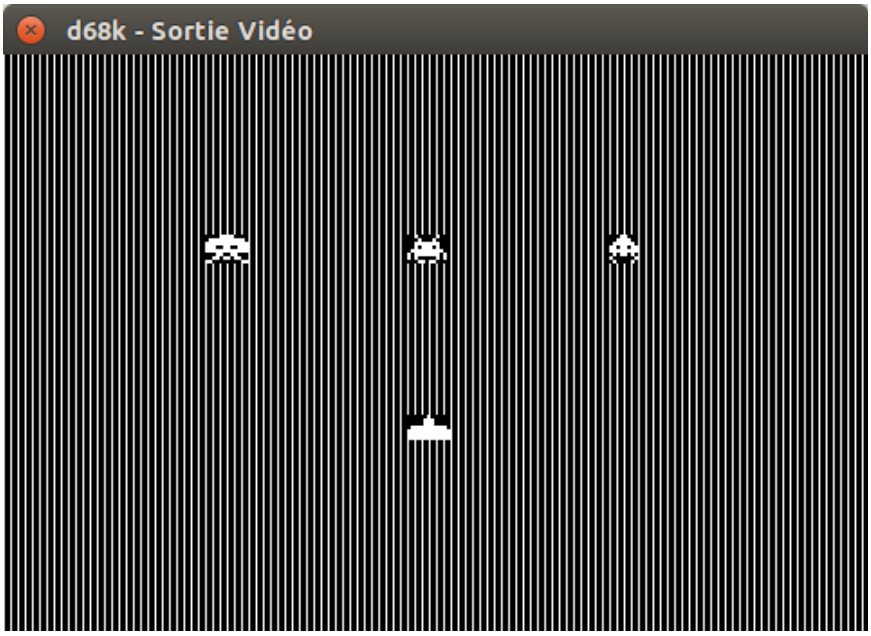
T.P. 1
Space Invaders (partie 4)

Étape 1

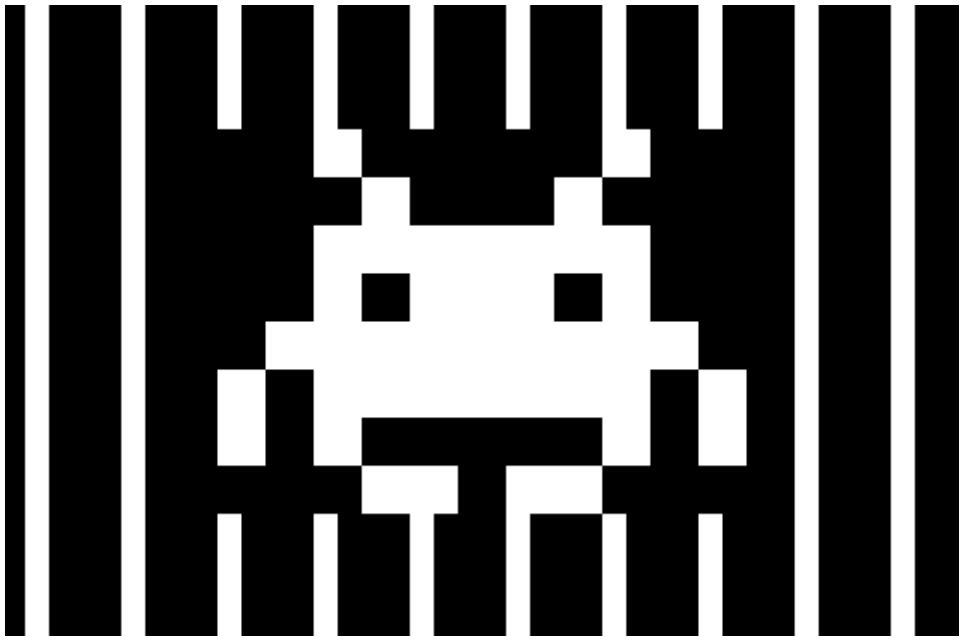
Pour l'instant, l'affichage des bitmaps ne prend pas en compte un éventuel arrière-plan. En effet, si un bitmap doit s'afficher par-dessus un autre bitmap ou par-dessus une image de fond, le bitmap écrase complètement l'arrière-plan. Cela pose un problème lorsque les pixels noirs d'un bitmap vont remplacer les pixels blancs de l'image de fond. Par exemple, reprenons le programme principal de la précédente étape en ajoutant un motif à l'arrière-plan à l'aide du sous-programme **`FillScreen`**.

```
Main                move.l  #$88888888,d0
                    jsr     FillScreen

                    lea     InvaderA_Bitmap,a0
                    lea     VIDEO_START+14+100*BYTE_PER_LINE,a1
                    jsr     CopyBitmap

                    lea     InvaderB_Bitmap,a0
                    lea     VIDEO_START+28+100*BYTE_PER_LINE,a1
                    jsr     CopyBitmap

                    lea     InvaderC_Bitmap,a0
                    lea     VIDEO_START+42+100*BYTE_PER_LINE,a1
                    jsr     CopyBitmap

                    lea     Ship_Bitmap,a0
                    lea     VIDEO_START+28+200*BYTE_PER_LINE,a1
                    jsr     CopyBitmap

                    illegal
```

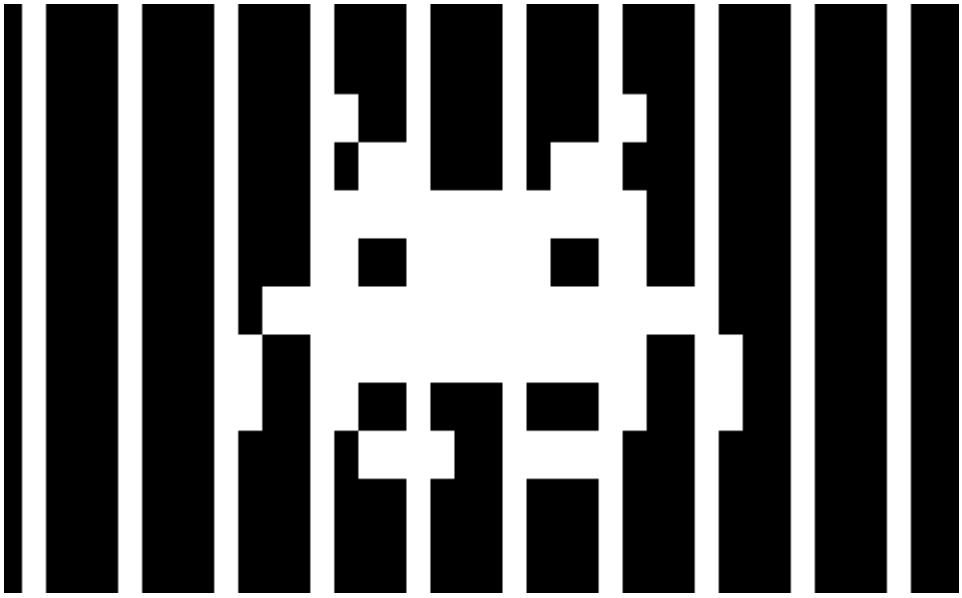
Le résultat est le suivant :



Si nous zoomons sur un envahisseur, voici ce que nous obtenons :



On voit bien que certaines lignes verticales sont effacées par les pixels noirs du bitmap. L'objectif de cette étape va être de supprimer cet effet. Nous souhaitons afficher uniquement les pixels blancs du bitmap et ne pas toucher à l'arrière-plan lorsque les pixels d'un bitmap sont noirs. Voici l'effet que nous souhaitons obtenir :



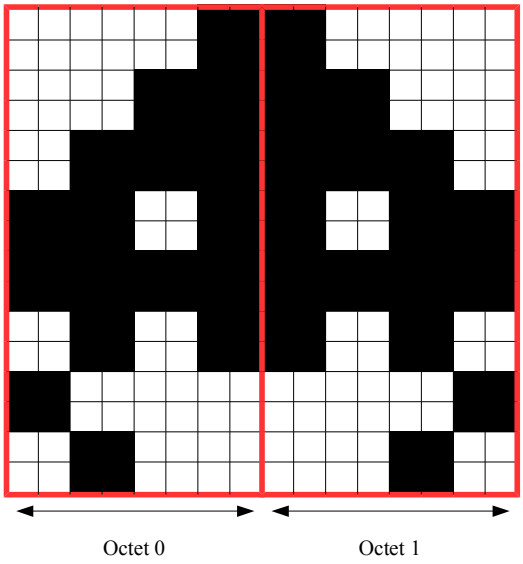
Modifiez votre sous-programme **`CopyLine`** afin d'obtenir l'effet escompté. Il s'agit simplement de remplacer l'instruction `MOVE`, qui copie un octet du bitmap vers la mémoire vidéo, par un opérateur logique. Attention, les modes d'adressage disponibles avec les instructions d'opérations logiques ne sont pas aussi nombreux que ceux disponibles avec l'instruction `MOVE`.

Étape 2

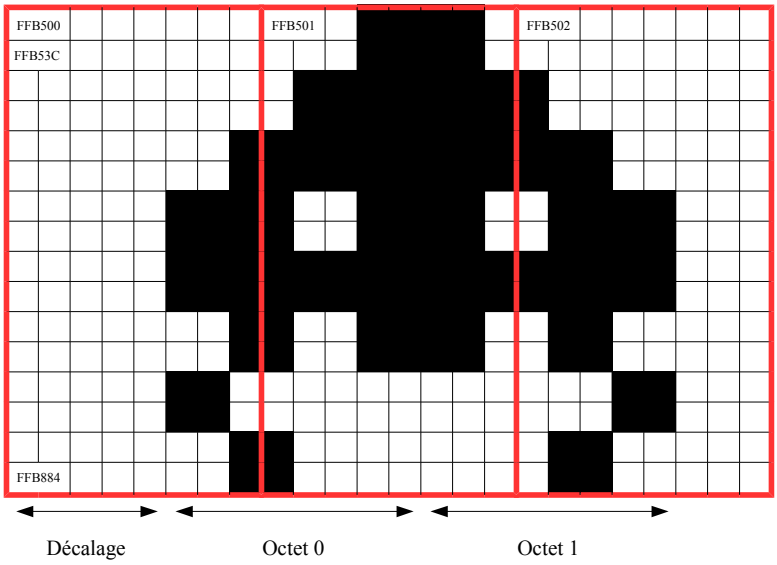
Le sous-programme **`CopyBitmap`** nous permet de placer un bitmap dans la mémoire vidéo, c'est-à-dire de l'afficher à l'écran. Mais ce sous-programme ne nous permet pas de placer un bitmap au pixel près. En effet, nous devons choisir l'adresse vidéo où sera placé le bitmap et non pas ses coordonnées en pixels. L'objectif des prochaines étapes va donc être de réaliser un sous-programme qui affichera un bitmap à partir de coordonnées en pixels.

Afin d'afficher un bitmap à des coordonnées bien précises, il faut introduire la notion de décalage. Par exemple, pour afficher un bitmap aux coordonnées (5, 0), l'adresse d'affichage sera `VIDEO_START` et le décalage sera de 5. C'est-à-dire que le premier octet de la matrice de points du bitmap sera décalé de 5 bits vers la droite et devra être affiché à l'adresse `VIDEO_START`. Les bits sortants quant à eux devront être affichés à l'adresse `VIDEO_START+1`. Pour chaque octet de la matrice de points du bitmap, l'affichage ne se fera plus sur un octet de la mémoire vidéo, mais sur deux octets. Prenons l'exemple du bitmap suivant qui à une largeur de 2 octets :

Bitmap stocké mémoire :



Bitmap affiché à l'écran (dans la mémoire vidéo) aux coordonnées (5,0) :



Réalisez le sous-programme **`PixelToAddress`** qui renvoie une adresse vidéo et un décalage en fonction de coordonnées en pixels.

Entrées : **D1.W** = Abscisse en pixels.
D2.W = Ordonnée en pixels.

Sorties : **A1.L** = Adresse vidéo du pixel.
D0.W = Décalage du pixel.

Vous trouverez ci-dessous un tableau qui regroupe quelques exemples de ce que doit renvoyer **`PixelToAddress`** en fonction de différentes coordonnées en pixels. Utilisez ces valeurs pour écrire un programme principal qui testera votre sous-programme.

Abscisse	Ordonnée	Adresse vidéo (hexadécimal)	Décalage
0	0	FFB500	0
3	0	FFB500	3
8	0	FFB501	0
10	0	FFB501	2
0	1	FFB53C	0
11	1	FFB53D	3
2	319	FFFFC4	2
400	319	FFFFF6	0
479	319	FFFFFF	7

Étape 3

Nous allons effectuer quelques modifications aux sous-programmes **`CopyBitmap`** et **`CopyLine`**. Ils devront maintenant prendre en compte un décalage en pixels. Pour cela, nous allons leur ajouter un paramètre d'entrée.

`CopyBitmap` : Copie la matrice de points d'un bitmap dans la mémoire vidéo avec la prise en compte d'un décalage en pixels.

Entrées : **A0.L** = Adresse du bitmap.
A1.L = Adresse vidéo où copier la matrice de points.
D0.W = Décalage en pixels.

`CopyLine` : Copie une ligne d'un bitmap dans la mémoire vidéo avec la prise en compte d'un décalage en pixels.

Entrées : **A0.L** = Adresse de la ligne du bitmap.
A1.L = Adresse vidéo où copier la ligne.
D0.W = Décalage en pixels.
D3.W = Largeur de la ligne en octets.

Sortie : **A0.L** = Adresse de la prochaine ligne du bitmap.

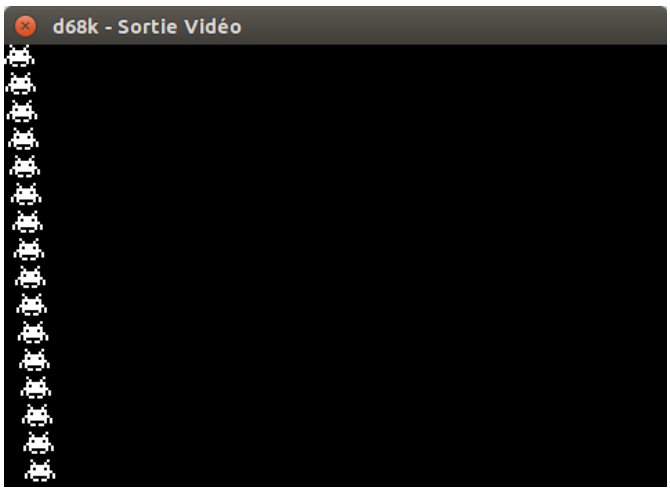
Le code du sous-programme **CopyBitmap** n'a pas besoin d'être modifié. En effet, celui-ci appelle **CopyLine** et la gestion du décalage se fera uniquement à partir de **CopyLine**. Par contre, il faudra bien prendre en compte la nouvelle entrée de **CopyBitmap** (**D0.W**) lors de son appel.

Modifiez votre sous-programme **CopyLine** afin de prendre en compte le décalage. Pour l'instant, vous n'avez pas besoin de vous servir du sous-programme **PixelToAddress**.

Utilisez le programme principal ci-dessous pour tester votre sous-programme :

```
Main                lea     InvaderB_Bitmap,a0
                    lea     VIDEO_START,a1
                    move.w  #0,d0
                    move.w  #1,d7

\loop               jsr     CopyBitmap

                    adda.w  #20*BYTE_PER_LINE,a1
                    addq.w  #1,d0
                    andi.w  #7,d0
                    bne     \loop

                    addq.l  #1,a1
                    dbra    d7,\loop

                    illegal
```

Capture d'écran du résultat attendu :



Étape 4

Pour finir, réaliser le sous-programme **PrintBitmap** qui affiche un bitmap à partir de coordonnées en pixels.

Entrées : **A0.L** = Adresse du bitmap.
D1.W = Abscisse en pixels où afficher le bitmap.
D2.W = Ordonnée en pixels où afficher le bitmap.

PrintBitmap est très proche de **CopyBitmap**. Ces deux sous-programmes permettent d'afficher un bitmap au pixel près. La position d'un bitmap est définie par une adresse et un décalage dans **CopyBitmap** alors qu'elle est définie par une abscisse et une ordonnée dans **PrintBitmap**. Ce dernier appellera donc **CopyBitmap** après avoir converti l'abscisse et l'ordonnée en une adresse et un décalage. Vous pourrez utiliser pour cela **PixelToAddress**.

Utilisez le programme principal ci-dessous pour tester votre sous-programme :

```
Main                lea     InvaderA_Bitmap,a0
                    move.w  #112,d1
                    move.w  #100,d2
                    jsr     PrintBitmap

                    lea     InvaderB_Bitmap,a0
                    move.w  #224,d1
                    jsr     PrintBitmap

                    lea     InvaderC_Bitmap,a0
                    move.w  #336,d1
                    jsr     PrintBitmap

                    lea     Ship_Bitmap,a0
                    move.w  #223,d1
                    move.w  #200,d2
                    jsr     PrintBitmap

                    illegal
```

Capture d'écran du résultat attendu :