Première Compilation

En ASM Z80 sur GameBoy

[**Introduction**](#_9f1hoxyu59a0) **2**

[**Les Outils**](#_6vgsiibj4qsr) **2**

[Éditeur de Texte](#_2ygbz36mk8sr) 2

[L’assembleur](#_p45lnl7xx50r) 2

[L’émulateur](#_4y8wug5ql5sk) 3

[**Mise en place du projet**](#_23eh02uffpqc) **3**

[**Première compilation**](#_tjsu4p5wg7j1) **4**

[**Le linkage**](#_jkkj3d6yhgmy) **7**

[**On se simplifie la vie**](#_iem8h1w51pnq) **7**

[**Explication du code**](#_vva97zmtm438) **8**

[Directive de compilation](#_axb2kk5661gs) 8

[L’assembleur Z80](#_n1ew87ypl3gu) 8

[Débugger](#_42c35d2ep63a) 9

[**Ressources**](#_3yyfxod02jxg) **12**

[Sur WLA-DX :](#_s8a34lovn52m) 12

[Sur le Z80](#_di4xazags9dk) 12

[Sur GameBoy](#_2oi5q7dfm2vw) 12

# Introduction

On va ici apprendre a utiliser plusieurs outils pour pouvoir compiler du code Assembleur Z80 pour la console Gameboy Noir & Blanche. On abordera pas le langage assembleur en détail ici, mais juste le nécessaire pour faire notre première compilation et que tout se passe facilement sans trop galérer comme je l’ai fait.

Avant tout une petite présentation de la Gameboy histoire de savoir de quoi l’on parle:

<https://fr.wikipedia.org/wiki/Game_Boy>

<https://blog.flozz.fr/2019/09/18/developpement-gameboy-1bis-re-hello-world/>

<https://www.youtube.com/watch?v=NSSrd1gr0LE>

# Les Outils

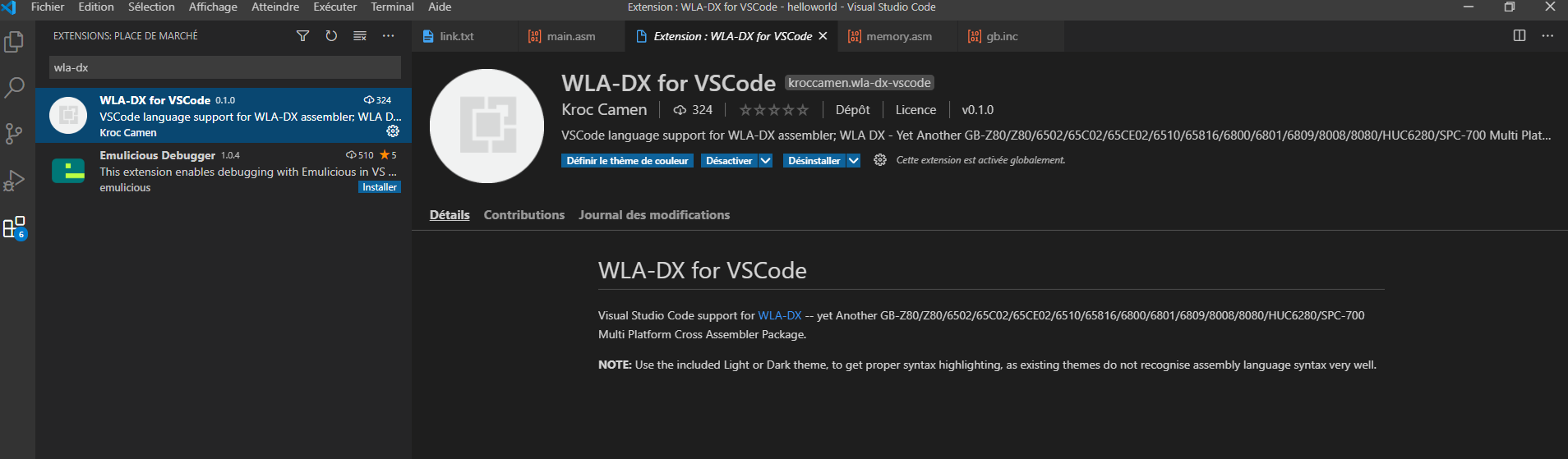
Pour développer sur gameboy on va utiliser plusieurs outils, et faire de la ligne de commande. J’ai choisi la solution la plus simple, libre à vous de la compliquer si vous voulez.

### Éditeur de Texte

Tout d’abord il faut un éditeur de texte pour écrire notre code. Je vous recommande VSCode, mais libre à vous de choisir ce que vous voulez.

Si vous ne l’avez pas vous pouvez le télécharger ici : <https://code.visualstudio.com/download>

Pour bénéficier de l’aide de vS-Code il faut aussi installer l’extension WLA-DX



### L’assembleur

Ensuite il va nous falloir également un assembleur, c'est-à-dire un programme qui va être capable de traduire ce que l’on écrit dans l'éditeur de texte en langage machine, une suite de 0 et de 1. C’est la seule chose que le microprocesseur est capable de comprendre, mais pour nous c’est plus compliqué.

J’ai choisi WLA-GB qui compile en Z80 pour Gameboy, mais il y a plusieurs versions de compilateur donc si vous voulez développer sur d'autres vieilles machines vous ne serez pas trop dépaysé.

Pour le télécharger il faut aller ici : <https://www.villehelin.com/wla-win32.html>

Sinon voila le site :

<https://www.villehelin.com/wla.html>

Et récupérer dans le répertoire …. deux fichiers

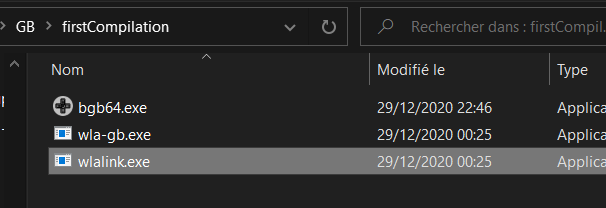
* wla-gb.exe
* wlalink.exe

### L’émulateur

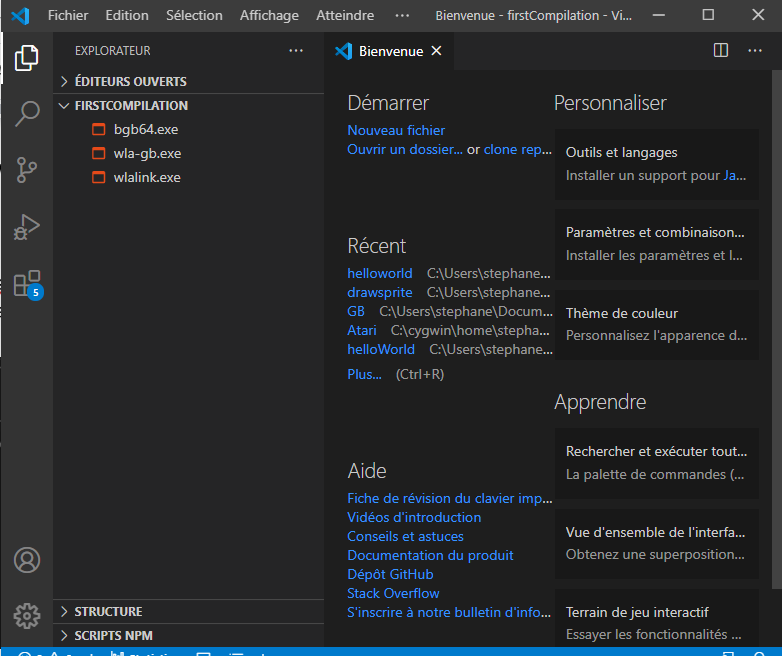
A défaut d’avoir le kit de développement de Nintendo pour la console, il va falloir passer par un émulateur. Ici je vais utiliser Bgb il offre plein d’options pour le développement. Pour le télécharger c’est ici : <https://bgb.bircd.org/>

# Mise en place du projet

On va déjà créer un dossier pour mettre nos fichiers, pour ce projet on va le nommer firstCompiltation et dedans on va rajouter dans la racine le compilateur et l'émulateur. on doit donc avoir maintenant ça dans notre répertoire :



Ensuite on lance VSCode et on va ouvrir notre dossier



On voit bien dans l'arborescence nos fichiers. On va alors créer notre fichier assembleur. On va dans l'arborescence des fichiers et on fait un clique droit et ‘nouveau fichier’. On le nomme main.s

Voilà on a le minimum pour écrire de l’assembleur et le compiler. C’est ce qu’on va faire maintenant.

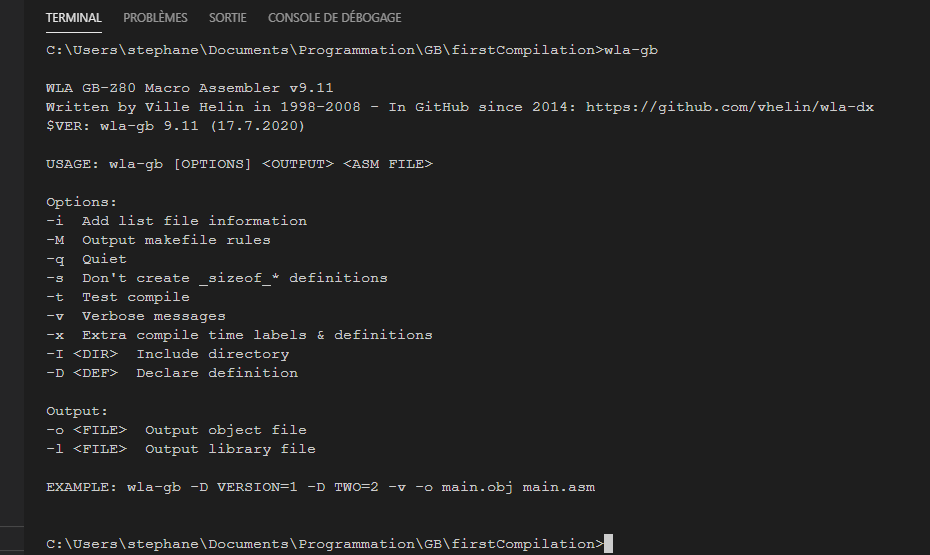
# Première compilation

On va déjà ouvrir un terminal pour pouvoir exécuter l’assembleur.

On fait ‘CTRL + ù’ ou onglet ‘Affichage’ et Terminal

Et dedans on écrit notre première ligne de code, le nom du compilateur : wla-gb et on valide

Voilà le résultat



La console nous affiche un message du compilateur qui nous explique comment l’utiliser.

USAGE: wla-gb -[iMqtvx]{lo} [DEFINITIONS] <ASM FILE> [OUTPUT FILE]

Nous on va avoir besoin d’un fichier objet donc c’est l’option o qui nous intéresse, ensuite il faut le fichier ou est contenu notre code assembleur et enfin le nom du fichier objet en \*.o, ce qui donne :

wla-gb -o main.o main.s

On valide et on a notre premier message d’erreur, c’est super 🤣

En fait c’est normal, il nous dit qu’il lui manque des informations pour compiler. Sur les machines Nintendo il faut fournir ce que l’on appelle un header qui contient des informations sur la cartouche et certaines sont obligatoires. Je vais donc vous fournir un exemple de header minimal pour faire notre première compilation.

On retourne dans notre main.s et on écrit ces lignes de code :

; Header pour Gameboy

.ROMDMG ;Pas de features CGB

.NAME "MONJEU" ;Nom du ROM inscrit dans le header

.CARTRIDGETYPE 0 ;ROM only

.RAMSIZE 0

.COMPUTEGBCHECKSUM ;WLA-DX écrira le checksum lui-même (nécessaire sur une vraie GB)

.COMPUTEGBCOMPLEMENTCHECK ;WLA-DX écrira le code de verif du header (nécessaire sur une vraie GB)

.LICENSEECODENEW "00" ;Code de license Nintendo, j'en ai pas donc...

.EMPTYFILL $00 ;Padding avec des 0

.MEMORYMAP

SLOTSIZE $4000

DEFAULTSLOT 0

SLOT 0 $0000

SLOT 1 $4000

.ENDME

.ROMBANKSIZE $4000 ;Deux banks de 16Ko

.ROMBANKS 2

.BANK 0 SLOT 0

.ENUM $C000

; ici on déclare les variables

.ENDE

.ORG $0040

reti

.ORG $0100

nop

jp start ;Entry point

.ORG $0104

;Logo Nintendo, obligatoire

.db $CE,$ED,$66,$66,$CC,$0D,$00,$0B,$03,$73,$00,$83,$00,$0C

.db $00,$0D,$00,$08,$11,$1F,$88,$89,$00,$0E,$DC,$CC,$6E,$E6

.db $DD,$DD,$D9,$99,$BB,$BB,$67,$63,$6E,$0E,$EC,$CC,$DD,$DC

.db $99,$9F,$BB,$B9,$33,$3E

.org $0150

start:

jp start

Maintenant on retente notre ligne de code, on peut faire un appuie sur la flèche haut du pavé numérique pour avoir la dernière ligne de commande exécuter.

wla-gb -o main.o main.s

en valide et cette fois aucun message d’erreur, c’est bon signe 🥳

On voit aussi que l’on a un nouveau fichier dans notre répertoire, c’est le main.o.

C’est bien beau mais j’en fait quoi de ce main.o ?

Il va falloir le linker pour en faire une cartouche que l'émulateur pourra comprendre et lancer.

# Le linkage

Il va falloir faire maintenant notre cartouche. Pour ça WLA a besoin d’un fichier de linkage contenant les fichiers objets pour faire notre cartouche. Nous on a que le main.o mais c’est possible d’en mettre plusieurs.

On va donc faire un nouveau fichier et le nommer link.txt et dedans écrire cela

[objects]

main.o

On retourne a notre console (CTRL + ù) et cette fois on écrit wlalink et on valide

On a cette fois un message nous expliquant comment le linker fonctionne. Nous on veut un fichier de sortie, une Rom donc c’est -r et on respecte l’ordre des fichiers ce qui nous donne :

wlalink -r link.txt game.gb

et normalement on obtient un nouveau fichier game.gb qui est la rom de notre cartouche Gameboy.

On peut maintenant lancer l’émulateur avec cette cartouche avec cette ligne de commande :

bgb64 game.gb

Et magie notre cartouche se lance et on est devant le logo de Nintendo 🥳

On a réussi à lancer notre propre code sur un émulateur, c’est déjà une belle victoire

# On se simplifie la vie

On va pas se voiler la face, si faut a chaque fois écrire les lignes de commandes pour compiler ça va être vite casse couille. On va se faire un fichier .bat et le lancer c’est lui qui lancera les lignes de code. On crée un nouveau fichier que l’on nomme start.bat et on écrit les lignes de commande dedans.

wla-gb -o main.o main.s

wlalink -r link.txt game.gb

bgb64 game.gb

et maintenant pour tout lancer on fera juste dans la console start.bat

C’est pas cool ça ?

# Explication du code

Ce qu’il faut retenir dans cet exemple c’est que dans ce bout de code nous avons deux types d’instruction.

### Directive de compilation

C’est des instructions qui vont dire au compilateur de faire des choses. Mais nous ne les retrouverons pas dans notre code finale et cela ne sera pas interpréter par le microporcesseur. C’est uniquement pour nous facilité la tache pour programmer. A savoir aussi , les directives ne sont valable que pour un assembleur( Le logiciel qui compile le code), ça peut changer suivant le logiciel.

C’est globalement toutes les instructions qui commencent par un . ex :

.org $0150

Ici on dit au compilateur que le code qui suit sera placé à l’adresse 150

Pour en savoir plus il faut lire la doc du compilateur ici : <https://wla-dx.readthedocs.io/_/downloads/en/latest/pdf/>

### L’assembleur Z80

Ca va être rapide car y a que deux instructions a voir, j’ai pas trop compliquer la chose pour pas vous faire fuir. Voilà les instructions :

nop

jp start

Là il s’agit bien d’instruction Z80

le NOP est une instruction qui ne fait rien, mais consomme un cycle de CPU quand même.

Doc ici : <https://fr.wikibooks.org/wiki/Programmation_Assembleur_Z80/Jeu_d_instructions#NOP>

et jp start

C’est un saut vers un label (directive compilateur) qui sert à faire un marqueur sur notre code. Il va être remplacé par l’adresse mémoire où il est défini. Dans notre cas c’est à cette partie du code :

.org $0150

start:

jp start

On retrouve notre directive .org qui place notre code à l’adresse 150

Puis on définit le label start, ne pas oublier les : le compilateur va remplacer start par l’adresse où il est utilisé, donc l’instruction jp start sera remplacé par jp 150.

Ensuite nous avons notre saut vers start, ce qui veut dire que nous avons fait une boucle infinie.

Le Cpu va lire notre code de façon linéaire, de haut en bas.

Sur la Gameboy il commence à lire le code à partir de l’adresse 100

Or si l’on relie notre code, on voit cela :

.ORG $0100

nop

jp start

Donc il va lire un Nop et ensuite un jp start, donc il va faire un saut a l’adresse où a été défini start, soit dans notre exemple a 150 et comme on l’a vu plus haut, il va faire une boucle infinie.

La doc sur jp : <https://fr.wikibooks.org/wiki/Programmation_Assembleur_Z80/Jeu_d_instructions#JP>

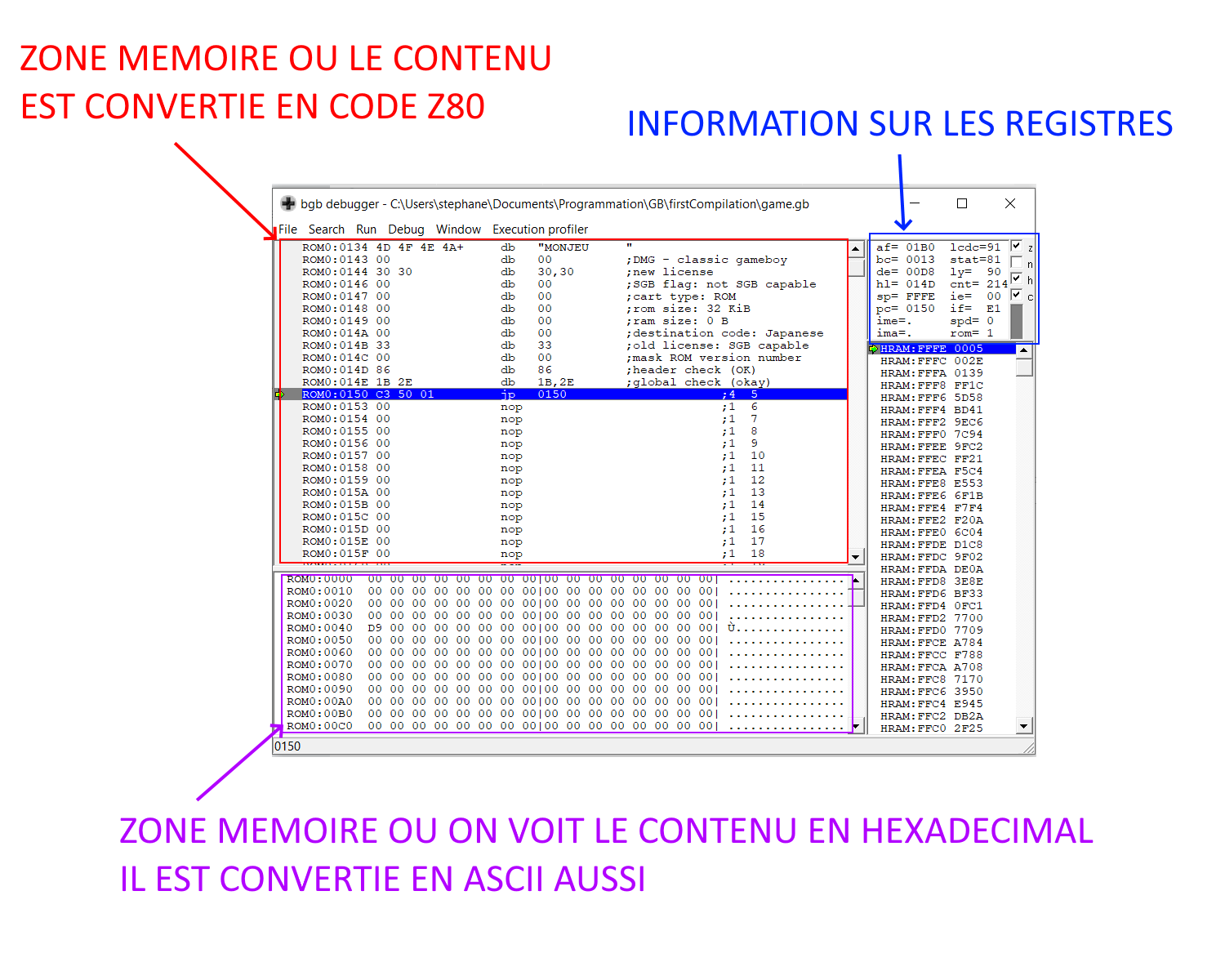
Profitez en car c’est la dernière fois que je donne le lien direct sur l’instruction, faudra le faire vous même ensuite.

Pour mieux comprendre ça on va utiliser le debugger dans l’émulateur pour comprendre comment ça marche.

### Débugger

On lance la compilation avec start.bat et on se retrouve dans l’émulateur

On fait un clique droit et -> Other -> debugger



Maintenant on va sur l’onglet Run -> Reset (numpad\*)

et notre pointeur vers pointe à l’adresse 100

On appuis sur F3 ou Run -> Step Over

A chaque fois que l’on appuiera sur F3 on exécutera le code ou se trouve la flèche verte

On pourra voir ce que fait le CPU étape par étape. Dans notre cas voilà ce qu’il va faire :



On retrouve bien notre nop que l’on a ecrit dans notre code a l’adresse 100

puis F3 pour continuer



Là on a notre jp start qui a été remplacé par son adresse

puis F3 pour continuer



On a bien fait notre saut à l’adresse 150 ici ROM0 : 0150

et maintenant on peux appuyer sur F3 frénétiquement pour voir que l’on a bien notre boucle infini.

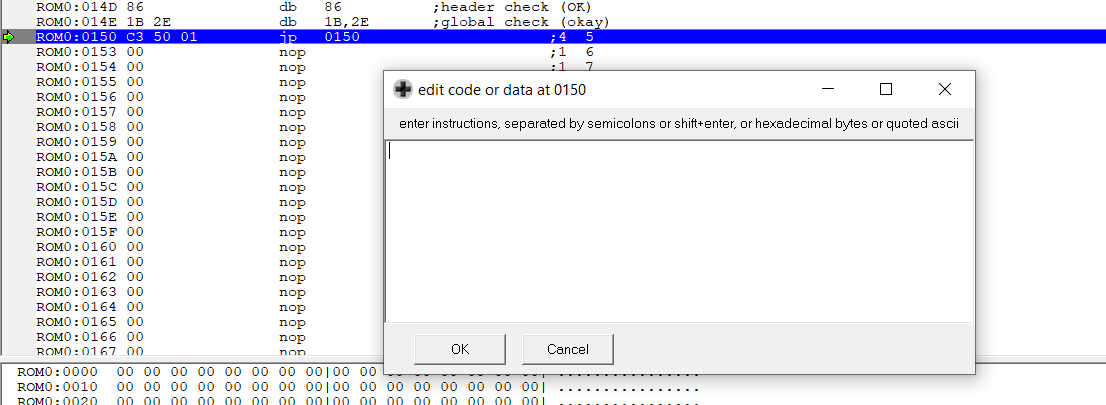
Essayer de placer le start a une autre adresse, bidouillé pour mieux comprendre comment ça marche, utilisé la doc pour explorer tout ça. C’est important de comprendre cette notion avant de continuer.

On peut aussi modifier le code en mémoire directement.

par exemple placé vous sur la zone mémoire 150 et faite :

Click droit -> Modifie Code/Date

On a cette fenetre qui s’ouvre :



On va y injecter du code Z80

ecrivez ceci

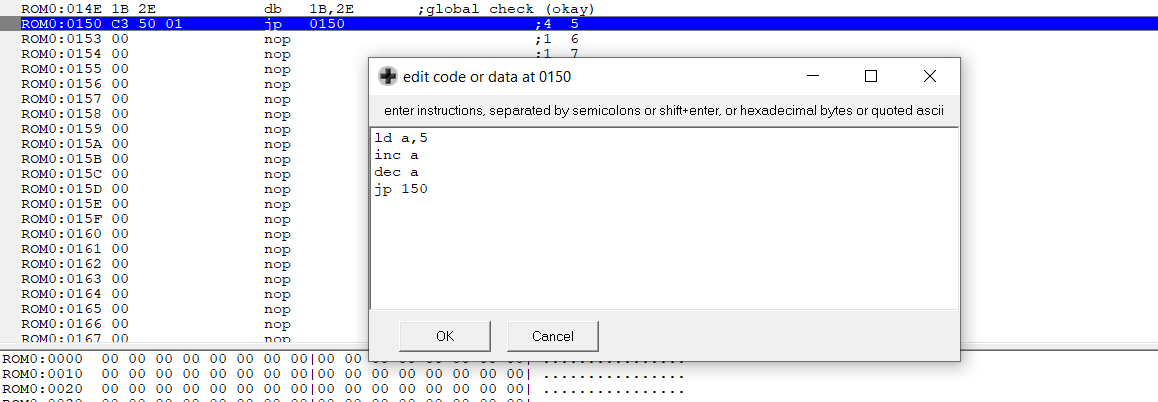
ld a,5

inc a

dec a

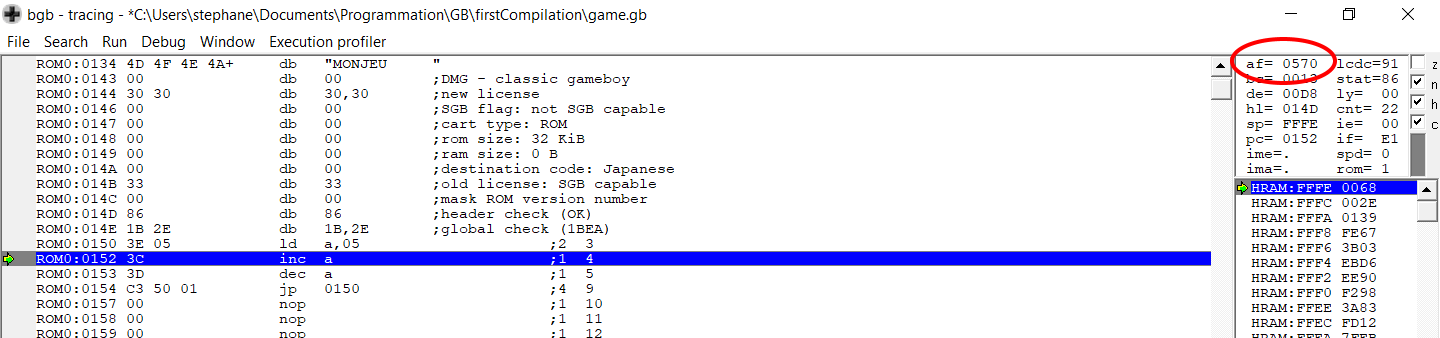
jp 150

Pour allez a à ligne il faut faire SHIFT + Enter, c’est ecrit dans la fenêtre mais au cas où vous l’auriez pas lu je le dit. Ce qui donne :



On valide et on a maintenant nos lignes sont rajoutées dans la mémoire, on peut tester notre code.

Regarder dans la fenêtre des registres au registre AF.



AF est un registre sur 16 bits, composé du registre A (8 Bits) et F(8 Bits) aussi.

Quand on fait un ld a,5 on affecte la valeur 5 dans le registre A donc ça a bien fonctionné.

J’en dit pas plus sur le reste du code a vous de lire la documentation sur les instructions inc et dec et à vous de tester avec F3 pour voir ce que ça fait.

Bon si vous êtes arrivée ici et que vous êtes excité comme un fou et que vous ne pouvez pas dormir car c’est super génial alors vous pourrez continuer sur le prochain cours.🥳

Si vous trouvez ça nul alors adieu.

Bon code pour les vikings

# Ressources

### Sur WLA-DX :

<https://www.villehelin.com/wla.html>

Documentation : <https://wla-dx.readthedocs.io/_/downloads/en/latest/pdf/>

### Sur le Z80

<https://fr.wikibooks.org/wiki/Programmation_Assembleur_Z80>

### Sur GameBoy

les "pandocs" c’est la bible hardwars de la Gameboy <https://gbdev.io/pandocs/>

le manuel de la Gameboy : <http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf>

tuto Hello world <https://eldred.fr/gb-asm-tutorial/hello-world.html>

Exemple d’un Pong : <http://furrtek.free.fr/?a=gbasm>

Plein de ressource pour l’asm : <https://www.chibiakumas.com/z80/Gameboy.php>

Cours en C sur Gameboy, avec pleins d’infos : <https://blog.flozz.fr/2019/09/18/developpement-gameboy-1bis-re-hello-world/>

Github de dev sur GB: <https://github.com/gbdev>

tuto sur la Gameboy : <https://gb-archive.github.io/salvage/tutorial_de_ensamblador/tutorial_de_ensamblador%20%5BLa%20decadence%5D.html>