD, et enfin, vous serez à même de concevoir, d'assembler et de déboguer vos propres réalisations jusqu'à leur bon fonctionnement.

Étape 1

Installez les logiciels nécessaires aux TP. Lisez attentivement la documentation concernant le débogueur.

Étape 2

Commençons par quelque chose de très simple afin d'acquérir les notions de base. Reprenons le programme déjà étudié en TD qui échange sur huit bits le contenu de deux registres sans passer par une troisième variable.

- **Tapez celui-ci dans l'éditeur de texte** sous la forme suivante :

```
            org     $0

vector_000  dc.l    $0
vector_001  dc.l    Main

            org     $400

            ; Placement d'une valeur initiale dans les registres D0 et D1.
Main        move.l  #$12345678,d0
            move.l  #$00abcdef,d1

            ; Échange des registres D0.B et D1.B.
            eor.b   d0,d1
            eor.b   d1,d0
            eor.b   d0,d1
```

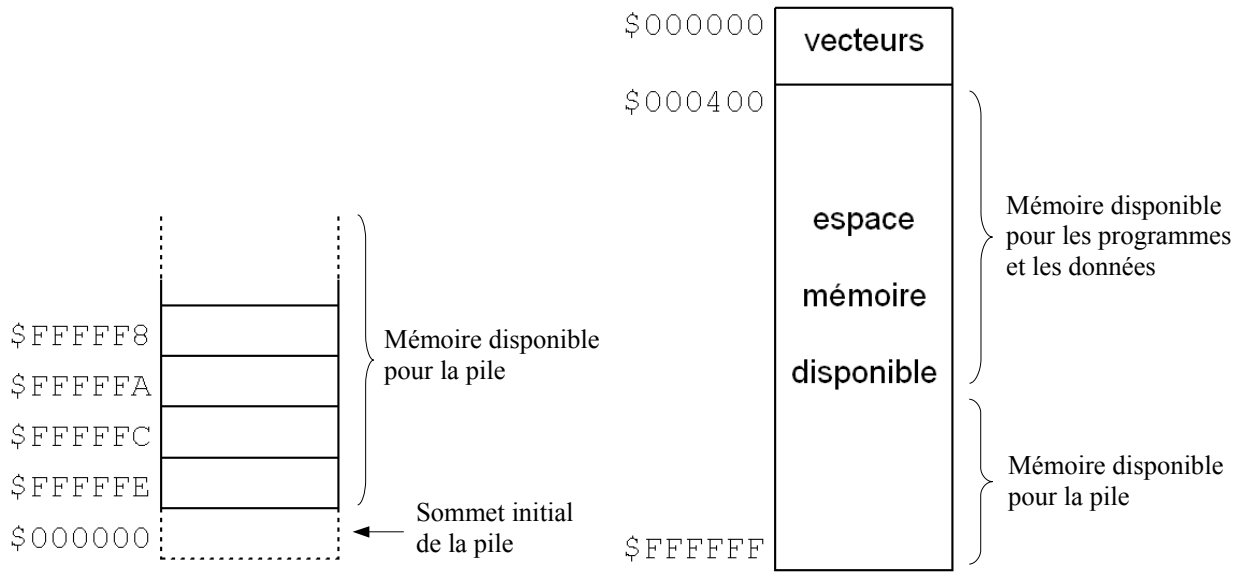
- **Sauvegardez-le** sous le nom "`TP1_Etape02.asm`".

- 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 `$000000`.

Pour le 68000, les 1024 premières adresses sont réservées aux vecteurs d'exception. Il y a 256 vecteurs contenant chacun une adresse de 32 bits. Les lignes suivantes vont donc servir à initialiser les vecteurs d'exception. La dernière page du manuel contient la liste complète de tous les vecteurs d'exception du 68000. Pour l'instant, nous allons uniquement nous intéresser aux deux premiers vecteurs.

- **Le vecteur 0** doit contenir l'adresse initiale de la pile qui sera chargée dans le registre **SSP** après un *reset* du 68000. Si nous choisissons l'adresse `$000000` comme étant le sommet initial de la pile, nous avons donc toutes les adresses hautes disponibles pour la pile comme le montre les deux figures ci-dessous :



- **Le vecteur 1** doit contenir l'adresse du point d'entrée du programme. C'est l'adresse qui sera chargée dans le **PC** après un *reset* du 68000. Il suffit donc d'y placer l'étiquette du point d'entrée du programme. À cet effet, nous utiliserons l'étiquette `Main` dans l'ensemble du TP.

Pour initialiser un vecteur d'exception, il faut écrire un mot long (32 bits) à l'emplacement mémoire réservé à ce vecteur. Nous utiliserons pour cela la directive `DC.L`, qui permet d'écrire un mot long dans la mémoire.

Ces trois premières lignes servent donc à initialiser correctement les vecteurs associés à la pile (**SSP**) et au compteur programme (**PC**). Elles devront être présentes dans tous vos codes sources.

Nous pouvons maintenant passer au programme principal. La directive `ORG $400`, nous indique qu'il va se situer à l'adresse `$400`. Puis arrivent deux instructions chargeant une valeur initiale dans les deux registres que l'on désire intervertir. On peut également remarquer l'emplacement de l'étiquette `Main`, placée devant la première instruction à exécuter. Pour finir, on retrouve les trois instructions OU EXCLUSIF réalisant l'échange du contenu des deux registres.

Vous pouvez maintenant assembler le programme.

- **Appuyez sur la touche `[F9]`** ou bien cliquez sur l'icône 1, en forme de visage, dans la barre d'outils de l'éditeur.

  Une fenêtre apparaît alors 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]`** ou bien cliquez sur l'icône 2, en forme de visage, dans la barre d'outils de l'éditeur.

  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 le code machine et les adresses. La fenêtre du bas affiche le contenu de tous les registres du 68000 et une petite partie de la mémoire.

  Dans la partie haute apparaît 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.
  Exécutons l'instruction suivante.

- **Appuyez de nouveau sur `[F11]`**, cette fois c'est **D1** qui s'affiche en rouge.

  Maintenant que vous savez exécuter une instruction en mode pas à pas, essayez de faire attention aux champs **`SRC`** et **`DST`** du bas de la fenêtre. Si tout va bien, ils contiennent les valeurs **`$78`** et **`$EF`**. Ces valeurs dépendent de l'instruction en cours représentée à droite du bouton **`[PC]`** (ou aussi l'instruction précédée d'une petite flèche juste en dessous de la zone grisée). Ce sont les valeurs de l'opérande source et de l'opérande destination. Leur taille correspond à celle de l'instruction. Ici, il s'agit donc de la valeur des registres **D0.B** et **D1.B**.

- **Exécutez maintenant les trois instructions `EOR`** en observant à chaque étape les registres modifiés.

  Une fois votre code exécuté, les registres **D0.B** et **D1.B** ont bien échangé leur contenu.

- **Fermez la fenêtre du débogueur**.

Étape 3

Continuons avec le code vu en TD permettant de déterminer la valeur absolue d'un nombre contenu dans le registre **D0**.

- **Tapez-le** puis **sauvegardez-le** sous le nom `"TP1_Etape03.asm"`.

```
            org     $0

vector_000  dc.l    $0
vector_001  dc.l    Main

            org     $a000

Main        tst.l   d0
            bpl.s   quit
            neg.l   d0

quit
```

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

  Cette fois, la valeur du registre **D0** n'a pas été initialisée dans le programme, mais il faut quand même lui donner une valeur avant d'exécuter le code.

- **Cliquez sur le bouton `[D0].`**

  Une nouvelle fenêtre apparaît et affiche la valeur actuelle de **D0** dans tous les formats utiles. Cette fenêtre peut également servir de calculatrice de conversion. Tapez par exemple la valeur `-1` puis cliquez sur le bouton **`[=]`**. Essayez également avec les expressions `$40+PC` et `(5+%101)*4`.

  Pour notre test, nous allons attribuer la valeur `-4` au registre **D0**.

- **Saisissez la valeur `-4`** puis **appuyez sur le bouton `[OK].`**

  La valeur du registre **D0** s'affiche en rouge et vaut `$FFFFFFFC` (c'est également la valeur du champ `SRC` puisque la prochaine instruction à exécuter est un `TST.L D0`).

- **Exécutez la première instruction.**

  Le *flag* **N** s'affiche en rouge. Il a changé de valeur et passe à `1` ; ce qui est naturel puisque **D0** est négatif. Exécutez les deux dernières instructions et voyez qu'en sortie du programme le registre **D0** contient bien la valeur `4`.

  On désire maintenant exécuter de nouveau ce programme avec une nouvelle valeur présente dans le registre **D0**.

- **Appuyez sur les touches `[Ctrl+R]`.**

  Le 68000 vient d'effectuer un *reset*.

- **Entrez une nouvelle valeur pour D0**, par exemple `-32500`.

  Nous allons maintenant exécuter plusieurs instructions à la fois.

- **Cliquez dans la bordure gauche au niveau de l'instruction** (située à l'adresse `$A006`) **qui suit votre programme.**

  Elle s'inscrit sur fond rouge et devient un point d'arrêt (*breakpoint*). Il s'agit d'un **point d'arrêt sur une adresse**.
  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.

  D0 contient maintenant la valeur `$00007EF4`. Pour connaître sa représentation décimale, cliquez sur le bouton **`[D0]`**. On peut maintenant apercevoir la valeur `32500` dans le champ **D0.L**. Fermez la fenêtre en cliquant sur le bouton **`[Annuler]`**.

  Pour supprimer un point d'arrêt sur une adresse, il suffit de cliquer dessus une nouvelle fois.

- **Fermez le débogueur.**

- **Ajoutez l'instruction `ILLEGAL` à la fin de votre programme** (au niveau de l'étiquette `quit`).

- **Réassemblez** le tout, **lancez le débogueur** puis **appuyez sur 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. Il s'agit d'un **point d'arrêt sur une instruction**.

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 un point d'arrêt.

- **Fermez le débogueur.**

Étape 4

Voyons maintenant l'appel à un sous-programme. Pour cela, nous allons reprendre le programme précédent et le transformer en sous-programme.

- **Tapez le fichier source** suivant et **sauvegardez-le** sous le nom "`TP1_Etape04.asm`".

```
            org     $0

vector_000  dc.l    $0
vector_001  dc.l    Main

            org     $a000

Main        move.l  #-4,d0
            jsr     Abs
            move.l  #-32500,d0
            jsr     Abs
            illegal

            org     $a030

Abs         tst.l   d0
            bpl.s   quit
            neg.l   d0
quit        rts
```

Si l'on ne tient pas compte de l'initialisation des vecteurs, vous pouvez constater que ce code source est divisé en deux parties :

- Le programme principal **`Main`** (situé à l'adresse `$A000`).
- Le sous-programme **`Abs`** (situé à l'adresse `$A030`).

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 votre sous-programme, avec différentes valeurs de test. **Votre programme principal devra se terminer par un point d'arrêt, votre sous-programme par un `RTS`.**

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

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

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

- **Appuyez de nouveau 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. 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`.

- **Exécutez le `RTS`.**

  Le **PC** revient après le `JSR` et 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**.

`[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 5

Reprenez le sous-programme étudié en TD qui permet le calcul d'une factorielle.

- **Tapez-le** sous la forme suivante puis **sauvegardez-le** sous le nom "`TP1_Etape05.asm`".

```
            org     $0

vector_000  dc.l    $0
vector_001  dc.l    Main

            org     $7000

Main        move.w  #3,d0
            jsr     Facto
            illegal

            org     $8000

Facto       tst.w   d0
            beq.s   quit

            move.w  d0,-(a7)
            subq.w  #1,d0
            bsr.s   Facto
            mulu.w  (a7)+,d0
            rts

quit        moveq.l #1,d0
            rts
```

Constatez, une fois encore, la structure contenant un programme principal avec un point d'arrêt, ainsi que l'appel au sous-programme.

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

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. Au lancement du débogueur, aucune instruction n'a encore été exécutée.

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

L'instruction qui vient d'être exécutée est passée dans la zone grisée.

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

L'instruction `JSR` passe à son tour dans la zone grisée. Si l'on observe l'adresse du `JSR` et l'adresse de la prochaine instruction à exécuter, on remarque un passage de `$7004` à `$8000`. Il est important de comprendre que l'instruction située dans la zone grisée, juste au-dessus de la prochaine instruction à exécuter n'est pas celle située précédemment en mémoire.

- La zone de désassemblage classique affiche la succession des instructions présentes dans la mémoire.
- La zone de désassemblage grisée affiche la succession des instructions dans l'ordre de leur exécution.

Il est possible d'annuler l'exécution de la dernière instruction.

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

  Le `JSR` a été annulé.

- **Appuyez de nouveau sur la touche `[Back Space]`.**

  Nous sommes revenus dans l'état initial du programme.

- **Exécutez maintenant toutes les instructions pas à pas** en observant la pile très attentivement (affichez le contenu de la pile sur 16 bits). Servez-vous des explications vues en TD et de vos observations afin de bien comprendre la récursivité de ce sous-programme.

- Déterminez la valeur limite de fonctionnement de ce sous-programme. À partir de quelle valeur d'entrée, présente dans le registre **D0**, la factorielle est-elle erronée ? Essayez de trouver l'explication de cette valeur limite.

- **Fermez le débogueur.**

Étape 6

Dans cette étape, nous allons étoffer un peu plus la procédure d'initialisation des vecteurs d'exception. La plupart de ces vecteurs permettent de définir une adresse de saut pour la gestion des erreurs. Pour simplifier cette procédure au maximum, nous allons initialiser chacun de ces vecteurs à la même adresse. Cette adresse d'initialisation pointera en direction d'un point d'arrêt sur l'instruction `ILLEGAL`. Ainsi, à chaque déclenchement d'une exception liée à une erreur de programmation, l'émulateur stoppera l'exécution du programme.

Pour écrire la même adresse d'initialisation dans tous les vecteurs, nous utiliserons la directive d'assemblage `DCB.L` (***D**efine **C**onstant **B**lock*).

Sa syntaxe est la suivante : `DCB.L n,data`

Elle permet d'écrire en mémoire `n` fois la donnée `data` sur 32 bits.

- **Tapez le fichier** source suivant et **sauvegardez-le** sous le nom "`TP1_Etape06.asm`".

```
                    org     $0

vector_000          dc.l    $0
vector_001          dc.l    Main
vector_002_255      dcb.l   254,break_exception
break_exception     illegal

                    org     $1000

Main                ; Division par 0
                    clr.w   d0
                    move.w  #50,d1
                    divs.w  d0,d1
```

Dans ce code source, les deux premiers vecteurs sont initialisés comme précédemment. Vient ensuite l'initialisation des 254 autres vecteurs. Ils pointent tous sur la même adresse : `break_exception`. Une instruction `ILLEGAL` faisant office de point d'arrêt est ensuite positionnée à cette adresse. Toute exception déclenchée par le 68000 sera alors suivie d'un point d'arrêt. Ce point d'arrêt sera situé à l'adresse `$400`, car il est déclaré juste après l'initialisation des 256 vecteurs (de l'adresse `$000` à `$3FF`).

`$000000 00 00 00 00` → Vecteur 000 (Initialisation **SSP**)
`$000004 00 00 10 00` → Vecteur 001 (Initialisation **PC**)
`$000008 00 00 04 00` → Vecteur 002
`$00000C 00 00 04 00` → Vecteur 003
`$000010 00 00 04 00` → Vecteur 004
`$000014 00 00 04 00` → Vecteur 005 (Division par zéro)
…
…
`$0003F8 00 00 04 00` → Vecteur 254
`$0003FC 00 00 04 00` → Vecteur 255
`$000400 4A FC` → Instruction `ILLEGAL`

La première adresse disponible comme point d'entrée d'un programme est donc l'adresse `$402`. Pour laisser une petite marge de manœuvre nous ne placerons jamais un point d'entrée en dessous de l'adresse `$1000`.

- **Assemblez ce code source** puis **lancez le débogueur**.

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

La valeur `0` est chargée dans le registre **D0** sur 16 bits (donc aucun changement si le registre était déjà initialisé à `0`).

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

  La valeur `50` (`$32`) est chargée dans le registre **D1** sur 16 bits.

  Arrive ensuite l'instruction de division qui va réaliser la division de **D1.W** par **D0.W** ; c'est-à-dire une division par zéro. Le 68000 n'exécutera pas l'instruction de division, mais déclenchera une exception et sautera à l'adresse `$400` qui contient le point d'arrêt.

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

  Nous constatons que l'émulateur se retrouve bien au niveau du point d'arrêt que nous avons positionné à l'adresse `$400`.

  Si maintenant on désire retrouver la source de l'erreur, il suffit d'annuler l'exécution de la dernière instruction.

- **Appuyez sur la touche [Back Space].**

  On se retrouve au niveau de l'instruction de division.

- **Fermez le débogueur.**

  Chacun de vos codes sources devra donc, pour l'instant, respecter la structure suivante :

```
                    ; ==============================
                    ; Initialisation des vecteurs
                    ; ==============================
                    org     $0
vector_000          dc.l    $0
vector_001          dc.l    Main
vector_002_255      dcb.l   254,break_exception
break_exception     illegal

                    ; ==============================
                    ; Programme principal
                    ; ==============================
                    org     $1000
Main                ……      ……
                    ……      ……
                    ……      ……
                    illegal

                    ; ==============================
                    ; Sous-programmes
                    ; ==============================
Subroutines         ……      ……
                    ……      ……
                    ……      ……
                    rts
```

Étape 7

Réalisez un sous-programme qui renvoie la plus petite valeur **non signée** contenue dans les registres **D1.L** et **D2.L**. La valeur de sortie se placera dans le registre **D0.L**. Vous le sauvegarderez sous le nom `"TP1_Etape07.asm"`.

Indications

- Utilisez l'instruction `CMP` (*cf.* manuel) pour la comparaison entre **D1** et **D2**.
- C'est au niveau du saut conditionnel que sera prise en compte la gestion d'une comparaison signée ou non signée (*cf.* famille des instructions de saut conditionnel).
- Votre programme principal devra contenir un ou plusieurs appels à votre sous-programme en initialisant **D1** et **D2** avec des valeurs de test suffisamment significatives.

- Appelez un enseignant pour valider votre travail -

Étape 8

Dans cette étape, on désire réaliser le sous-programme **`StrLen`** qui renvoie la taille d'une chaîne de caractères.

- **Tapez le fichier** source suivant et **sauvegardez-le** sous le nom `"TP1_Etape08.asm"`.

```
                    ; ==============================
                    ; Initialisation des vecteurs
                    ; ==============================
                    org     $0
vector_000          dc.l    $0
vector_001          dc.l    Main
vector_002_255      dcb.l   254,break_exception
break_exception     illegal

                    ; ==============================
                    ; Programme principal
                    ; ==============================
                    org     $1000
Main                lea     sTest,a0
                    jsr     StrLen
                    illegal

                    ; ==============================
                    ; Sous-programmes
                    ; ==============================
StrLen              rts

                    ; ==============================
                    ; Données
                    ; ==============================
sTest               dc.b    "Ceci est la chaine de test",0
```

Une zone de données séparée du code a été ajoutée. C'est dans cette zone mémoire que nous écrirons la chaîne de caractères grâce à la directive `DC.B`. Un zéro terminal servira à préciser la fin de la chaîne.

- **Assemblez** puis **lancez le débogueur**.
- **Cliquez sur l'onglet `[Hexa]`** pour afficher la vue hexadécimale.
- Trouvez à quelle adresse est placée la chaîne en mémoire.
- Revenez sur la vue de désassemblage.
- Quelles instructions apparaissent à l'emplacement de la chaîne ?
- Est-ce que le désassemblage fonctionne normalement ?
- **Exécutez la première instruction.**

  L'adresse de départ de la chaîne est chargée dans le registre **A0** qui apparaît maintenant en rouge. À droite du contenu du registre, se trouve affiché, en représentation hexadécimale, le contenu de la mémoire à partir de l'adresse pointée par **A0**. Cette représentation hexadécimale est suivie d'une représentation ASCII. On peut donc y apercevoir une partie de la chaîne de caractères.

  N'hésitez pas, lors de l'exécution pas à pas de vos sous-programmes, à vous servir de ces représentations. Elles vous aideront à analyser correctement, et en profondeur, leur comportement.

- **Fermez le débogueur.**

Réalisez maintenant le sous-programme **`StrLen`**. Aucun registre hormis **D0** ne doit être modifié en sortie du sous-programme.
Entrée : **A0.L** pointe sur une chaîne terminée par un caractère nul.
Sortie : **D0.L** renvoie la taille de la chaîne en octets (sans le caractère nul).

Indications
- Initialisez un compteur de caractères.
- Réalisez une boucle qui teste chaque caractère de la chaîne.
- Tant que le caractère n'est pas nul, incrémentez le compteur.

- Appelez un enseignant pour valider votre travail -

Étape 9

Réalisez le sous-programme **`Atoui`** 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 65535). Par exemple, la chaîne "52146" devra renvoyer la valeur numérique 52146. Vous testerez votre sous-programme en respectant les structures utilisées précédemment. Aucun registre hormis **D0** ne doit être modifié en sortie du sous-programme.
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 : Code ASCII de `'0'` = `$30`
  Code ASCII de `'6'` = `$36`
  Valeur numérique du caractère `'6'` = `$36` - `$30` = `6`

  Remarque :
  En assembleur la notation `#'0'` est équivalente à la notation `#$30`, mais elle est bien plus explicite.

- 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 résultat.

  Exemple :
  Pour 52146, on aura une variable résultat 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.

- Appelez un enseignant pour valider votre travail -