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 x 0,299 + V x 0,587 + B x 0,114

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

Bleu:

Vert:

Rouge:

Ce qui donne en hexa :

0001 1101

01 Dh

1001 0110

09 6h

0100 1100

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**  **;**  **; MI01 - TP Assembleur 2 à 5**  **;**  **; Réalise le traitement d'une image 32 bits.**  **.686**  **.MODEL FLAT, C**  **.DATA**    **.CODE**  **; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **; Sous-programme \_process\_image\_asm**  **;**  **; Réalise le traitement d'une image 32 bits.**  **;**  **; Entrées sur la pile :**  **; Largeur de l'image (entier 32 bits)**  **; Hauteur de l'image (entier 32 bits)**  **; Pointeur sur l'image source (dépl. 32 bits)**  **; Pointeur sur l'image tampon 1 (dépl. 32 bits)**  **; Pointeur sur l'image tampon 2 (dépl. 32 bits)**  **; Pointeur sur l'image finale (dépl. 32 bits)**  **; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **PUBLIC process\_image\_asm**  **process\_image\_asm PROC NEA**    **push ebp**  **mov ebp, esp**  **push ebx**  **push esi**  **push edi**    **mov ecx, [ebp + 8]**  **imul ecx, [ebp + 12]**  **mov esi, [ebp + 16]**  **mov edi, [ebp + 20]**  **;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **suivant :**  **MOV EAX, [ESI + ECX\*4 - 4]**  **MOV EDX, EAX**  **MOV EBX, EAX**    **AND EAX, 00FFh**  **IMUL EAX, 01Dh**    **AND EDX, 0FF00h**  **IMUL EDX, 096h**  **SHR EDX, 8**  **ADD EAX, EDX**    **AND EBX, 0FF0000h**  **IMUL EBX, 04Ch**  **SHR EBX, 16**  **ADD EAX, EBX**  **SHR EAX, 8**    **MOV [EDI + ECX\*4 - 4], EAX**  **DEC ECX**  **CMP ECX, 0**  **JNE suivant**  **;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***    **fin:**  **pop edi**  **pop esi**  **pop ebx**  **pop ebp**  **ret ; Retour à la fonction MainWndProc**    **process\_image\_asm ENDP**  **END** | entête  segment données  segment code  PILE :  Adresse image finale  Adresse image tampon 2  Adresse image tampon 1  Adresse image source  Hauteur  Largeur  Adresse de retour  Point d'entrée du sous-programme  sauvegarde ebp  ebp sur le sommet de la pile  ECX = largeur 8 = 4(EBP) + 4(EIP)  ECX = largeur\*longueur  12 = 4(EBP) + 4(EIP) + 4(largeur)  ESI : Pointeur sur le premier pixel  EDI : pixel dans le tampon  EAX : du dernier au premier pixel [255,0]  Duplication pour utiliser le 2eme masque  Duplication pour utiliser le 3eme masque  Récupération 1er octet  Multiplication par 0,114  Récupération 2eme octet  Multiplication par 0,587  Décalage à droite de 8 BITs  addition  Récupération 3eme octet  Multiplication par 0,299  Décalage à droite de 16 BITs  Addition  Sauvegarde du nouveau pixel  Décrémentation du compteur  Si compteur égale à 0 on jump |

**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.