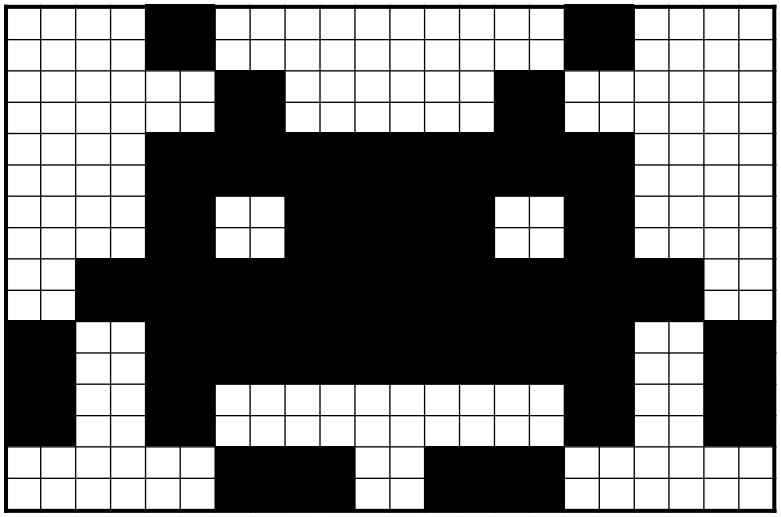
T.P. 10
Space Invaders (partie 3)

Étape 1

Maintenant que vous êtes un peu plus familiarisés avec le fonctionnement de la mémoire vidéo, nous pouvons commencer à dessiner notre premier envahisseur. Nous reprendrons au maximum les graphismes du jeu original. Le bitmap du premier envahisseur que nous souhaitons afficher est le suivant :



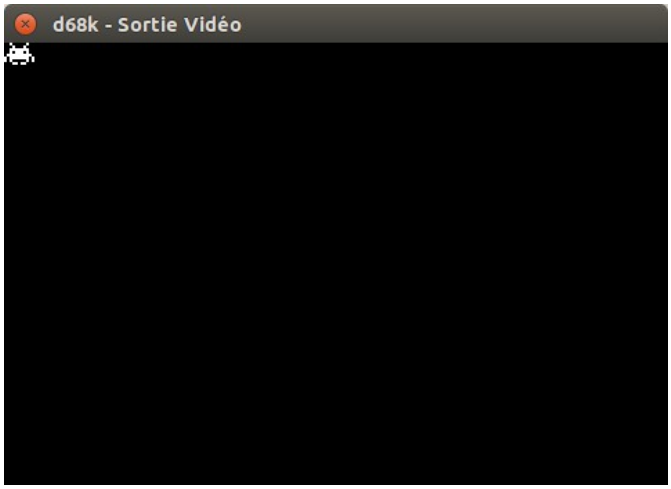
Notre bitmap est une matrice de points d'une taille de 22 × 16 pixels. Étant donné qu'une case mémoire ne contient qu'un octet, il faut découper la largeur en trois octets (3 × 8 = 24).

Notre matrice de points peut être stockée en mémoire directement à partir du code source à l'aide de la directive `DC.B` (*cf.* cours). De plus, nous pouvons saisir les données directement en binaire à l'aide du préfixe « % ». Vous trouverez ci-après ce qu'il faut saisir dans la partie contenant les données de votre code source. Si l'on regarde bien, on peut distinguer la forme de l'envahisseur dans les 0 et les 1.

```
Invader_Bitmap      dc.b    %00001100,%00000000,%11000000
                    dc.b    %00001100,%00000000,%11000000
                    dc.b    %00000011,%00000011,%00000000
                    dc.b    %00000011,%00000011,%00000000
                    dc.b    %00001111,%11111111,%11000000
                    dc.b    %00001111,%11111111,%11000000
                    dc.b    %00001100,%11111100,%11000000
                    dc.b    %00001100,%11111100,%11000000
                    dc.b    %00111111,%11111111,%11110000
                    dc.b    %00111111,%11111111,%11110000
                    dc.b    %11001111,%11111111,%11001100
                    dc.b    %11001111,%11111111,%11001100
                    dc.b    %11001100,%00000000,%11001100
                    dc.b    %11001100,%00000000,%11001100
                    dc.b    %00000011,%11001111,%00000000
                    dc.b    %00000011,%11001111,%00000000
```

Dans cette étape, vous devez simplement écrire quelques instructions dans le programme principal (pas de sous-programme). Ces instructions devront copier la matrice de points dans la mémoire vidéo afin d'afficher l'envahisseur en haut à gauche dans la fenêtre de sortie vidéo.

Capture d'écran du résultat attendu :



Étape 2

Le jeu devra afficher plusieurs types d'envahisseurs et bien d'autres bitmaps. Tous ces bitmaps ne posséderont pas nécessairement la même taille. Afin de faciliter la gestion de l'affichage, nous écrirons en mémoire la taille d'un bitmap juste avant sa matrice de points. La structure d'un bitmap sera donc la suivante :

- Largeur du bitmap en pixels (16 bits non signés) ;
- Hauteur du bitmap en pixels (16 bits non signés) ;
- Matrice de points du bitmap (taille variable).

```
                    ; ==============================
                    ; Données
                    ; ==============================

Invader_Bitmap      dc.w    22,16                           ; Largeur, Hauteur
                    dc.b    %00001100,%00000000,%11000000   ; Matrice de points
                    dc.b    %00001100,%00000000,%11000000
                    dc.b    %00000011,%00000011,%00000000
                    dc.b    %00000011,%00000011,%00000000
                    dc.b    %00001111,%11111111,%11000000
                    dc.b    %00001111,%11111111,%11000000
                    dc.b    %00001100,%11111100,%11000000
                    dc.b    %00001100,%11111100,%11000000
                    dc.b    %00111111,%11111111,%11110000
                    dc.b    %00111111,%11111111,%11110000
                    dc.b    %11001111,%11111111,%11001100
                    dc.b    %11001111,%11111111,%11001100
                    dc.b    %11001100,%00000000,%11001100
                    dc.b    %11001100,%00000000,%11001100
                    dc.b    %00000011,%11001111,%00000000
                    dc.b    %00000011,%11001111,%00000000
```

Supposons maintenant que l'adresse `Invader_Bitmap` soit dans **A0**. Nous pouvons en déduire que :

- la largeur du bitmap est située à l'adresse pointée par **A0** ;
- la hauteur du bitmap est située à l'adresse pointée par **A0 + 2** :
- la matrice de points du bitmap est située à partir de l'adresse **A0 + 4**.

Ainsi, si l'on souhaite accéder à ces données, nous pouvons utiliser le mode d'adressage indirect avec déplacement. Par exemple, si nous voulons stocker respectivement la largeur, la hauteur et l'adresse de la matrice de points dans les registres **D0.W**, **D1.W** et **A1.L**, nous pouvons procéder de la façon suivante :

```
                    move.w  (a0),d0         ; Largeur                 -> D0.W
                    move.w  2(a0),d1        ; Hauteur                 -> D1.W
                    lea     4(a0),a1        ; Adresse de la matrice   -> A1.L
```

Toutefois, afin de rendre plus explicite ces instructions, nous allons définir trois nouvelles constantes :

```
WIDTH               equ     0
HEIGHT              equ     2
MATRIX              equ     4

                    move.w  WIDTH(a0),d0    ; Largeur                 -> D0.W
                    move.w  HEIGHT(a0),d1   ; Hauteur                 -> D1.W
                    lea     MATRIX(a0),a1   ; Adresse de la matrice   -> A1.L
```

L'objectif de cette étape est de concevoir le sous-programme **`CopyBitmap`** qui copie la matrice d'un bitmap dans la mémoire vidéo. Cependant, dans le but de décomposer les différentes tâches, mais aussi de minimiser le nombre de lignes du sous-programme, vous devrez également concevoir deux autres sous-programmes qui seront appelés par **`CopyBitmap`** ; nous les appellerons **`CopyLine`** et **`PixelToByte`**.

Commençons par donner une brève description ainsi que les entrées-sorties de ces trois sous-programmes :

`CopyBitmap` : Copie la matrice de points d'un bitmap dans la mémoire vidéo.
Entrées : **A0.L** = Adresse du bitmap.
A1.L = Adresse vidéo où copier le bitmap.

`CopyLine` : Copie une ligne d'un bitmap dans la mémoire vidéo.
Entrées : **A0.L** = Adresse de la ligne du bitmap.
A1.L = Adresse vidéo où copier la ligne.
D3.W = Largeur de la ligne en octets.
Sortie : **A0.L** = Adresse de la prochaine ligne du bitmap.

`PixelToByte` : Convertit une taille en pixels en une taille en octets.
Entrée : **D3.W** = Taille en pixels à convertir (entier non signé inférieur ou égal à 480).
Sortie : **D3.W** = Taille en octets (nombre d'octets minimum pour contenir le nombre de pixels).

Commencez par écrire **`PixelToByte`**. À première vue, la taille en octets semble être la taille en pixels divisée par 8. Mais attention ceci ne fonctionne que lorsque la taille en pixels est un nombre multiple de 8. En effet, si la taille en pixels est de 8, 16, 24 ou 32 pixels, la taille en octets sera respectivement de 1, 2, 3 ou 4 octets. Par contre, si la taille en pixels est de 10, alors la taille en octets sera de 2 (il faut au moins 2 octets pour stocker 10 pixels : 8 pixels sur le premier octet et 2 pixels sur le second).

Un nombre binaire est multiple de 8 quand ses trois bits de poids faible sont à 0. Pour savoir si un nombre est multiple de 8, il faut donc simplement mettre ses bits de poids forts à 0 et conserver la valeur de ses bits de poids faible. Ceci s'appelle un masque et peut facilement être réalisé à l'aide d'un ET logique. Si le résultat du masque est nul, alors le nombre est multiple de 8.

Vous testerez votre sous-programme avec les valeurs suivantes :

```
Main                move.w  #3,d3
                    jsr     PixelToByte     ; D3.W = 1

                    move.w  #8,d3
                    jsr     PixelToByte     ; D3.W = 1

                    move.w  #9,d3
                    jsr     PixelToByte     ; D3.W = 2

                    move.w  #16,d3
                    jsr     PixelToByte     ; D3.W = 2

                    move.w  #20,d3
                    jsr     PixelToByte     ; D3.W = 3

                    move.w  #24,d3
                    jsr     PixelToByte     ; D3.W = 3

                    move.w  #31,d3
                    jsr     PixelToByte     ; D3.W = 4

                    move.w  #32,d3
                    jsr     PixelToByte     ; D3.W = 4

                    illegal
```

Réalisez ensuite le sous-programme **`CopyLine`**.

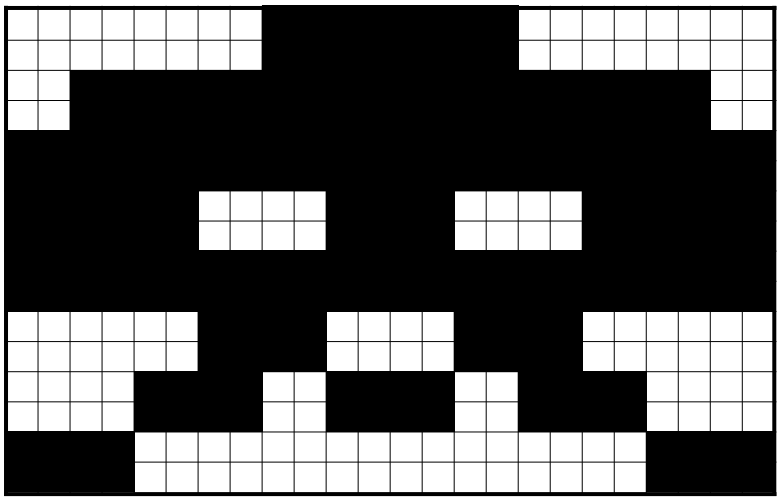
Pour terminer, réalisez le sous-programme **`CopyBitmap`**. Vous utiliserez la structure ci-après afin de tester votre sous-programme. Si tout va bien, l'envahisseur devrait s'afficher en haut à gauche de la fenêtre de sortie vidéo (de façon identique à l'étape précédente).

```
                    ; ==============================
                    ; 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

                    ; Bitmaps
                    ; ------------------------------

WIDTH               equ     0                           ; Largeur en pixels
HEIGHT              equ     2                           ; Hauteur en pixels
MATRIX              equ     4                           ; Matrice de points

                    ; ==============================
                    ; 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                lea     Invader_Bitmap,a0
                    lea     VIDEO_START,a1
                    jsr     CopyBitmap

                    illegal

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

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

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

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

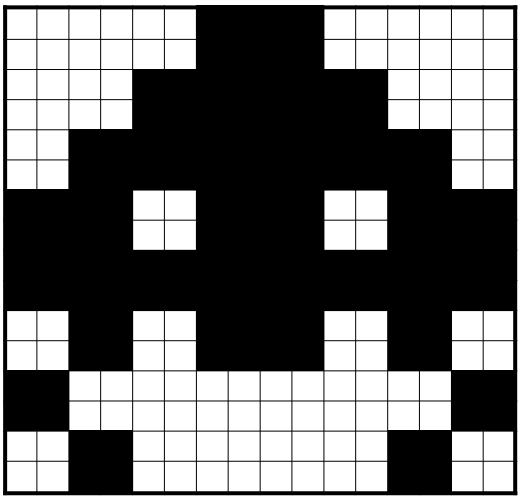
Étape 3

Dans cette étape, vous allez de nouveau tester le sous-programme **`CopyBitmap`**, mais cette fois sur différents types de bitmaps. Pour cela, nous allons ajouter deux envahisseurs ainsi que le vaisseau spatial du joueur. Les matrices de points sont les suivantes :

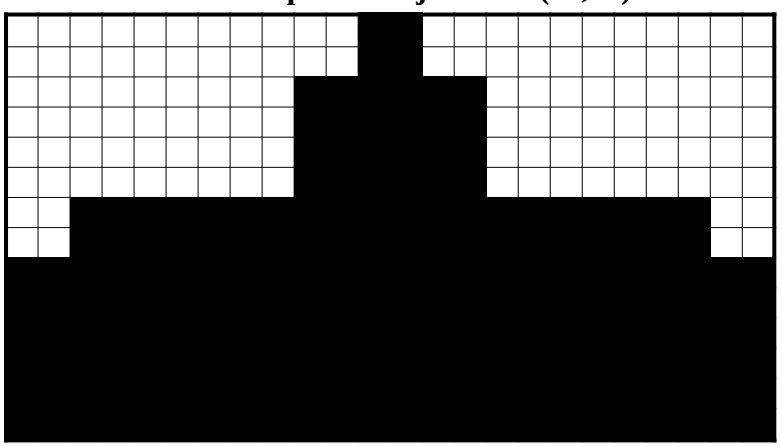
Envahisseur A – (24,16)



Envahisseur C – (16,16)



Vaisseau spatial du joueur – (24,14)



Nous nommerons les trois types d'envahisseurs *A*, *B* et *C*. L'envahisseur B, est celui que nous avons déjà affiché dans les étapes précédentes. Les envahisseurs *A* et *C* sont donnés ci-dessus.

Complétez la partie contenant les données de votre fichier source en fonction des matrices de points ci-dessus et du modèle ci-dessous :

```
                    ; ==============================
                    ; Données
                    ; ==============================

InvaderA_Bitmap     dc.w    24,16
                    dc.b    ; ...Saisir la matrice de points en binaire...
                    dc.b    ; ...

InvaderB_Bitmap     dc.w    22,16
                    dc.b    %00001100,%00000000,%11000000
                    dc.b    %00001100,%00000000,%11000000
                    dc.b    %00000011,%00000011,%00000000
                    dc.b    %00000011,%00000011,%00000000
                    dc.b    %00001111,%11111111,%11000000
                    dc.b    %00001111,%11111111,%11000000
                    dc.b    %00001100,%11111100,%11000000
                    dc.b    %00001100,%11111100,%11000000
                    dc.b    %00111111,%11111111,%11110000
                    dc.b    %00111111,%11111111,%11110000
                    dc.b    %11001111,%11111111,%11001100
                    dc.b    %11001111,%11111111,%11001100
                    dc.b    %11001100,%00000000,%11001100
                    dc.b    %11001100,%00000000,%11001100
                    dc.b    %00000011,%11001111,%00000000
                    dc.b    %00000011,%11001111,%00000000

InvaderC_Bitmap     dc.w    16,16
                    dc.b    ; ...Saisir la matrice de points en binaire...
                    dc.b    ; ...

Ship_Bitmap         dc.w    24,14
                    dc.b    ; ...Saisir la matrice de points en binaire...
                    dc.b    ; ...
```

Testez ensuite l'affichage de ces nouveaux bitmaps à l'aide du programme principal suivant :

```
Main                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
```

Capture d’écran du résultat attendu :