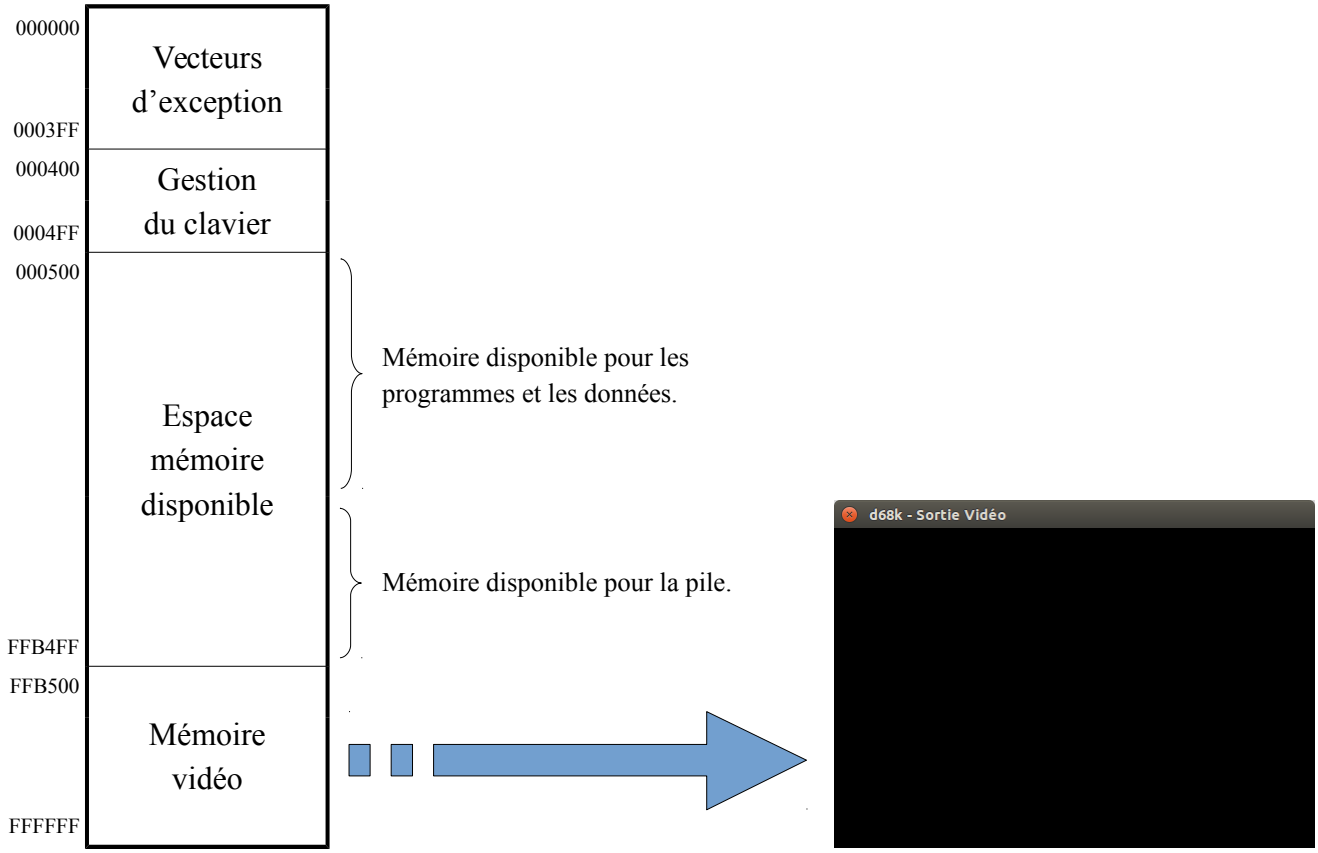
T.P. 8
Space Invaders (partie 1)

Dans cette série de travaux pratiques, vous allez réaliser une version du jeu *Space Invaders*. Vous devrez progresser étape par étape en respectant le cheminement qui vous sera proposé.

Commençons par quelques règles simples que vous devrez appliquer tout au long de votre progression :

- Votre fichier source se divisera en cinq parties bien distinctes : définitions des constantes, initialisation des vecteurs, programme principal, sous-programmes et données.
- Votre fichier source contiendra un seul programme principal et plusieurs sous-programmes.
- Vous devrez adapter votre programme principal afin de tester le sous-programme en cours de développement.
- **À l'exception des registres utilisés pour renvoyer une valeur de sortie, aucun registre ne devra être modifié en sortie de vos sous-programmes.**

Pour mener à bien les étapes qui vont suivre, il est nécessaire de comprendre comment réaliser un affichage dans la fenêtre de sortie vidéo de l'émulateur. Ce dernier utilise une partie de la mémoire du 68000 comme mémoire vidéo (un peu comme le ferait une carte graphique). Cette mémoire vidéo est située de l'adresse $FFB500_{16}$ à l'adresse $FFFFFF_{16}$. Plus précisément, voici comment s'organise la mémoire de notre système :



Concernant les vecteurs d'exception, nous initialiserons uniquement deux vecteurs :

- **Le vecteur 0 (situé à l'adresse 0) qui sert à initialiser le pointeur de pile superviseur (SSP).**

  Afin de pouvoir empiler des données dans l'espace mémoire disponible tout en bénéficiant d'un maximum de place, il est judicieux d'initialiser le pointeur de pile à l'adresse $FFB500_{16}$ (c'est-à-dire l'adresse de départ de la mémoire vidéo). En effet, pour empiler des données, il faut commencer par décrémenter le pointeur de pile ; le sommet de la pile sera donc toujours inférieur à la mémoire vidéo et n'empiétera jamais sur cette dernière. Nous placerons nos programmes et nos données à partir de l'adresse 500_{16}, ce qui permettra d'avoir un écart maximum entre le programme en cours d'exécution et le pointeur de pile. Cet écart sera largement suffisant pour qu'aucun chevauchement ne se produise.

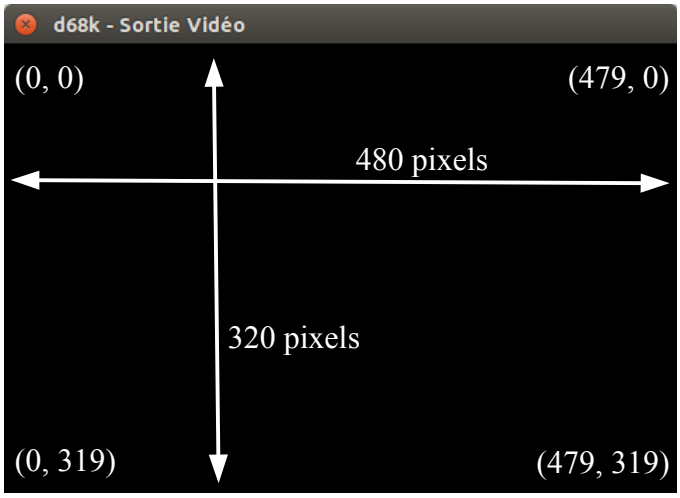
- **Le vecteur 1 (situé à l'adresse 4) qui sert à initialiser le compteur programme (PC).**

  Vous avez déjà utilisé ce vecteur dans les travaux pratiques précédents. Pour rappel, il doit contenir le point d'entrée du programme. C'est-à-dire l'adresse qui sera chargée dans le registre **PC** après la mise sous tension ou la réinitialisation du 68000.

La zone mémoire de 400_{16} à $4FF_{16}$ est réservée à la gestion du clavier. Nous y reviendrons lorsque nous aurons besoin de détecter différents appuis sur les touches du clavier. Pour l'instant, retenez simplement qu'il ne faut absolument pas écrire dans cette zone mémoire.

Intéressons-nous maintenant en détail à la mémoire vidéo. Le principe de base est assez simple : un bit de la mémoire vidéo correspond à un pixel de la fenêtre vidéo. Si un bit vidéo est à 0, le pixel associé est affiché en noir. Si ce même bit est à 1, le pixel associé est affiché en blanc.

Un pixel possède des coordonnées (abscisse, ordonnée). La résolution en pixels de la fenêtre vidéo est de 480 × 320 (largeur × hauteur). Les pixels se répartissent de la façon suivante :



La première adresse vidéo ($FFB500_{16}$) contient huit bits, donc huit pixels. Il s'agit des huit pixels situés en haut à gauche. L'adresse suivante ($FFB501_{16}$) contient les 8 pixels suivants et ainsi de suite jusqu'à la fin de la première ligne (ligne d'ordonnée 0).

Adresse :	**$FFB500_{16}$**								**$FFB501_{16}$**							
Pixel :	0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0	8,0	9,0	10,0	11,0	12,0	13,0	14,0	15,0

Une ligne est composée de 60 octets (480 / 8). Pour obtenir l'adresse de la deuxième ligne (ligne d'ordonnée 1), il faut ajouter 60 ($60_{10} = 3C_{16}$) à l'adresse de la première ligne, de même pour passer de la deuxième ligne à la troisième ligne et ainsi de suite. Le tableau ci-dessous donne un aperçu des différentes adresses vidéo.

Ligne	**Adresse (en hexadécimal)**				
0	FFB500	FFB501	FFB502	…	FFB53B
1	FFB53C	FFB53D	FFB53E	…	FFB577
2	FFB578	FFB579	FFB57A	…	FFB5B3
…	…	…	…	…	…
318	FFFF88	FFFF89	FFFF8A	…	FFFFC3
319	FFFFC4	FFFFC5	FFFFC6	…	FFFFFF

Par exemple :

- Le pixel de coordonnées (0, 0) est le bit 7 de l'adresse $FFB500_{16}$;
- Le pixel de coordonnées (0, 319) est le bit 7 de l'adresse $FFFFC4_{16}$;
- Le pixel de coordonnées (479, 0) est le bit 0 de l'adresse $FFB53B_{16}$;
- Le pixel de coordonnées (479, 319) est le bit 0 de l'adresse $FFFFFF_{16}$;
- Le pixel de coordonnées (3, 2) est le bit 4 de l'adresse $FFB578_{16}$;
- Le pixel de coordonnées (21, 318) est le bit 2 de l'adresse $FFFF8A_{16}$.

Afin d'éviter de manipuler directement des valeurs numériques, nous allons définir plusieurs constantes grâce à la directive `EQU` (*cf.* cours). Par exemple, la constante `VIDEO_START` sera utilisée pour faire référence à l'adresse de départ de la mémoire vidéo. Ainsi, l'adresse $FFB500_{16}$ ne devra apparaître qu'une seule fois dans tout votre code source, et ce, au moment de la définition de la constante `VIDEO_START`. Par la suite, à chaque fois que vous aurez besoin de l'adresse de départ de la mémoire vidéo, vous utiliserez la constante et non pas la valeur numérique.

Votre code source devra donc respecter la structure suivante :

```
                    ; ==============================
                    ; Définition des constantes
                    ; ==============================

                    ; Mémoire vidéo
                    ; ------------------------------

VIDEO_START         equ     $ffb500                         ; Adresse de départ
VIDEO_WIDTH         equ     480                             ; Largeur en pixels
VIDEO_HEIGHT        equ     320                             ; Hauteur en pixels
VIDEO_SIZE          equ     (VIDEO_WIDTH*VIDEO_HEIGHT/8)    ; Taille en octets
BYTE_PER_LINE       equ     (VIDEO_WIDTH/8)                 ; Nombre d'octets par ligne

                    ; ==============================
                    ; Initialisation des vecteurs
                    ; ==============================

                    org     $0

vector_000          dc.l    VIDEO_START                     ; Valeur initiale de A7
vector_001          dc.l    Main                            ; Valeur initiale du PC

                    ; ==============================
                    ; Programme principal
                    ; ==============================

                    org     $500

Main                ; ...
                    ; ...
                    ; ...

                    illegal

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

                    ; ...
                    ; ...
                    ; ...

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

                    ; ...
                    ; ...
                    ; ...
```

Étape 1

Dans cette étape, vous allez commencer par quelque chose de très simple afin d'assimiler le fonctionnement de la mémoire vidéo. Réalisez le sous-programme **`FillScreen`** qui remplit la mémoire vidéo avec une valeur numérique entière. Le remplissage se fera par mot de 32 bits.
Entrée : **D0.L** = Entier sur 32 bits avec lequel sera remplie la mémoire vidéo.

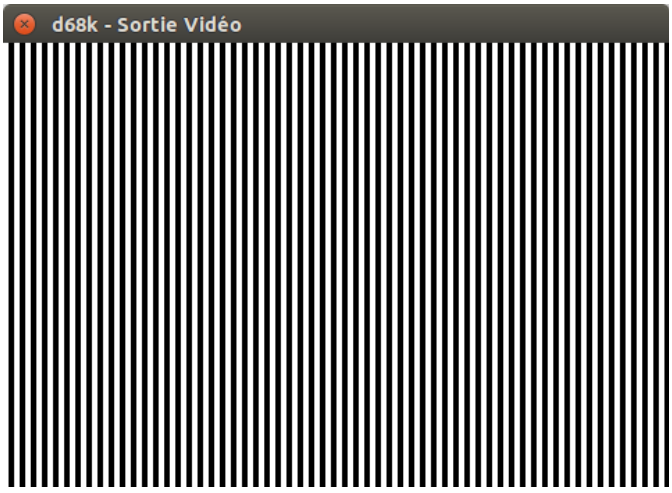
Vous testerez votre sous-programme à l'aide du programme principal ci-dessous. Utilisez de préférence la touche **`[F10]`** de l'émulateur pour exécuter les appels au sous-programme. Essayez également de laisser la touche **`[F11]`** appuyée, cela risque de prendre du temps, mais vous pourrez constatez les changements sur l'écran et mieux comprendre le fonctionnement de votre programme. Attention, si vous utilisez la touche **`[F9]`**, l'émulation ira trop vite et vous n'aurez pas le temps de voir les changements qui s'opèrent. Pour rappel, la fenêtre de sortie vidéo s'obtient après un appui sur la touche **`[F4]`**.

```
Main                ; Test 1
                    move.l  #$ffffffff,d0
                    jsr     FillScreen

                    ; Test 2
                    move.l  #$f0f0f0f0,d0
                    jsr     FillScreen

                    ; Test 3
                    move.l  #$fff0fff0,d0
                    jsr     FillScreen

                    ; Test 4
                    moveq.l #$0,d0
                    jsr     FillScreen

                    illegal
```

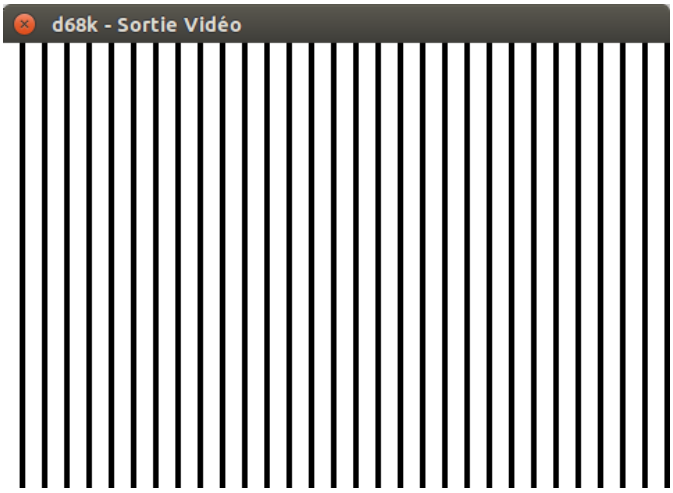
Vérifiez que vous obtenez bien un résultat identique aux captures d'écran suivantes :

Test 1
(écran blanc)

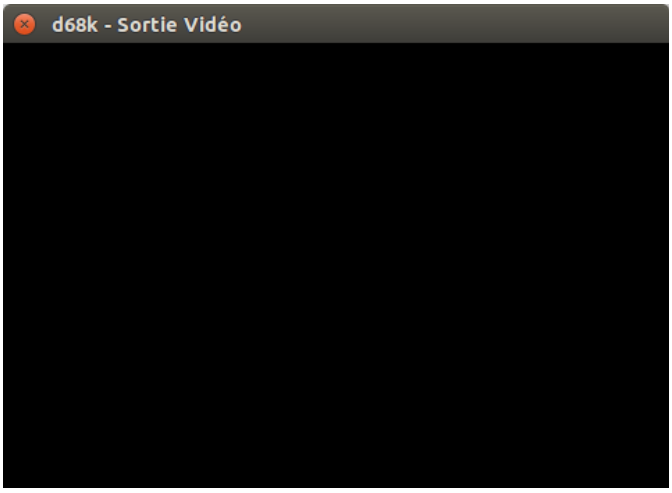
Test 2
(rayures noires et blanches de largeur identique)



Test 3
(rayures blanches plus larges que rayres noires)



Test 4
(écran noir)



Étape 2

Réalisez le sous-programme **`HLines`** qui dessine des rayures horizontales noires et blanches. La hauteur des rayures noires sera de 8 pixels. Idem pour les rayures blanches.

Capture d'écran du résultat attendu :