T.P. 3
Calculatrice (partie 2)

Pour ce TP, le sous-programme **`GetInput`** est mis à votre disposition. Il permet de récupérer une chaîne de caractères saisie par l'utilisateur.

`GetInput` contient les entrées-sorties suivantes :

Entrées : **A0.L** pointe sur un *buffer* de 60 octets où sera stockée la chaîne après la saisie.
D1.B contient le numéro de colonne où sera affichée la chaîne à saisir.
D2.B contient le numéro de ligne où sera affichée la chaîne à saisir.

Sortie : Aucune.

Le *buffer* sera réservé par la directive d'assemblage `DS.B` qui permet de réserver un espace défini dans la mémoire.

Tapez le code source ci-dessous puis copiez le fichier `"GetInput.bin"` dans le même dossier.

```
                    ; ==============================
                    ; Initialisation des vecteurs
                    ; ==============================
                    org     $0
vector_000          dc.l    $ffb500
vector_001          dc.l    Main
vector_002_255      dcb.l   254,break_exception
break_exception     illegal

                    ; ==============================
                    ; Programme principal
                    ; ==============================
                    org     $1000

Main                lea     sBuffer,a0
                    clr.b   d1
                    clr.b   d2
                    jsr     GetInput

                    illegal

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

GetInput            incbin  "GetInput.bin"

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

sBuffer             ds.b    60
```

Exécutez-le à l'aide du débogueur en activant la fenêtre de sortie vidéo. Saisissez une chaîne de caractères puis appuyez sur la touche **[Entrée]**. Observez alors ce que contient la mémoire à l'adresse `buffer`. Recommencez jusqu'à bien comprendre le fonctionnement de **GetInput**. Attention, il n'est pas demandé d'exécuter pas à pas le contenu de **GetInput** afin de comprendre son fonctionnement interne, mais simplement de savoir l'utiliser.

Remarque

Le sous-programme **GetInput** ne prend en compte ni les majuscules, ni les accents, ni la répétition automatique des touches du clavier.

Étape 1

Réalisez le sous-programme **NextOp** qui détermine soit la position du prochain opérateur dans une chaîne, soit la position du caractère nul si elle ne contient aucun opérateur.

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

Sortie : **A0.L** renvoie l'adresse du premier opérateur de la chaîne s'il existe, ou l'adresse du caractère nul de la chaîne si elle ne contient aucun opérateur.

Ex : Avant :

A0 ↓									
'1'	'0'	'4'	'+'	'9'	'*'	'2'	'-'	'3'	0

Après :

			A0 ↓						
'1'	'0'	'4'	'+'	'9'	'*'	'2'	'-'	'3'	0

Étape 2

Réalisez le sous-programme **GetNum** qui détermine la valeur numérique du prochain nombre dans une chaîne avec une gestion des erreurs.

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

Sorties : **Z** renvoie `0` si la conversion contient une erreur.

Z renvoie `1` si la conversion est valide.

Si **Z** renvoie `0`, alors **D0.L** et **A0.L** ne sont pas modifiés.

Si **Z** renvoie `1`, alors :

- **D0.L** renvoie la valeur numérique du nombre.
- **A0.L** renvoie l'adresse du caractère situé juste après le nombre converti.

Ex : Avant :

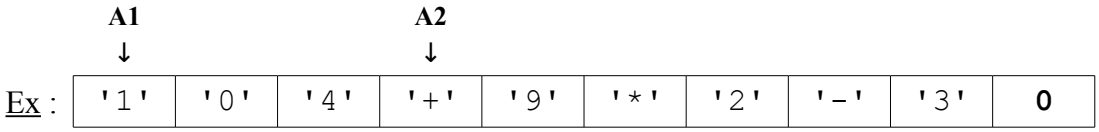
A0 ↓									
'1'	'0'	'4'	'+'	'9'	'*'	'2'	'-'	'3'	0

Après :

			A0 ↓						
'1'	'0'	'4'	'+'	'9'	'*'	'2'	'-'	'3'	0

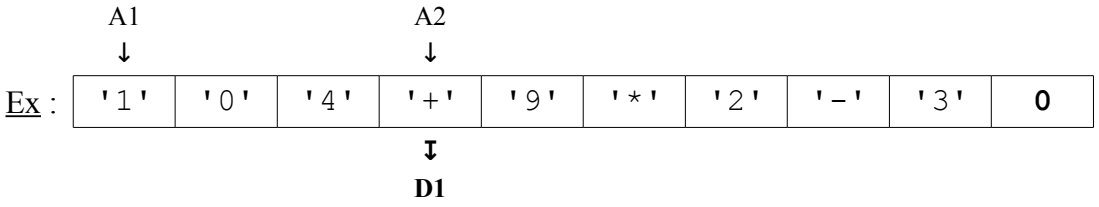
Avec **Z** = **1** et **D0** = **104**

- Utilisez les registres **A1** et **A2** pour encadrer un nombre.

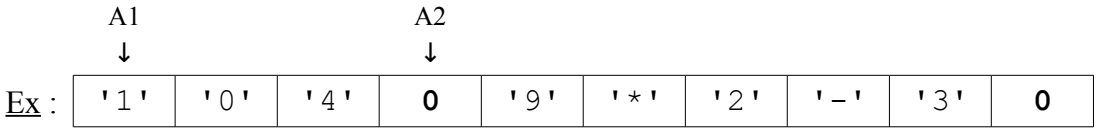


Chaque nombre est séparé par un opérateur (sauf le dernier qui est suivi d'un caractère nul). Aidez-vous de **`NextOp`**, qui détermine la position du prochain opérateur (ou du caractère nul), pour réaliser cet encadrement.

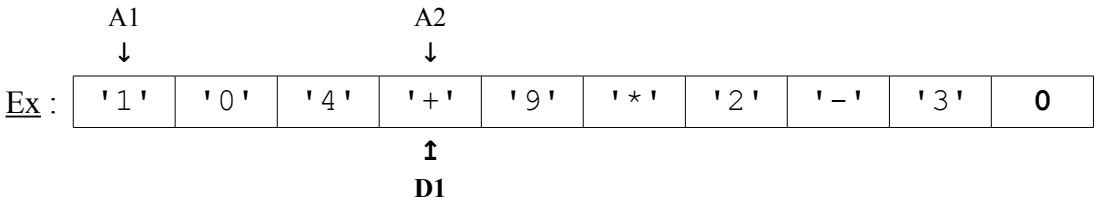
- Sauvegardez dans **D1** le caractère qui suit le nombre.



- Isolez le nombre du reste de la chaîne en remplaçant le caractère qui suit ce nombre par un caractère nul.



- Utilisez le sous-programme **`Conv`** pour obtenir la conversion du nombre isolé.
- Replacez le caractère sauvegardé dans **D1** à son emplacement d'origine.



- Faites attention aux valeurs de sortie et aux cas d'erreurs.

Étape 3

Réalisez le sous-programme **`GetExpr`** qui détermine la valeur numérique d'une expression contenue dans une chaîne avec une gestion des erreurs.

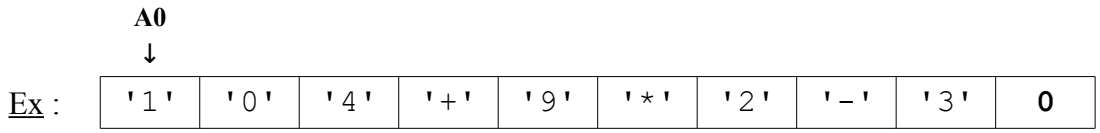
Entrée : **A0.L** pointe sur le premier caractère d'une chaîne contenant une expression.

Sorties : **Z** renvoie `0` si la conversion contient une erreur (elle est invalide).
Z renvoie `1` si la conversion est valide.
Si **Z** renvoie `0`, alors **D0.L** est perdu.
Si **Z** renvoie `1`, alors **D0.L** renvoie la valeur numérique de l'expression.

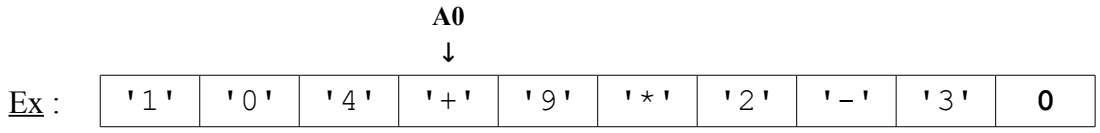
Indications

- Il est conseillé d'utiliser les registres comme suit :
  - ➔ **A0** doit servir à se déplacer dans la chaîne.
  - ➔ **D0** doit servir à la conversion de chacun des nombres de la chaîne.
  - ➔ **D1** doit servir à accumuler le résultat final.
  - ➔ **D2** doit servir à contenir l'opérateur (ou le caractère nul).

- Prenons l'exemple de la chaîne suivante :



- À l'aide de **`GetNum`**, récupérez la valeur numérique du premier nombre.



  - ➔ **A0** pointera sur le prochain opérateur.
  - ➔ **D0** contiendra la valeur numérique `104`.
  - ➔ **Z** contiendra `1`.

- Renvoyez une erreur si la conversion est erronée.

- Initialisez la valeur de **D1** (résultat final) avec celle du premier nombre.

- Il faut maintenant réaliser une boucle qui effectuera les opérations suivantes :
  - ➔ Lecture de l'opérateur (ou du caractère nul) dans **D2**.
  - ➔ Faire pointer **A0** sur le prochain nombre.
  - ➔ Si le contenu de **D2** est nul, alors on peut quitter sans erreur.
  - ➔ Sinon, il faut lancer la conversion du prochain nombre (et quitter si erreur).
  - ➔ Selon l'opérateur contenu dans **D2**, réaliser l'opération. C'est-à-dire additionner, soustraire, multiplier ou diviser le contenu de **D0** au contenu de **D1**.

- N'oubliez pas de traiter le cas de la division par zéro qui doit renvoyer une erreur.

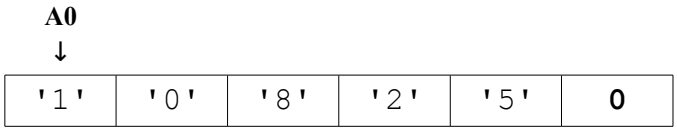
- Attention à l'instruction de division du 68000 (*cf.* manuel) qui stocke le quotient entier dans les 16 bits de poids faible de la destination et le reste dans les 16 bits de poids fort. Une extension de signe sera peut-être nécessaire.

Étape 4

Réalisez le sous-programme **`Uitoa`** qui convertit une valeur numérique entière, codée sur 16 bits non signés, en une chaîne de caractères ASCII.

Entrées : **A0.L** pointe sur un *buffer* où sera stockée la chaîne après la conversion.
D0.W contient une valeur numérique entière non signée à convertir.

Ex : Si **D0.W** = `10825`, la chaîne suivante devra être placée à l'adresse pointée par **A0** :



Indications

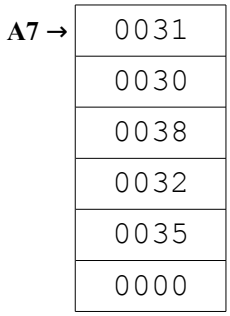
- La technique consiste à diviser successivement par 10 le nombre à convertir et à récupérer le reste de chaque division afin de pouvoir le transformer en caractère.

  Par exemple si **D0.W** = `10825`, on aura les cinq divisions suivantes :

Division	Quotient	Reste
10825/10	1082	5
1082/10	108	2
108/10	10	8
10/10	1	0
1/10	0	1

Il suffit de récupérer chaque reste et de le transformer en caractère ASCII.

- Un problème se pose : le premier reste que l'on récupère correspond au dernier caractère à stocker dans la chaîne. Les caractères nous arrivent dans l'ordre inverse.
- Après avoir transformé un reste en caractère, il n'est donc pas possible de le placer tout de suite dans la chaîne. Il faut dans un premier temps le placer dans la pile.
- Le premier caractère à empiler sera le caractère nul (avant même la première division).
- Seront empilés ensuite les caractères `'5'`, `'2'`, `'8'`, `'0'` et `'1'`.
- C'est la valeur nulle du quotient qui indiquera à quel moment arrêter les divisions.
- La pile aura alors l'aspect suivant (en représentation hexadécimale) :



Remarque
Il n'est pas possible d'empiler un octet. La taille minimale d'une donnée empilée est de 16 bits.

Rappel : `'0'` = `$30`

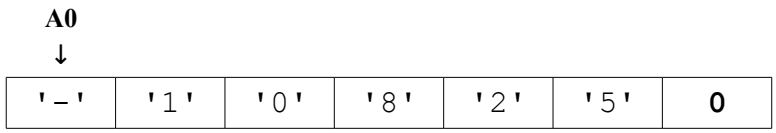
- Il suffit maintenant de dépiler tous les caractères (jusqu'au caractère nul) et de les placer dans la chaîne.
- Attention, il n'est pas possible de copier directement le contenu d'un caractère de la pile dans la chaîne. Dans la pile, un caractère est codé sur 16 bits, alors que dans la chaîne, un caractère est codé sur 8 bits. On a donc une source sur 16 bits et une destination sur 8 bits. Pour réaliser cette opération, il faudra passer par un registre. C'est-à-dire copier le caractère de la pile dans un registre (opération sur 16 bits), puis copier le registre dans la chaîne (opération sur 8 bits).
- Attention à l'instruction de division du 68000 (*cf*. manuel) qui stocke le quotient entier dans les 16 bits de poids faible de la destination et le reste dans les 16 bits de poids fort. Aidez-vous de l'instruction `SWAP` (*cf*. manuel) qui permet d'échanger les paquets de 16 bits d'un registre.
- Attention également à bien comprendre la taille prise en compte pour chacun des opérandes de l'instruction `DIVU`. Dans notre cas, seuls les 16 bits du dividende nous intéressent. Si les 32 bits sont pris en compte dans la division, il faudra ajouter un masque initialisant à zéro les 16 bits de poids fort du dividende juste avant la division.

Étape 5

Réalisez le sous-programme **`Itoa`** qui convertit une valeur numérique entière, codée sur 16 bits signés, en une chaîne de caractères ASCII.

Entrées : **A0.L** pointe sur un *buffer* où sera stockée la chaîne après la conversion.
D0.W contient une valeur numérique entière signée à convertir.

Ex : Si **D0.W** = `-10825`, la chaîne suivante devra être placée à l'adresse pointée par **A0** :



Indications

- Si le nombre est positif, il suffit d'appeler **`Uitoa`**.
- Si le nombre est négatif, il faut positionner le caractère `'-'` dans la chaîne puis appeler **`Uitoa`** avec l'opposé du nombre à convertir.

Étape 6

Réalisez le programme **`Main`** de la calculatrice en respectant l'affichage de l'exemple suivant :

```
Veuillez saisir une expression :
50*4+2
Resultat :
202
```