

# T.P. 6 – Corrigé

## Calculatrice (partie 3)

### Étape 1

```

GetNum      ; Sauvegarde les registres.
            movem.l d1/a1-a2,-(a7)

            ; Mémoire le début de la chaîne dans A1.
            movea.l a0,a1

            ; Cherche le prochain opérateur ou le caractère nul
            ; (c'est-à-dire le caractère qui suit le nombre),
            ; et mémorise sa position dans A2.
            jsr    NextOp
            movea.l a0,a2

            ; Sauvegarde l'opérateur ou le caractère nul dans D1.
            move.b (a2),d1

            ; Place un caractère nul juste après le nombre.
            clr.b (a2)

            ; Lance la conversion
            ; (avec l'adresse de départ comme paramètre dans A0).
            movea.l a1,a0
            jsr    Convert

            ; Si la conversion est valide,
            ; D0 contient la valeur numérique du nombre ASCII.
            ; On quitte sans erreur.
            beq    \true

\false      ; Sortie avec erreur.
            ; D0 n'a pas été modifié.
            ; A0 contient l'adresse de départ de la chaîne.
            ; Il ne reste plus qu'à restaurer le caractère sauvegardé dans D1.
            move.b d1,(a2)

            ; Et renvoyer Z = 0.
            andi.b #%11111011,ccr
            bra    \quit

\true       ; Sortie sans erreur.
            ; On commence par restaurer le caractère sauvegardé dans D1.
            move.b d1,(a2)

            ; On place l'adresse située après le nombre dans A0.
            movea.l a2,a0

            ; Et enfin, on renvoie Z = 1.
            ori.b  #%00000100,ccr

\quit       ; Restaure les registres.
            movem.l (a7)+,d1/a1-a2
            rts

```

**Étape 2**

```

GetExpr      ; Sauvegarde les registres.
              movem.l d1-d2/a0,-(a7)

              ; Conversion du premier nombre de l'expression (dans D0).
              ; Si erreur, on renvoie false.
              jsr      GetNum
              bne      \false

              ; Le premier nombre est chargé dans D1.
              ; (D1 contiendra le résultat des opérations successives.)
              move.l   d0,d1

\loop        ; L'opérateur ou le caractère nul est copié dans D2.
              ; S'il s'agit du caractère nul, on renvoie true (pas d'erreur).
              move.b   (a0)+,d2
              beq      \true

              ; Conversion du prochain nombre (dans D0).
              ; Si erreur, on renvoie false.
              jsr      GetNum
              bne      \false

              ; Détermine le type de l'opération (+, -, *, /).
              cmp.b    #'+',d2
              beq      \add

              cmp.b    #'-',d2
              beq      \subtract

              cmp.b    #'*',d2
              beq      \multiply

              bra       \divide

\add         ; Effectue l'opération puis passe au nombre suivant.
              add.l    d0,d1
              bra       \loop

\subtract    sub.l     d0,d1
              bra       \loop

\multiply    muls.w    d0,d1
              bra       \loop

\divide      ; Renvoie une erreur si une division par zéro est détectée.
              tst.w    d0
              beq      \false

              ; Le résultat entier de la division est sur 16 bits. Il faut
              ; réaliser une extension de signe pour l'avoir sur 32 bits.
              divs.w    d0,d1
              ext.l     d1
              bra       \loop

\false       ; Sortie avec erreur (Z = 0).
              andi.b    #%11111011,ccr
              bra       \quit

\true        ; Sortie sans erreur (Z = 1).
              ; (Avec la copie du résultat dans D0.)
              move.l    d1,d0

```

```

ori.b    %#00000100,ccr
quit      ; Restaure les registres puis sortie.
movem.l  (a7)+,d1-d2/a0
rts

```

### Étape 3

```

Uitoa      ; Sauvegarde les registres.
movem.l  d0/a0,-(a7)

          ; Empile le caractère nul de fin de chaîne.
clr.w    -(a7)

\loop      ; Limite D0 à 16 bits pour la division (seuls les 16 bits de
          ; poids faible contiennent le nombre à diviser).
andi.l   #$ffff,d0

          ; Divise D0 par 10 afin de récupérer le reste.
          ; Le quotient est placé dans les 16 bits de poids faible.
          ; Le reste est placé dans les 16 bits de poids fort.
divu.w   #10,d0

          ; Fait passer le reste dans les 16 bits de poids faible.
          ; (Le quotient passe dans les 16 bits de poids fort.)
swap     d0

          ; Convertit le reste en caractère ASCII (sur 8 bits).
addi.b   #'0',d0

          ; Empile le caractère ASCII (sur 16 bits).
move.w   d0,-(a7)

          ; Fait repasser le quotient dans les 16 bits de poids faible.
swap     d0

          ; Si le quotient n'est pas nul,
          ; il reste des chiffres à convertir.
          ; On passe donc au chiffre suivant.
tst.w    d0
bne      \loop

          ; Sinon tous les chiffres ont été traités,
          ; il ne reste plus qu'à les écrire dans la chaîne.
\writeChar ; Dépile le caractère (sur 16 bits).
move.w   (a7)+,d0

          ; Puis l'écrit dans la chaîne (sur 8 bits).
move.b   d0,(a0)+

          ; Continue tant que le caractère n'est pas nul.
bne      \writeChar

          ; Restaure les registres puis sortie.
movem.l  (a7)+,d0/a0
rts

```