

Développement d'applications graphiques en assembleur ARM

Objectifs du TP

A l'issue de ce TP vous devez pouvoir montrer un petit vaisseau spatial dessiné en assembleur qui se déplace à l'aide des touches de direction. La gestion des touches et des déplacements, de même que la gestion de la boucle principale d'animation, resteront uniquement en C. Cette partie principale en C appellera simplement des sous-programmes en assembleur pour effacer et afficher le vaisseau.

Les paragraphes suivants proposent une approche pas à pas vous amenant à ce résultat. Vous êtes libres de proposer des variantes si vous en avez le temps, à condition de réaliser un travail de développement graphique en assembleur.

Création de projet

Commencez par créer un nouveau projet dans un répertoire personnel, comme pour le dernier exercice du TP 1. Attention avec l'environnement de développement VHAM il est essentiel de créer un répertoire pour chaque nouveau projet.

Premier sous-programme en assembleur

Les détails et explications complètes concernant les sous-programmes assembleur sont données en cours. Pour l'instant on se contentera d'insérer des lignes de code en assembleur dans un cadre prédéfini. Le but (modeste) de ce 1er sous-programme est d'afficher un pixel blanc au milieu de l'écran.

Le principe est le suivant :

```
initialiser le registre r2 avec l'adresse en RAM vidéo du pixel de coordonnées (120,80)
initialiser le registre r3 avec le code couleur BGR 15 bits correspondant au blanc
utiliser une instruction de stockage d'un demi-mot en mémoire : strh
pour écrire le contenu de r3 à l'adresse spécifiée par r2
```

- 1/ Depuis l'éditeur VHAM, éditer le fichier source du projet `asm.s` Tous les fichiers sources sont accessibles dans le panneau de gauche de VHAM
- 2/ Vous trouverez au début de ce fichier source un sous programme assembleur vide (qui ne fait rien) qui s'appelle `asmProc`
- 3/ Ajoutez les lignes d'instructions en assembleur au niveau du commentaire (sans modifier le reste)

```
@ DEVELOPPER ICI le code assembleur de la procédure asmProc
ldr    r2,=0x06000000+2*120+480*80
ldr    r3,=0x7FFF
strh   r3,[r2]
```

- 4/ On a écrit un sous-programme assembleur mais à ce stade le programme dans son ensemble ne fait rien...

Il faut appeler le sous-programme assembleur depuis le *main* pour lancer son exécution. Pour l'instant on fait des tests donc on se contentera d'appeler `asmProc` une seule fois, entre l'initialisation du mode graphique et la boucle infinie d'animation.

L'appel d'un sous-programme assembleur depuis le C se fait comme pour l'appel d'un sous-programme écrit en C.

Le fichier d'en-tête de projet `projet.h` indique le prototype d'`asmProc` : c'est une procédure sans argument qui ne retourne rien.

Dans le *main* (fichier source `main.c`) l'appel se fait donc ainsi :

```
setMode3(); // Initialisation mode graphique
asmProc(); // Appel du sous programme assembleur asmProc
while(1) // Boucle principale d'animation -vide pour l'instant-
{
}
```

- 5/ Compiler (F5) et exécuter (F6) Pour plus de lisibilité des pixels dans le simulateur vous pouvez cocher `menu->Options->Filter->Simple 2x`

Petit segment

En reprenant le même principe que la procédure `petitSegment` du dernier exercice du TP 1, une boucle "répéter 10 fois" va permettre de colorier des pixels consécutifs en mémoire vidéo et de dessiner un segment blanc de 10 pixels. On modifie donc le code dans `asmProc` comme suit :

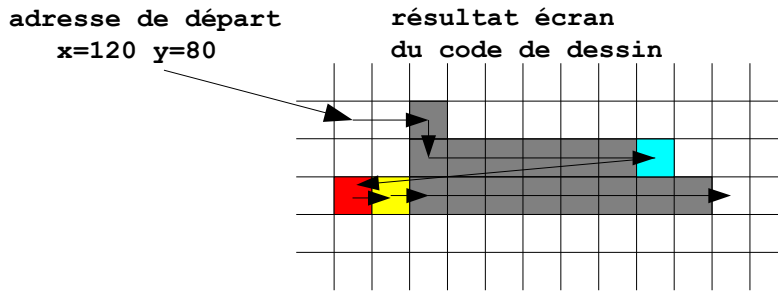
```
@ DEVELOPPER ICI le code assembleur de la procédure asmProc
ldr    r2,=0x06000000+2*120+480*80 @ Adresse du pixel de départ
ldr    r3,=0x7FFF                @ Couleur des pixels à colorier
ldr    r0,=10                    @ Compteur de boucle "répéter 10 fois"
BoucleSeg: @ Etiquette de début de boucle
    strh   r3,[r2]                @ Colorier le pixel courant
    add    r2,r2,#2                @ Adresse du pixel suivant (déplacement d'un pixel à droite)
    subs   r0,r0,#1                @ Décrémenter le compteur de boucle et mettre NZCV à jour
    bne   BoucleSeg                @ Recommencer la boucle tant que résultat du décrétement non nul
```

Testez le code ci dessus puis essayez de le modifier pour faire :

- Un segment vertical (revoir indications dernier exercice TP 1)
- Un segment diagonal à 45°... quel est le problème qui apparaît ? Comment le résoudre ?

Petit vaisseau spatial

Sur ce principe de déplacement relatif d'une adresse, on peut visiter successivement et colorier avec différentes couleurs les pixels d'un sprite (petit dessin) à l'écran. Sur le schéma les flèches indiquent les pixels parcourus par l'adresse (dans r2). On fera toujours en sorte que l'adresse de départ corresponde au pixel du coin supérieur gauche du rectangle dans lequel le sprite s'inscrit.



```
ldr    r2,=0x06000000+2*120+480*80 @ Adresse de départ...

add    r2,r2,#4      @ déplacement de 2 pixels à droite
ldr    r3,=0x5294    @ couleur grise
strh   r3,[r2]      @ colorier un pixel
add    r2,r2,#480    @ déplacement d'1 pixel vers le bas
ldr    r0,=6        @ on va colorier un segment horizontal de 6 pixels
BoucleSeg1:
    strh   r3,[r2]    @ colorier un pixel du segment
    add    r2,r2,#2    @ adresse pixel suivant
    subs   r0,r0,#1   @ décrémenter compteur de boucle...
    bne   BoucleSeg1 @ répéter tant que compteur non nul
ldr    r3,=0x7FE0    @ couleur cyan (bleu + vert)
strh   r3,[r2]      @ colorier un pixel (cockpit)
add    r2,r2,#480-16 @ déplacement d'1 pixel vers le bas et de 8 à gauche
ldr    r3,=0x001F    @ couleur rouge
strh   r3,[r2]      @ colorier un pixel (flamme réacteur)
add    r2,r2,#2      @ déplacement d'1 pixel à droite
ldr    r3,=0x03FF    @ couleur jaune
strh   r3,[r2]      @ colorier un pixel (flamme réacteur)
add    r2,r2,#2      @ déplacement d'1 pixel à droite
ldr    r3,=0x5294    @ couleur grise
ldr    r0,=8        @ on va colorier un segment horizontal de 8 pixels
BoucleSeg2:
    strh   r3,[r2]    @ colorier un pixel du segment
    add    r2,r2,#2    @ adresse pixel suivant
    subs   r0,r0,#1   @ décrémenter compteur de boucle...
    bne   BoucleSeg2 @ répéter tant que compteur non nul
```

Remplacez vos essais précédents par ce code (copiez/collez depuis l'énoncé pour aller plus vite) dans asmProc, testez le.

Si vous avez le temps dans la séance, réalisez une variante simple (changez les dimensions ou ajoutez un élément graphique, un aileron...)

Remarque : pour conserver une trace de vos essais précédents vous pouvez ouvrir un nouveau projet, ou plus simplement copier le fichier asm.s

Il est également possible de mettre en commentaire des blocs de code assembleur en sélectionnant des lignes puis click droit et "Comment Add"

Prise en compte de coordonnées d'affichage

On souhaite maintenant transformer la procédure asmProc pour qu'elle puisse prendre en compte 2 paramètres x et y au moment de l'appel.

Les valeurs des 2 paramètres fournis au moment de l'appel (au niveau du C) se retrouveront automatiquement dans les registres r0 et r1 au début de la procédure asmProc. Il n'y a aucune déclaration particulière à faire au niveau de l'assembleur, il suffit d'utiliser les valeurs dans r0 et r1.

1/ Modifier le prototype de asmProc dans **projet.h** pour indiquer au compilateur C le format de l'appel : `void asmProc(int x,int y) ;`

2/ Modifier au niveau du **main** l'appel à asmProc pour fournir des coordonnées : par exemple `asmProc(60,40) ;`

3/ Remplacer l'initialisation de r2 par une constante (au début du code : `ldr r2,=0x06000000+2*120+480*80`)

par le code vu au TD 2 qui effectue le calcul d'adresse d'un pixel (dans r2) à partir des coordonnées x et y (qui sont dans r0 et r1)

4/ Recompiler et vérifier que le même graphique de sprite s'affiche maintenant à la position indiquée depuis l'appel en C

Note : il est exclu d'initialiser r2 avec une instruction de la forme `ldr r2,=0x06000000+2*r0+480*r1` car il s'agit d'un calcul effectué à la compilation par l'assembleur. Seules des valeurs constantes (connues dès la compilation) sont possibles dans une telle expression.

Animation interactive

En reprenant le code C de l'exemple 2 du TP 1 et en l'adaptant (enlevez l'affichage de l'état des touches, nettoyez les variables inutiles ...) donnez une version interactive du programme avec déplacement du vaisseau. La principale différence est d'appeler asmProc au lieu de asmDrawBlock au niveau de // Afficher nouvelle position

Améliorations ...

- Modifier le code C de gestion des déplacements pour bloquer le vaisseau dans le rectangle écran (plutôt que de passer d'un côté à l'autre)

- Un bloc de commentaire en bas du fichier asm.c décrit comment créer de nouvelles procédures assembleur : créez un nouveau sous programme qui reprend le code du dessin de vaisseau mais colorie les pixels en noir. Cette procédure remplacera asmDrawBlock pour "Effacer l'ancienne position".