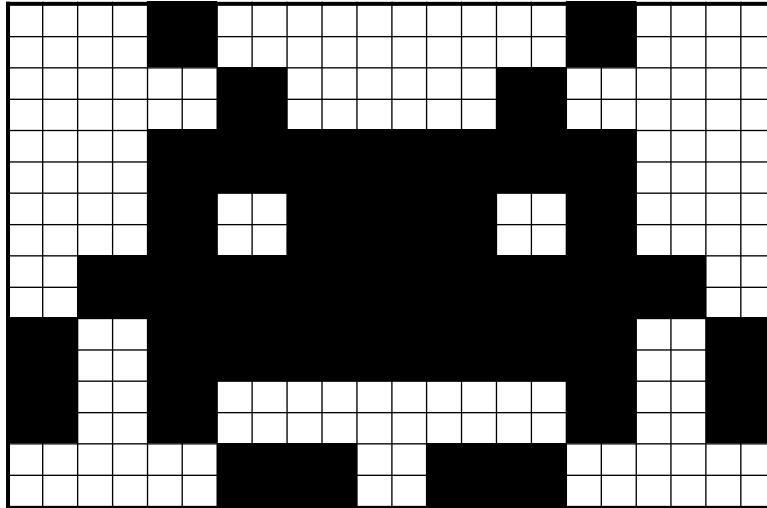


# Practical 10

## Space Invaders (Part 3)

### Step 1

Now that you have familiarized yourselves with the structure of the video memory, we can start to draw our first invader. We are going to reproduce the original graphics as much as possible. Here is the bitmap of the first invader we are going to display:



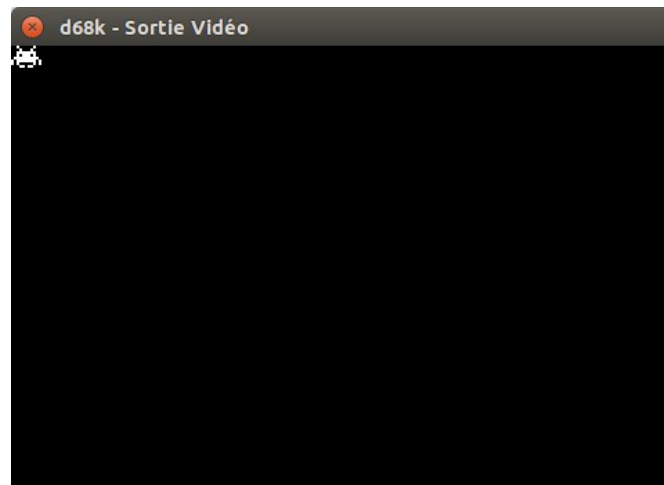
The size of the bitmap is  $22 \times 16$  pixels. Since a memory location holds a single byte, the width of the invader is 3 bytes ( $3 \times 8 = 24$ ).

The dot matrix can be stored in the memory by using the `DC.B` directive (see Chapter 1). Moreover, the prefix “%” can be used to key in data in its binary form. You can find below the instructions to be inserted as data into your source code. If you look carefully, you can make out the shape of the invaders among the 0s and 1s.

```
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
```

In this step, you simply have to write some instructions in the main program (no subroutine). These instructions must copy the dot matrix into the video memory in order to display the invader on the top left corner of the video output window.

Screenshot of the expected result:



## Step 2

Several types of invaders and some other bitmaps are going to be displayed during a game. All of these bitmaps can have different sizes. In order to simplify display handling, the size of a bitmap will be stored in the memory just before the dot matrix. Here is the structure of a bitmap:

- Width of the bitmap in pixels (unsigned 16-bit word).
- Height of the bitmap in pixels (unsigned 16-bit word).
- Dot matrix of the bitmap (variable size).

```

; =====
; Data
; =====

Invader_Bitmap  dc.w    22,16                ; Width, Height
                 dc.b    %00001100,%00000000,%11000000  ; Dot matrix
                 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

```

Now, let us assume that **A0** holds the **Invader\_Bitmap** address. We can deduce that:

- The width of the bitmap is pointed at by **A0**.
- The height of the bitmap is pointed at by **A0 + 2**.
- The dot matrix of the bitmap is pointed at by **A0 + 4**.

Therefore, the address register indirect mode with displacement can be used to access data. For instance, the following instructions copy the width, the height and the address of the dot matrix from the memory into the **D0.W**, **D1.W** and **A1.L** registers:

```

move.w (a0),d0      ; Width          -> D0.W
move.w 2(a0),d1     ; Height         -> D1.W
lea     4(a0),a1     ; Address of the matrix -> A1.L

```

However, in order to improve the readability of the code, it is advisable to define three new constants:

```

WIDTH      equ      0
HEIGHT     equ      2
MATRIX     equ      4

move.w WIDTH(a0),d0  ; Width          -> D0.W
move.w HEIGHT(a0),d1 ; Height         -> D1.W
lea     MATRIX(a0),a1 ; Address of the matrix -> A1.L

```

The purpose of this step is to write the **CopyBitmap** subroutine, which copies the dot matrix of a bitmap into the video memory. In order to keep the subroutine short and simple, we will break it down and start by writing another two subroutines: **CopyLine** and **PixelToByte**, which will then be called by **CopyBitmap**.

Here is a brief description of the subroutines and their I/O registers:

**CopyBitmap:** Copies the dot matrix of a bitmap into the video memory.

Inputs: **A0.L** = Address of the bitmap.

**A1.L** = Video address where the dot matrix must be copied.

**CopyLine:** Copies a bitmap line into the video memory with offset handling.

Inputs: **A0.L** = Address of the line.

**A1.L** = Video address where the line must be copied.

**D3.W** = Width of the line in bytes.

Output: **A0.L** = Address of the next line.

**PixelToByte:** Convert a size in pixels into a size in bytes:

Input: **D3.W** = Size in pixels to convert (unsigned integer lower than or equal to 480).

Output: **D3.W** = Size in bytes (minimum number of bytes that contains the number of pixels).

Start by writing **PixelToByte**. At first glance, the size in bytes seems to be the size in pixels divided by 8. But be careful! This is right only when the size in pixels is a multiple of 8. For instance, if the size in pixels is 8, 16, 24 or 32, the size in bytes is 1, 2 3 or 4 respectively. On the other hand, if the size in pixels is 10, the size in bytes is 2 (2 bytes are required to store 10 pixels: 8 pixels in the first byte and 2 pixels in the second one).

Test your subroutine with the following values:

```

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

```

Then, write the **CopyLine** subroutine.

Finally, write the **CopyBitmap** subroutine and use the following structure in order to test it. If everything is fine, the invader should be displayed on the top left corner of the video output window (in the same way as the previous step).

```

; =====
; Definitions of Constants
; =====

; Video Memory
; -----

VIDEO_START    equ    $ffb500        ; Starting address
VIDEO_WIDTH    equ    480            ; Width in pixels
VIDEO_HEIGHT   equ    320            ; Height in pixels
VIDEO_SIZE     equ    (VIDEO_WIDTH*VIDEO_HEIGHT/8) ; Size in bytes
BYTE_PER_LINE  equ    (VIDEO_WIDTH/8) ; Number of bytes per line

; Bitmaps
; -----

WIDTH          equ    0              ; Width in pixels
HEIGHT         equ    2              ; Height in pixels
MATRIX         equ    4              ; Dot matrix

; =====
; Vector Initialization
; =====

org            $0

vector_000     dc.l    VIDEO_START    ; Initial value of A7
vector_001     dc.l    Main          ; Initial value of the PC

; =====
; Main Program
; =====

org            $500

Main           lea      Invader_Bitmap,a0
               lea      VIDEO_START,a1
               jsr      CopyBitmap

               illegal

; =====
; Subroutines
; =====

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

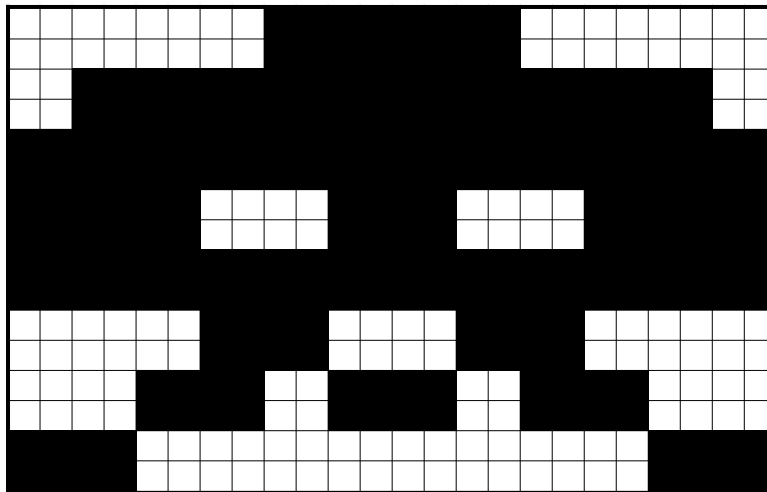
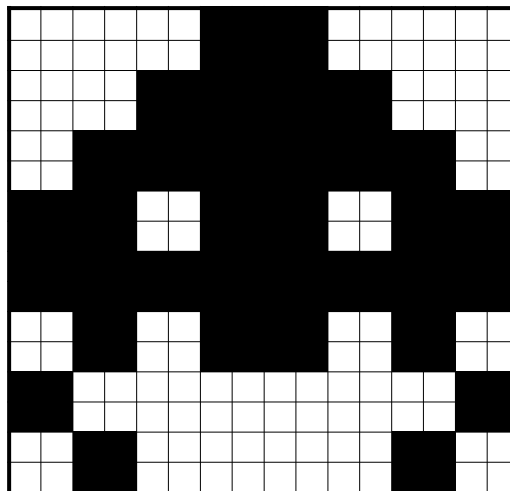
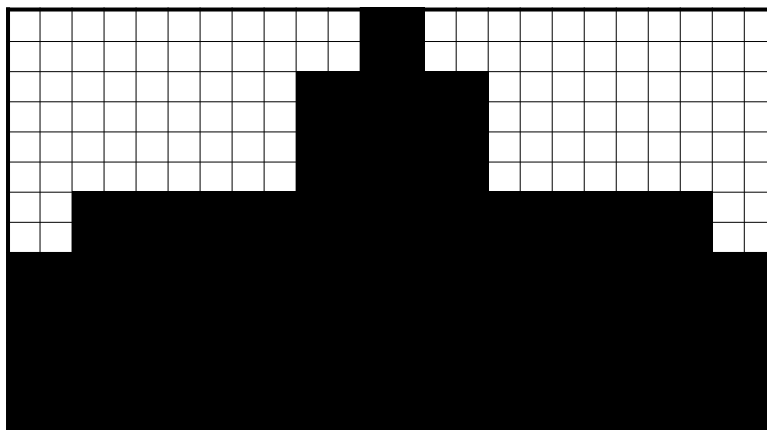
; =====
; Data
; =====

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

```

**Step 3**

In this step, you are going to test the **CopyBitmap** subroutine with different types of bitmaps. To do so, we are going to add two invaders and the spaceship of the player. The dot matrices are as follows:

**A Invader – (24,16)****C Invader – (16,16)****Spaceship of the player – (24,14)**

Let us call the three types of invaders *A*, *B* and *C*. The *B* invader is the one we have already displayed in the previous steps. The *A* and *C* invaders are given above.

Complete the “Data” part in your source file from the above dot matrices and the template below:

```

; =====
; Data
; =====

InvaderA_Bitmap    dc.w    24,16
                   dc.b    ; ...Insert the binary dot matrix...
                   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    ; ...Insert the binary dot matrix...
                   dc.b    ; ...

Ship_Bitmap        dc.w    24,14
                   dc.b    ; ...Insert the binary dot matrix...
                   dc.b    ; ...

```

Then, use the following main program in order to test the display of these new bitmaps:

```

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

```

Screenshot of the expected result:

