Practical 2
Space Invaders (Part 5)

Step 1

Let us consider the following main program that moves an invader from left to right.

```
Main                ; A0 points to an invader.
                    lea     InvaderA_Bitmap,a0

                    ; Place the invader on the middle left side.
                    move.w  #0,d1
                    move.w  #152,d2

\loop               ; Clear the screen and print the invader.
                    jsr     ClearScreen
                    jsr     PrintBitmap

                    ; Increment the abscissa of the invader.
                    addq.w  #1,d1

                    ; Branch to loop as long as the invader
                    ; has not reached the middle right side.
                    cmpi.w  #456,d1
                    blt     \loop

                    illegal
```

To begin with, write the **`ClearScreen`** subroutine that fully clears the screen (you can call the **`FillScreen`** subroutine).

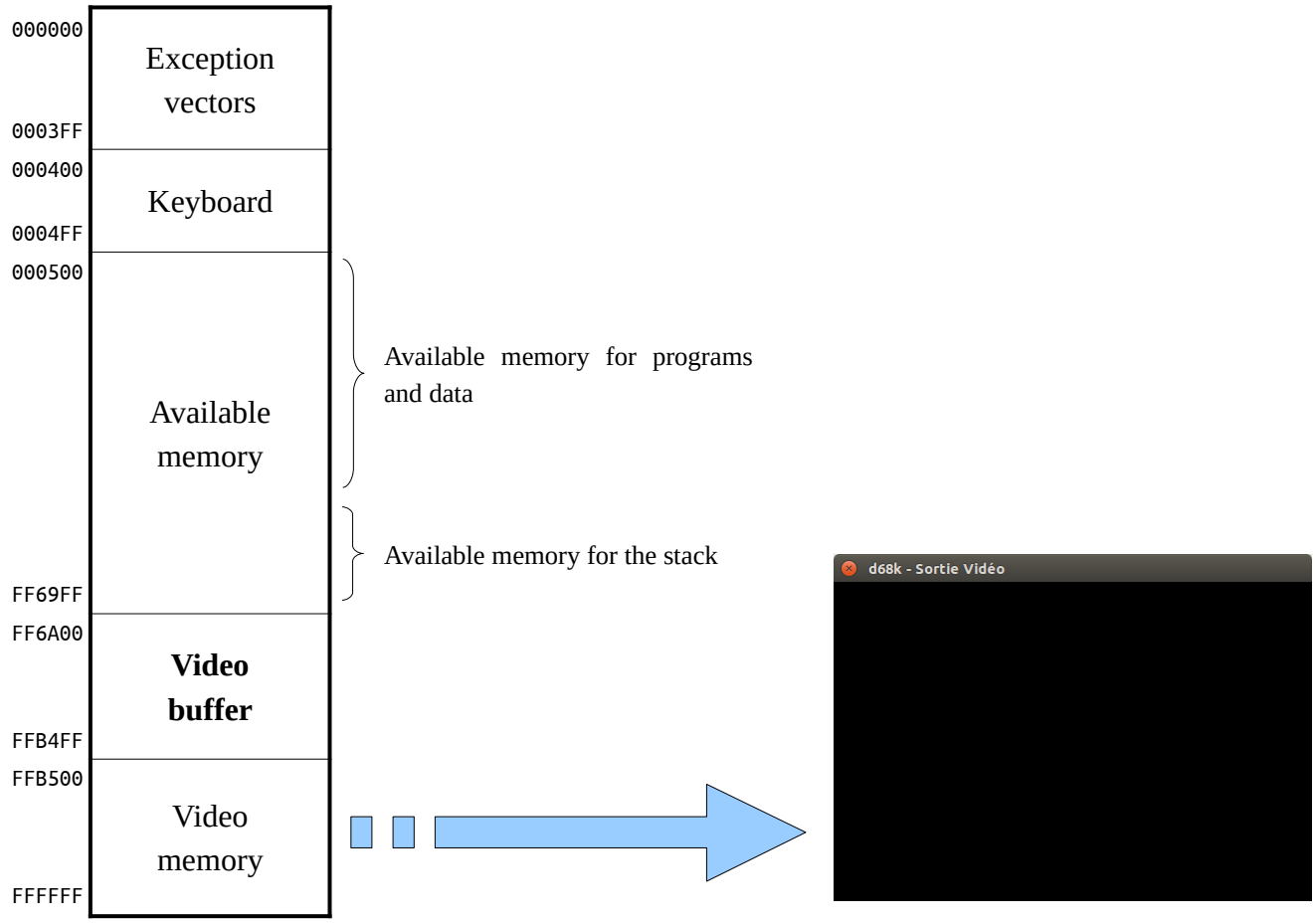
Now, execute the above main program and look at the invader moving. You can see that the screen is flickering. This effect is unwanted and must be sorted.

Step 2

In order to remove the flickering effect we are going to use the *double-buffering* technique. This technique consists of writing bitmaps in a video buffer, which will then be copied into the video memory.

A video buffer is a simple memory space that has the same size as the video memory. This buffer will be located just before the video memory.

Here is how the memory space of d68k is organized:



In order to use this video buffer, you should start by replacing all the occurrences of `VIDEO_START` by `VIDEO_BUFFER` in your source file.

Then, define the labels relative to the video memory as shown below:

```
                ; 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
VIDEO_BUFFER    equ     (VIDEO_START-VIDEO_SIZE)        ; Video buffer
```

The initial address of the stack must also be changed:

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

                    org     $0

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

Now, anything that must be displayed is written into the video buffer instead of the video memory. Your program operates as before except that nothing is displayed in the video output window.

So, you must copy the contents of the video buffer into the video memory in order to see the invader moving. Modify your main program in the following way:

```
Main                ; A0 points to an invader.
                    lea     InvaderA_Bitmap,a0

                    ; Place the invader on the middle left side.
                    move.w  #0,d1
                    move.w  #152,d2

\loop               ; Display the invader in the video buffer.
                    jsr     PrintBitmap

                    ; Copy the video buffer into the video memory.
                    ; (Then, the contents of the buffer are cleared.)
                    jsr     BufferToScreen

                    ; Increment the abscissa of the invader.
                    addq.w  #1,d1

                    ; Branch to loop as long as the invader
                    ; has not reached the middle right side.
                    cmpi.w  #456,d1
                    blt     \loop

                    illegal
```

Then, write the **`BufferToScreen`** subroutine that copies the video buffer into the video memory (copy long words one by one). Once a long word has been copied from the buffer into the video memory, it must be set to zero in the buffer (this way, the video buffer will be cleared before the next display).

Finally, run the above main program in order to test your subroutine. Check that the invader moves from left to right without flickering.

Step 3

We are going to define a sprite structure. A sprite is a small animated graphic element that can be manoeuvred around the screen. First, we are going to use sprites only for saving the location of the bitmaps displayed on the screen. Then, we are going to animate them.

Let us start by defining the structure of a sprite, which is made up of the five following fields:

Field	Size (bits)	Encoding	Description
STATE	16	Unsigned integer	Current display state of the sprite Only two possible values: HIDE = 0 or SHOW = 1
X	16	Signed integer	Abscissa of the sprite
Y	16	Signed integer	Ordinate of the sprite
BITMAP1	32	Unsigned integer	Address of the first bitmap
BITMAP2	32	Unsigned integer	Address of the second bitmap

The two bitmaps will be used to animate the sprite, but for the time being, let us ignore the second bitmap. In other words only the first bitmap will be used to display the sprite.

From this structure, we are going to define some new constants in order to manipulate sprites more easily. Add the following lines to the “Definitions of Constants” part of your source code.

```
                    ; Sprites
                    ; -----------------------------

STATE               equ     0                       ; Current display state
X                   equ     2                       ; Abscissa
Y                   equ     4                       ; Ordinate
BITMAP1             equ     6                       ; Bitmap 1
BITMAP2             equ     10                      ; Bitmap 2

HIDE                equ     0                       ; Hide the sprite
SHOW                equ     1                       ; Show the sprite
```

Also add your first sprite to the “Data” part of your source code. This sprite will be an invader placed on the middle left side of the video output window.

```
Invader             dc.w    SHOW                    ; Show the sprite
                    dc.w    0,152                   ; X = 0, Y = 152
                    dc.l    InvaderA_Bitmap         ; Bitmap to display
                    dc.l    0                       ; Unused
```

Now, let us assume that **A1** holds the `Invader` address. We can easily access the different fields of the sprites.

For instance:

```
                    move.w  STATE(a1),d0    ; Current display state   -> D0.W
                    move.w  X(a1),d1        ; X                       -> D1.W
                    move.w  Y(a1),d2        ; Y                       -> D2.W
                    movea.l BITMAP1(a1),a0  ; Address of the bitmap 1 -> A0.L
```

Write the **`PrintSprite`** subroutine that copies a sprite into the video buffer. Be careful, if the display state is HIDE, the sprite must not be copied.
Input: **A1.L** = Address of the sprite.

Use the main program below in order to test your subroutine:

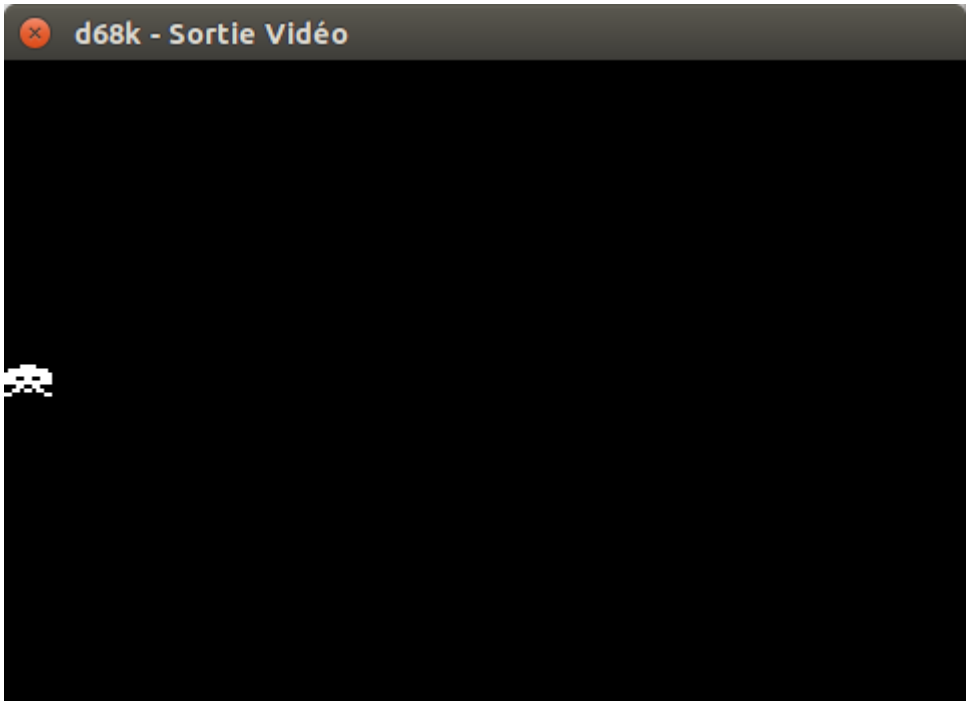
```
Main                ; A1 points to a sprite.
                    lea     Invader,a1

                    ; Copy the sprite into the video buffer.
                    jsr     PrintSprite

                    ; Copy the video buffer into the video memory.
                    jsr     BufferToScreen

                    illegal
```

Screenshot of the expected result:



Finally, set the display state to HIDE and run the main program again. Check that the sprite is not displayed.