Matériel :

1. /RST est relié par une résistance de tirage au niveau haut : le processeur n'est donc *pas*, par défaut, en état de réinitialisation.
2. 16 lignes d'adresse A0-A15 donc 65536 octets.
3. 8 bits de données D0-D7 donc 1 octet.
4. A14 est connectée à CE. Donc le bit 14 doit être à 1 : y1xx xxxx xxxx xxxx. Si y=0, nous avons la plage 4000 à 7FFF. Si y=1, nous avons la plage C000-DFFF. Ce résultat est cohérent avec l'organisation de la mémoire proposée à `http://gameboy.mongenel.com/dmg/asmmemmap.html`.
5. Aucune précaution n'est nécessaire, il suffit de connecter A0-Ax de la ROM aux broches du bus A0-Ax du processeur. Cependant, afin d'éviter tout conflit avec un accès aux périphériques embarqués sur la console, il faudra par ailleurs invalider l'accès à la ROM si la RAM interne est accédée. Dans ce cas nous plaçons une porte NOT entre la ligne d'adresse 14 et le signal de sélection de la ROM : si la RAM est activée par CE, alors la ROM ne l'est pas, et réciproquement.
6. nous devons adresser le convertisseur par une plage d'adresses occupée ni par la RAM, ni par la ROM, et câbler une combinaison des bits de poids fort du bus d'adresses pour déclencher le signal d'activation du convertisseur (CS#) dans le cas adéquat.
7. `*(unsigned char*)adresse=valeur;` avec `adresse` une valeur en dehors de la plage d'accès aux RAM, ROM et mémoire vidéo.

Assembleur :

1. à l'intérieur de la boucle : (9+4+5+7+7+6+6+6+10)=60 cycles d'horloge
2. 60 cycles à 4 MHz (250 ns) durent 15 μs. Au total pour BC données : BC$\times$15 + (9 + 4 + 11) $\times$ 0,25 μs
3. Entre deux rafraîchissements : 1/60 Hz = 16667 μs $\Rightarrow$ (16667-24$\times$0,25)/15=1110 octets. Nous sommes largement en-deça de la taille de l'écran de 160$\times$144 pixels=23040 bits=2880 octets.
4. Chaque instruction prend de nombreux cycles d'horloge sur Z80, contre 1 instruction par cycle d'horloge sur AVR8 sauf lorsqu'il faut vider le pipeline d'instructions.
5. décrémenter une valeur sur 16 bits n'affecte pas le drapeau Z (cf instruction DEC ss). Il faut donc tester les deux octets du mot de 16 bits formé par BC, d'abord en copiant B dans l'accumulateur qui est l'argument implicite de OR, et ensuite faire un OU de A (qui contient B) avec C : ce OU logique n'est nul que si B et C sont tous deux nuls, c'est à dire BC=0 qui indique la fin de la boucle. Le code proposé est une version décomposée de l'instruction LDIR qui fait toutes ces opérations en 21 cycles d'horloge. Noter que le gain d'un facteur de près de 5 en vitesse nous permet maintenant de mettre à jour l'écran entre deux raffraîchissements.

La capture d'écran de droite propose une exécution de ce code dans l'émulateur de ZXSpectrum (aussi sur un Z80, mais avec une carte mémoire un peu différente de la console Gameboy) `fuse-emulator` avec une compilation du programme au moyen des outils disponibles à `https://github.com/tstih/yx.git`. On y remarque l'affichage de pixels représentant la séquence binaire de 0 à 19, et en insert le contenu de la mémoire qui confirme que nous avons bien copié le contenu des mémoires à partir de la RAM vers la mémoire vidéo.

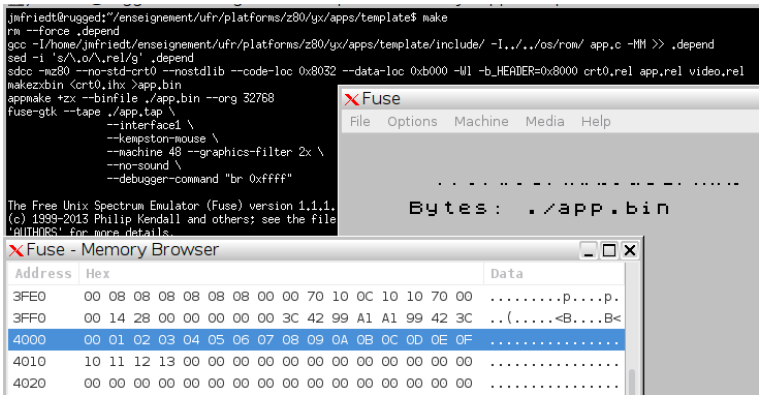

Figure 1: Exécution de l'exemple de copie entre blocs mémoire dans l'émulateur ZXSpectrum. Le motif affiché à l'écran représente le contenu de la mémoire (surligné en bleu en insert).

Bresenham :

1. $y = \frac{\Delta y}{\Delta x} \cdot x + B$ avec $\Delta x = x_1 - x_0$ et $\Delta y = y_1 - y_0$
2. $f(x) = y - \frac{\Delta y}{\Delta x} \cdot x - B$
3. $f(x+1) - f(x) = \frac{\Delta y}{\Delta x}$ si y ne change pas (la valeur de $f(x+1) - f(x)$ détermine si y change ou non)
4. la fonction de coût évolue par pas de $\frac{\Delta y}{\Delta x}$ qui n'est pas un entier. Nous exprimons donc l'algorithme sur $\Delta x \cdot (f(x+1) - f(x))$ qui évolue par pas de Δy qui est quant à lui un entier. À chaque pas de x, nous incrémentons la fonction de coût de Δy, et si la fonction de coût change de signe, alors nous lui soustrayons Δx. En pratique cela s'écrit

```
// http://hugi.scene.org/online/hugi20/cogb1.htm
// ./gbdk/build/ppc-unknown-linux2.2/gbdk/bin/lcc -o pixel pixel.c
#include <gb/gb.h>
#include <stdio.h>
#include <gb/drawing.h>

int main()
{ char x0=64,y0=68,x1=100,y1=88;
  char x,y,dx,dy;
  int D;
  color(DKGREY, WHITE, SOLID);
//  printf("Hello World\n");

// trace' d'une droite
  x=x0;y=y0; dx=x1-x0; dy=y1-y0;
  D=0;          // en (x0,y0), l'erreur est nulle !
  while (x<x1)
   { plot_point(x,y);
     D+=dy;  // on exprime erreur*dx
     if (D>0) {y+=1;D-=(dx);} // si colonne change, retranche dx
     x++;
   }
// trace' d'un cercle
  x=dx; y=0;  // x=rayon
  D=1-x;    // D @ x=r, y=0
  while(y<=x)
  { plot_point(x+x0,y+y0);
    y++;
    if (D<=0) D+=2*y+1;
    else {x--;D+=2*(y-x)+1;}
  }
  while(1){}  //loop forever
}
```

5. À chaque incrément nous ajoutons Δy qui vaut au plus 160. Cette opération est répétée au plus 160 fois donc $160 \times 160 = 25600$ qui tient sur 16 bits. La variable doit donc être un short puisqu'un char ne peut supporter une telle variable dans le pire cas, et un long est un gaspillage de ressources.