Le but de ce TP et des suivants sera donc de développer petit à petit en assembleur un logiciel de détection de contours dans une image, en utilisant un algorithme simple.

Conversion en niveaux de gris

La conversion en niveaux de gris I d'un pixel RVB se fait selon la formule suivante : $I = R \times 0,299 + V \times 0,587 + B \times 0,114$

Nous devons donc convertir les coefficients décimaux en hexadécimal et les avons décalés de 8 BITS vers la gauche.

Bleu:	Vert:	Rouge:
0,114*2=0,228:0	<i>0,587*2=1,174:1</i>	0,299*2=0,598:0
0,228*2=0,456:0	0,174*2=0,348:0	0,598*2=1,196:1
<i>0,456*2=0,912:0</i>	<i>0,348*2=0,696:0</i>	0,196*2=0,392:0
<i>0,912*2=1,824:1</i>	0,696*2=1,392:1	0,392*2=0,784:0
<i>0,824*2=1,648:1</i>	0,392 *2=0,784 :0	0,784*2=1,568:1
<i>0,648*2=1,296:1</i>	<i>0,784*2=1,568:1</i>	0,568*2=1,136:1
0,296*2=0,592:0	<i>0,568*2=1,136:1</i>	0,136*2=0,272:0
0,592 *2=1,184 :1	0,136*2=0,272:0	0,272*2=0,544:0
Ce qui donne en hexa :		
0001 1101	1001 0110	0100 1100
01 Dh	09 6h	04 Dh

Traitement d'image - Première partie

ECX contiendra la taille (en pixels) de l'image : il nous servira donc de compteur qu'on décrémentera jusqu'à 0, moment où le test conditionnel permettra de ne pas revenir au début de la boucle, mais de continuer le programme (de l'achever en fait).

La boucle consiste à manipuler les octets du rouge, vert, et bleu d'un séparément au moyen de trois registres différents et à les additionner dans un registre unique. On sauvegarde en fin de boucle. Chaque itération traite un pixel de l'image.

```
; IMAGE.ASM
                                                   entête
; MI01 - TP Assembleur 2 à 5
; Réalise le traitement d'une image 32 bits.
.MODEL FLAT, C
                                                   segment données
.DATA
.CODE
                                                   segment code
; Sous-programme _process_image_asm
; Réalise le traitement d'une image 32 bits.
                                                   PILE :
                                                   Adresse image finale
; Entrées sur la pile :
                                                   Adresse image tampon 2
; Largeur de l'image (entier 32 bits)
                                                   Adresse image tampon 1
; Hauteur de l'image (entier 32 bits)
                                                   Adresse image source
; Pointeur sur l'image source (dépl. 32 bits)
                                                   Hauteur
; Pointeur sur l'image tampon 1 (dépl. 32 bits)
                                                   Largeur
; Pointeur sur l'image tampon 2 (dépl. 32 bits)
                                                   Adresse de retour
; Pointeur sur l'image finale (dépl. 32 bits)
; **************
              process_image_asm
process_image_asm
                      PROC NEA
                                                   Point d'entrée du sous-programme
               ebp
       push
                                                   sauvegarde ebp
       mov
               ebp, esp
                                                   ebp sur le sommet de la pile
               ebx
       push
       push
               esi
               edi
       push
                                                   ECX = largeur 8 = 4(EBP) + 4(EIP)
       mov
               ecx, [ebp + 8]
               ecx, [ebp + 12]
                                                   ECX = largeur*longueur
       imul
                                                   12 = 4(EBP) + 4(EIP) + 4(largeur)
       mov
               esi, [ebp + 16]
                                                   ESI : Pointeur sur le premier pixel
               edi, [ebp + 20]
                                                   EDI : pixel dans le tampon
       mov
suivant :
                                                   EAX : du dernier au premier pixel [255,0]
       MOV EAX, [ESI + ECX*4 - 4]
       MOV EDX, EAX
                                                   Duplication pour utiliser le 2eme masque
       MOV EBX, EAX
                                                   Duplication pour utiliser le 3eme masque
       AND EAX, 00FFh
                                                   Récupération ler octet
       IMUL EAX, 01Dh
                                                   Multiplication par 0,114
       AND EDX, OFFOOh
                                                   Récupération 2eme octet
                                                   Multiplication par 0,587
       IMUL EDX, 096h
       SHR EDX, 8
                                                   Décalage à droite de 8 BITs
       ADD EAX, EDX
                                                   addition
       AND EBX, OFF0000h
                                                   Récupération 3eme octet
       IMUL EBX, 04Ch
                                                   Multiplication par 0,299
       SHR EBX, 16
                                                   Décalage à droite de 16 BITs
```

```
ADD EAX, EBX
SHR EAX, 8
                                                      Addition
       MOV [EDI + ECX*4 - 4], EAX DEC ECX
                                                      Sauvegarde du nouveau pixel
                                                      Décrémentation du compteur
                                                      Si compteur égale à 0 on jump
       CMP ECX, 0
       JNE suivant
fin:
                edi
       pop
       рор
                esi
                ebx
       pop
                ebp
       pop
            ; Retour à la fonction MainWndProc
                       ENDP
process_image_asm
         END
```

Conclusion:

Ce TP nous a une fois de plus amené à manipuler les sauts conditionnels. Nous avons également été amenés à prêter attention à la taille des registres que nous utilisions en fonction de des mnémoniques utilisées. Enfin nous avons dû utiliser à bon escient les décalages à droite afin de pouvoir réaliser les additions comme il faut et les multiplications sans déborder. Evidemment nous avons dû faire attention à employer aussi peu d'accès mémoire que possible.